Technical Support

<table>
<thead>
<tr>
<th>Location</th>
<th>Telephone</th>
<th>e-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>North America</td>
<td>+1 248 614 2425</td>
<td><a href="mailto:pbssupport@altair.com">pbssupport@altair.com</a></td>
</tr>
<tr>
<td>China</td>
<td>+86 (0)21 6117 1666</td>
<td><a href="mailto:es@altair.com.cn">es@altair.com.cn</a></td>
</tr>
<tr>
<td>France</td>
<td>+33 (0)1 4133 0992</td>
<td><a href="mailto:Francesupport@altair.com">Francesupport@altair.com</a></td>
</tr>
<tr>
<td>Germany</td>
<td>+49 (0)7031 6208 22</td>
<td><a href="mailto:hwsupport@altair.de">hwsupport@altair.de</a></td>
</tr>
<tr>
<td>India</td>
<td>+91 80 66 29 4500</td>
<td><a href="mailto:pbs-support@india.altair.com">pbs-support@india.altair.com</a></td>
</tr>
<tr>
<td>Italy</td>
<td>+39 0832 315573</td>
<td><a href="mailto:support@altairengineering.it">support@altairengineering.it</a></td>
</tr>
<tr>
<td></td>
<td>+39 800 905595</td>
<td></td>
</tr>
<tr>
<td>Japan</td>
<td>+81 3 5396 2881</td>
<td><a href="mailto:pbs@altairjp.co.jp">pbs@altairjp.co.jp</a></td>
</tr>
<tr>
<td>Korea</td>
<td>+82 31 728 8600</td>
<td><a href="mailto:support@altair.co.kr">support@altair.co.kr</a></td>
</tr>
<tr>
<td>Scandinavia</td>
<td>+46 (0)46 286 2050</td>
<td><a href="mailto:support@altair.se">support@altair.se</a></td>
</tr>
<tr>
<td>UK</td>
<td>+44 (0)1926 468 600</td>
<td><a href="mailto:pbssupport@uk.altair.com">pbssupport@uk.altair.com</a></td>
</tr>
</tbody>
</table>

This document is proprietary information of Altair Engineering, Inc.
# Table of Contents

About PBS Documentation .................................................. ix

1 New Features ................................................................. 1
  1.1 New Features in PBS 11.1 .......................................... 2
  1.2 Changes in Previous Releases ...................................... 2
  1.3 Deprecations ........................................................... 10
  1.4 Backward Compatibility ............................................. 13

2 Configuring the Server and Queues .................................. 15
  2.1 The Server .............................................................. 15
  2.2 Queues ................................................................. 19

3 Configuring MOMs and Vnodes ........................................ 41
  3.1 Vnodes: Virtual Nodes .............................................. 41
  3.2 MOMs ................................................................. 48
  3.3 Files and Directories Used by MOM ............................... 50
  3.4 MOM and Vnode Configuration Files ............................... 52
  3.5 How to Configure MOMs and Vnodes ............................... 56
  3.6 Configuring MOM and Vnode Features ............................. 65
# Table of Contents

## 4 Scheduling

4.1 Chapter Contents ........................................... 71
4.2 Scheduling Policy Basics ................................. 74
4.3 Choosing a Policy ......................................... 100
4.4 The Scheduler ............................................ 117
4.5 Using Queues in Scheduling ............................... 132
4.6 Scheduling Restrictions and Caveats ...................... 133
4.7 Errors and Logging ...................................... 135
4.8 Scheduling Tools ........................................ 135

## 5 PBS Resources

5.1 Introduction .............................................. 317
5.2 Chapter Contents ....................................... 318
5.3 Terminology ............................................. 320
5.4 Categories of Resources ................................. 324
5.5 Resource Types .......................................... 329
5.6 Behavior of Resources .................................. 330
5.7 How to Set Resource Values ............................. 332
5.8 Overview of Ways Resources Are Used ............... 335
5.9 Resources Allocated to Jobs and Reservations ....... 336
5.10 Using Resources to Track and Control Allocation ... 347
5.11 Using Resources for Topology and Job Placement .... 350
5.12 Using Resources to Prioritize Jobs ..................... 350
5.13 Using Resources to Restrict Server, Queue Access ... 351
5.14 Custom Resources ...................................... 352
5.15 Managing Resource Usage ............................... 409
5.16 Where Resource Information Is Kept .................. 446
5.17 Viewing Resource Information .......................... 453
5.18 Resource Recommendations and Caveats ............. 456

## 6 Hooks

6.1 Chapter Contents ....................................... 461
6.2 Introduction to Hooks ................................... 463
# Table of Contents

6.3 Definitions .................................................. 464  
6.4 How Hooks Can Be Used. ................................. 464  
6.5 How Hooks Work. ........................................... 468  
6.6 Creating Hooks .............................................. 473  
6.7 Configuring Hooks ........................................... 479  
6.8 Viewing Hook Information ............................... 485  
6.9 Interface to Hooks .......................................... 487  
6.10 Examples ....................................................... 543  
6.11 Errors, Logging, and Troubleshooting ............... 555  
6.12 Advice and Caveats ........................................ 563  
6.13 See Also ....................................................... 576  

7 Metascheduling Using HPC Basic Profile 579  
7.1 Definitions ..................................................... 579  
7.2 Using HPC Basic Profile .................................... 580  
7.3 Configuring PBS for HPCBP ............................... 585  
7.4 Viewing Status of Nodes ................................. 592  
7.5 Errors, Logging, and Troubleshooting ................ 593  
7.6 Restrictions and Requirements ......................... 594  
7.7 Advice and Caveats .......................................... 595  
7.8 See Also ....................................................... 597  

8 Provisioning ..................................................... 599  
8.1 Introduction .................................................... 599  
8.2 Definitions ..................................................... 600  
8.3 How Provisioning Can Be Used ......................... 600  
8.4 How Provisioning Works ................................. 602  
8.5 Configuring Provisioning ................................. 613  
8.6 Viewing Provisioning Information ..................... 623  
8.7 Requirements and Restrictions ....................... 627  
8.8 Defaults and Backward Compatibility ............... 630  
8.9 Example Scripts .............................................. 630  
8.10 Advice and Caveats ........................................ 645  
8.11 Errors and Logging ....................................... 647
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Security</td>
<td>653</td>
</tr>
<tr>
<td>9.1</td>
<td>Configurable Features</td>
<td>653</td>
</tr>
<tr>
<td>9.2</td>
<td>Setting User Roles</td>
<td>655</td>
</tr>
<tr>
<td>9.3</td>
<td>Using Access Control</td>
<td>659</td>
</tr>
<tr>
<td>9.4</td>
<td>Restricting Execution Host Access</td>
<td>683</td>
</tr>
<tr>
<td>9.5</td>
<td>Logging Security Events</td>
<td>685</td>
</tr>
<tr>
<td>9.6</td>
<td>Setting File Copy Mechanism</td>
<td>688</td>
</tr>
<tr>
<td>9.7</td>
<td>Security Considerations for Copying Files</td>
<td>689</td>
</tr>
<tr>
<td>9.8</td>
<td>Changing the PBS Service Account Password</td>
<td>690</td>
</tr>
<tr>
<td>9.9</td>
<td>Paths and Environment Variables</td>
<td>692</td>
</tr>
<tr>
<td>9.10</td>
<td>Authentication &amp; Authorization</td>
<td>693</td>
</tr>
<tr>
<td>9.11</td>
<td>Enabling Hostbased Authentication on Linux</td>
<td>694</td>
</tr>
<tr>
<td>9.12</td>
<td>Root-owned Jobs</td>
<td>698</td>
</tr>
<tr>
<td>9.13</td>
<td>User Passwords</td>
<td>699</td>
</tr>
<tr>
<td>9.14</td>
<td>File and Directory Permissions</td>
<td>702</td>
</tr>
<tr>
<td>9.15</td>
<td>Windows Caveats</td>
<td>702</td>
</tr>
<tr>
<td>10</td>
<td>Making Your Site More Robust</td>
<td>703</td>
</tr>
<tr>
<td>10.1</td>
<td>Robustness</td>
<td>703</td>
</tr>
<tr>
<td>10.2</td>
<td>Failover</td>
<td>704</td>
</tr>
<tr>
<td>10.3</td>
<td>Checkpoint and Restart</td>
<td>735</td>
</tr>
<tr>
<td>10.4</td>
<td>Preventing Communication and Timing Problems</td>
<td>761</td>
</tr>
<tr>
<td>10.5</td>
<td>Reservation Fault Tolerance</td>
<td>774</td>
</tr>
<tr>
<td>11</td>
<td>Integrations</td>
<td>779</td>
</tr>
<tr>
<td>11.1</td>
<td>Support for MPI</td>
<td>779</td>
</tr>
<tr>
<td>11.2</td>
<td>Support for AIX</td>
<td>792</td>
</tr>
<tr>
<td>11.3</td>
<td>User Space Mode on InfiniBand Switches</td>
<td>792</td>
</tr>
<tr>
<td>11.4</td>
<td>Notes on SGI ICE</td>
<td>798</td>
</tr>
<tr>
<td>11.5</td>
<td>Support for Cray Systems</td>
<td>798</td>
</tr>
<tr>
<td>11.6</td>
<td>Configuring MOM for Machines with cpusets</td>
<td>827</td>
</tr>
<tr>
<td>11.7</td>
<td>Configuring MOM on a Cpusetted Machine</td>
<td>828</td>
</tr>
<tr>
<td>11.8</td>
<td>MOM Globus Configuration</td>
<td>836</td>
</tr>
</tbody>
</table>
# Table of Contents

11.9 Using Comprehensive System Accounting .................................................. 836
11.10 Systems with Hyper-Threaded Technology .............................................. 839

12 Managing Jobs ......................................................................................... 841

12.1 Routing Jobs ............................................................................................ 841
12.2 Allocating Resources to Jobs ...................................................................... 842
12.3 Job Prologue and Epilogue .......................................................................... 843
12.4 UNIX Shell Invocation .............................................................................. 850
12.5 Sequence of Events for Start of Job .......................................................... 851
12.6 Sequence of Events for End of Job ........................................................... 852
12.7 Job Termination ....................................................................................... 852
12.8 Job Exit Codes .......................................................................................... 857
12.9 Rerunning or Requeueing a Job ................................................................. 859
12.10 Job IDs ..................................................................................................... 860
12.11 Where to Find Job Information ............................................................... 861
12.12 The Job’s Node File ................................................................................ 862
12.13 Job Directories ........................................................................................ 864
12.14 The Job’s Lifecycle .................................................................................. 869
12.15 Managing Job History ............................................................................ 874
12.16 Environment Variables ........................................................................... 879

13 Administration ............................................................................................ 881

13.1 The PBS Configuration File ....................................................................... 881
13.2 Environment Variables ............................................................................. 885
13.3 The Accounting Log .................................................................................. 886
13.4 Event Logging ............................................................................................ 890
13.5 Using the UNIX syslog Facility .................................................................. 900
13.6 Managing Machines .................................................................................. 901
13.7 Managing the Data Service ....................................................................... 902

14 Problem Solving .......................................................................................... 907

14.1 Server Host Bogs Down After Startup ..................................................... 907
14.2 Finding PBS Version Information ............................................................. 909
14.3 Troubleshooting and Hooks ...................................................................... 909
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.4</td>
<td>Directory Permission Problems</td>
<td>909</td>
</tr>
<tr>
<td>14.5</td>
<td>Common Errors</td>
<td>910</td>
</tr>
<tr>
<td>14.6</td>
<td>Errors on Windows</td>
<td>917</td>
</tr>
<tr>
<td>14.7</td>
<td>Troubleshooting PBS Licenses</td>
<td>924</td>
</tr>
<tr>
<td>14.8</td>
<td>Security-related Problems</td>
<td>926</td>
</tr>
<tr>
<td>14.9</td>
<td>Getting Help</td>
<td>931</td>
</tr>
<tr>
<td>Appendix A</td>
<td>License Agreement</td>
<td>933</td>
</tr>
<tr>
<td>Index</td>
<td></td>
<td>947</td>
</tr>
</tbody>
</table>
About PBS Documentation

Where to Keep the Documentation

To make cross-references work, put all of the PBS guides in the same directory.

What is PBS Professional?

PBS is a workload management system that provides a unified batch queuing and job management interface to a set of computing resources.

The PBS Professional Documentation

The documentation for PBS Professional includes the following:
PBS Professional Administrator’s Guide
Provides the PBS administrator with the information required to configure and manage PBS Professional (PBS).

PBS Professional Quick Start Guide:
Provides a quick overview of PBS Professional installation and license file generation.

PBS Professional Installation & Upgrade Guide:
Contains information on installing and upgrading PBS Professional.

PBS Professional User’s Guide:
Covers user commands and how to submit, monitor, track, delete, and manipulate jobs.

PBS Professional Programmer’s Guide:
Discusses the PBS application programming interface (API).

PBS Professional Reference Guide:
Contains PBS reference material.

PBS Manual Pages:
Describe PBS commands, resources, attributes, APIs

Ordering Software and Publications
To order additional copies of this manual and other PBS publications, or to purchase additional software licenses, contact your Altair sales representative. Contact information is included on the copyright page of this book.

Document Conventions
PBS documentation uses the following typographic conventions:

abbreviation
    The shortest acceptable abbreviation of a command or subcommand is underlined.

command
    Commands such as qmgr and scp
input

Command-line instructions

manpage(x)

File and path names. Manual page references include the section number in parentheses appended to the manual page name.

formats

Formats

Attributes

Attributes, parameters, objects, variable names, resources, types

Values

Keywords, instances, states, values, labels

Definitions

Terms being defined

Output

Output or example code

File contents
Chapter 1

New Features

This chapter briefly lists new features by release, with the most recent listed first. This chapter also lists deprecated elements, such as options, keywords, etc.

The *Release Notes* included with this release of PBS Professional list all new features in this version of PBS Professional, and any warnings or caveats. Be sure to review the Release Notes, as they may contain information that was not available when this book was written.

The PBS Professional manual pages that were reproduced in this guide are available in the PBS Professional Reference Guide or as UNIX man pages. They have been removed from this book to save space.
1.1 New Features in PBS 11.1

1.1.1 Support for Interlagos on Cray

You can allow users to request vnodes that have Interlagos hardware. See section 11.5.7.14 “Allowing Users to Request Interlagos Hardware” on page 818.

1.2 Changes in Previous Releases

1.2.1 Improved Cray Integration (11.0)

PBS is more tightly integrated with Cray systems. You can use the PBS select and place language when submitting Cray jobs. See section 11.5 “Support for Cray Systems” on page 798.

1.2.2 Vnode Access for Hooks (11.0)

Hooks have access to vnode attributes and resources. See Chapter 6, "Hooks", on page 461.

1.2.3 Enhanced Job Placement (11.0)

PBS allows job submitters to scatter chunks by vnode in addition to scattering by host. PBS also allows job submitters to reserve entire hosts via a job’s placement request. See section 3.6, "Placing Jobs on Vnodes", on page 50 of the PBS Professional User’s Guide.
1.2.4 Choice in PBS service account Name (11.0)

Under Windows, the PBS service account used to run PBS daemons can have any name. See section 2.5.2.3 “The PBS Service Account” on page 18 in the PBS Professional Installation & Upgrade Guide and section 2.5.3.3 “The PBS service account for Standalone Environments” on page 21 in the PBS Professional Installation & Upgrade Guide.

1.2.5 Change of Licensing Method (11.0)

As of 11.0, PBS is licensed using a new Altair license server. See section “Licensing” on page 101 in the PBS Professional Installation & Upgrade Guide.

1.2.6 Change in Data Management (11.0)

PBS uses a new data service. See section 13.7 “Managing the Data Service” on page 902.

1.2.7 Choice in Job Requeue Timeout (11.0)

You can choose how long the job requeue process should be allowed to run. See section 10.4.3 “Setting Job Requeue Timeout” on page 767.

1.2.8 Backfilling Around Top N Jobs (10.4)

PBS can backfill around the most deserving jobs. You can configure the number of jobs PBS backfills around. See section 4.8.3 “Using Backfilling” on page 145.
1.2.9 Estimating Job Start Times (10.4)

PBS can estimate when jobs will run, and which vnodes each job will use. See section 4.8.15 “Estimating Job Start Time” on page 181.

1.2.10 Unified Job Submission (10.4)

PBS allows users to submit jobs using the same scripts, whether the job is submitted on a Windows or UNIX/Linux system. See section 3.3.3.1, "Python Job Scripts", on page 29 of the PBS Professional User’s Guide.

1.2.11 Provisioning (10.2)

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application. See Chapter 8, "Provisioning", on page 599.

1.2.12 New Hook Type (10.2)

PBS has a new hook type which can be triggered when a job is to be run. See “Hooks” on page 461.

1.2.13 New Scheduler Attribute (10.2)

PBS allows the administrator to set the scheduler’s cycle time using the new sched_cycle_length scheduler attribute. See the pbs_sched_attributes(7B) manual page.
1.2.14 Walltime as Checkpoint Interval Measure (10.2)

PBS allows a job to be checkpointed according to its walltime usage. See the pbs_job_attributes(7B) manual page.

1.2.15 Employing User Space Mode on IBM InfiniBand Switches (10.2)

PBS allows users submitting POE jobs to use InfiniBand switches in User Space mode. See section 11.3 “User Space Mode on InfiniBand Switches” on page 792.

1.2.16 Scheduling Jobs Using HPCBP (10.1)

PBS Professional can schedule and manage jobs on one or more Microsoft Windows HPC Servers using the Grid Forum OGSA HPC Basic Profile web services standard. See Chapter 7, "Metascheduling Using HPC Basic Profile", on page 579.

1.2.17 Managing Resource Usage (10.1)

You can set separate limits for resource usage by individual users, individual groups, generic users, generic groups, and the total used by all users. You can limit the amount of resources used, and the number of running and queued jobs. These limits can be defined separately for each queue and for the server. See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410. These new limits are incompatible with the limit attributes existing before Version 10.1.
1.2.18 Managing Job History (10.1)

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used. PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server. See section 12.15 “Managing Job History” on page 874.

1.2.19 Reservation Fault Tolerance (10.1)

PBS attempts to reconfirm reservations for which associated vnodes have become unavailable. See section 10.5 “Reservation Fault Tolerance” on page 774.

1.2.20 Checkpoint Support via Epilogue (10.1)

Checkpointed jobs can be requeued if the epilogue exits with a special value. See section 10.3.7.3 “Requeueing via Epilogue” on page 757.

1.2.21 Hooks (10.0)

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code. See section “Hooks” on page 461.

1.2.22 Versioned Installation (10.0)

PBS is now automatically installed in versioned directories. For most platforms, different versions of PBS can coexist, and upgrading is simplified. See Chapter 3, "Installation", on page 31 and Chapter 5, "Upgrading PBS Professional", on page 127 in the PBS Professional Installation and Upgrade Guide.
1.2.23 Resource Permissions for Custom Resources (9.2)

You can set permissions on custom resources so that they are either invisible to users or cannot be requested by users. This also means that users cannot modify a resource request for those resources via `qalter`. See section 5.14.2.5 “Resource Permission Flags” on page 362.

1.2.24 Extension to Job Sorting Formula (9.2)

The job sorting formula has been extended to include parentheses, exponentiation, division, and unary plus and minus. See section 4.8.3 “Using Backfilling” on page 145.

1.2.25 Eligible Wait Time for Jobs (9.2)

A job that is waiting to run can be accruing “eligible time”. Jobs can accrue eligible time when they are blocked due to a lack of resources. This eligible time can be used in the job sorting formula. Jobs have two new attributes, `eligible_time` and `accrue_type`, which indicates what kind of wait time the job is accruing. See section 4.8.13 “Eligible Wait Time for Jobs” on page 173.

1.2.26 Job Staging and Execution Directories (9.2)

PBS now provides per-job staging and execution directories. Jobs have new attributes `sandbox` and `jobdir`, the MOM has a new option `$jobdir_root`, and there is a new environment variable called `PBS_JOBDIR`. If the job’s `sandbox` attribute is set to `PRIVATE`, PBS creates a job-specific staging and execution directory. If the job’s `sandbox` attribute is unset or is set to `HOME`, PBS uses the user’s home directory for staging and execution, which is how previous versions of PBS behaved. If MOM’s `$jobdir_root` is set to a specific directory, that is where PBS will create job-specific staging and execution directories. If MOM’s
$\texttt{jobdir\_root}$ is unset, PBS will create the job-specific staging and execution directory under the user’s home directory. See section 12.13.1 “The Job’s Staging and Execution Directories” on page 864.

### 1.2.27 Standing Reservations (9.2)

PBS now provides both advance and standing reservation of resources. A standing reservation is a reservation of resources for specific recurring periods of time. See section 4.8.37 “Advance and Standing Reservations” on page 283.

### 1.2.28 New Server Attribute for Job Sorting Formula (9.1)

The new server attribute “job\_sort\_formula” is used for sorting jobs according to a site-defined formula. See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

### 1.2.29 Change to sched\_config (9.1)

The default for $\texttt{job\_sort\_key}$ of “cput” is commented out in the default $\texttt{sched\_config}$ file. It is left in as a usage example.

### 1.2.30 Change to Licensing (9.0)

PBS now depends on an Altair license server that will hand out licenses to be assigned to PBS jobs. See section “Licensing” on page 101 in the PBS Professional Installation & Upgrade Guide. PBS Professional versions 8.0 and below will continue to be licensed using the proprietary licensing scheme.
1.2.31 Installing With Altair Licensing (9.0)

You must install and configure the Altair license server before installing and configuring PBS. See section 3.1 “Requirements for Installing PBS Professional” on page 32 in the PBS Professional Installation & Upgrade Guide.

1.2.32 Unset Host-level Resources Have Zero Value (9.0)

An unset numerical resource at the host level behaves as if its value is zero, but at the server or queue level it behaves as if it were infinite. An unset string or string array resource cannot be matched by a job’s resource request. An unset boolean resource behaves as if it is set to “False”. See section 4.8.28.6 “Matching Unset Resources” on page 224.

1.2.33 Better Management of Resources Allocated to Jobs (9.0)

The resources allocated to a job from vnodes will not be released until certain allocated resources have been freed by all MOMs running the job. The end of job accounting record will not be written until all of the resources have been freed. The “end” entry in the job end (‘E’) record will include the time to stage out files, delete files, and free the resources. This will not change the recorded “walltime” for the job.

1.2.34 Support for Large Page Mode on AIX (9.0)

PBS Professional supports Large Page Mode on AIX. No additional steps are required from the PBS administrator.
1.3 Deprecations

The -a alarm option to pbs_sched is deprecated, and is replaced with the sched_cycle_length scheduler attribute.

The sort_priority option to job_sort_key is deprecated and is replaced with the job_priority option.

The -lnodes=nodespec form is replaced by the -l select= and -l place= statements.

The nodes resource is no longer used.

The -l resource=rescspec form is replaced by the -l select= statement.

The time-shared node type is no longer used, and the :ts suffix is obsolete.

The cluster node type is no longer used.

The resource arch is only used inside of a select statement.

The resource host is only used inside of a select statement.

The nodect resource is obsolete. The ncpus resource should be used instead. Sites which currently have default values or limits based on nodect should change them to be based on ncpus.

The neednodes resource is obsolete.

The ssinodes resource is obsolete.

Properties are replaced by boolean resources.

The -a option to the qselect command is deprecated.

The -Wdelay=nnnn option to the qdel command is deprecated.

The -c, -o and -d options to the pbsnodes command are deprecated.

The -c and -d options to pbsnodes are removed.

The memreserved MOM configuration option is deprecated.

The pbs_tclapi pbsrescquery command is deprecated.

The pbs_rescquery command is deprecated.
The **sync_time** scheduler configuration option is deprecated.

The Cray **mpp** syntax is deprecated with PBS version 11. Requesting the **mpp** resources in a `qsub`, `pbs_rsub`, or `qalter` command is deprecated.

- The following resources are deprecated:
  - `mppwidth`
  - `mppdepth`
  - `mppnppn`
  - `mppmem`
  - `mpparch`
  - `mpphost`
  - `mpplabels`
  - `mppnodes`

- PBS does not support server or queue level **mpp** defaults. The following are deprecated:
  - `resources_default.mppwidth`
  - `resources_default.mppdepth`
  - `resources_default.mppnppn`
  - `resources_default.mppmem`
  - `resources_default.mpparch`
  - `resources_default.mpphost`
  - `resources_default.mpplabels`
  - `resources_default.mppnodes`

- PBS does not support **mpp** minima or maxima for server and queues.
The following are deprecated:

resources_min.mppwidth
resources_min.mppdepth
resources_min.mppnppn
resources_min.mppmem
resources_min.mpparch
resources_min.mpphost
resources_min.mpplabels
resources_min.mppnodes
resources_max.mppwidth
resources_max.mppdepth
resources_max.mppnppn
resources_max.mppmem
resources_max.mpparch
resources_max.mpphost
resources_max.mpplabels
resources_max.mppnodes

The \texttt{pbs\_license\_file\_location} server attribute is deprecated and replaced by \texttt{pbs\_license\_info}.

The \texttt{configm()} resource monitor API call is deprecated.
1.4 Backward Compatibility

1.4.1 New and Old Resource Usage Limits Incompatible

The new resource usage limits are incompatible with the old resource usage limits. See section 5.15.1.14 “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1” on page 432, section 5.15.1.12.6 “Do Not Mix Old And New Limits” on page 430, and section 5.15.1.13.1 “Error When Setting Limit Attributes” on page 431.

1.4.2 Job Dependencies Affected By Job History

Enabling job history changes the behavior of dependent jobs. If a job j1 depends on a finished job j2 for which PBS is maintaining history than j1 will go into the held state. If job j1 depends on a finished job j3 that has been purged from the historical records than j1 will be rejected just as in previous versions of PBS where the job was no longer in the system.

1.4.3 PBS path information no longer saved in AUTOEXEC.BAT

Any value for PATH saved in AUTOEXEC.BAT may be lost after installation of PBS. If there is any path information that needs to be saved, AUTOEXEC.BAT must be edited by hand after the installation of PBS. PBS path information is no longer saved in AUTOEXEC.BAT.
1.4.4  OS-level Checkpointing Not Supported

PBS does not directly support OS-level checkpointing. PBS supports checkpointing using site-supplied methods. See section 10.3 “Checkpoint and Restart” on page 735.
Chapter 2

Configuring the Server and Queues

This chapter describes how to configure the server and any queues.

2.1 The Server

2.1.1 Configuring the Server

You configure the server by setting server attributes via the `qmgr` command:

```
  Qmgr: set server <attribute> = <value>
```

For a description of the server attributes, see “Server Attributes” on page 365 of the PBS Professional Reference Guide.
For a description of the `qmgr` command, see “qmgr” on page 178 of the PBS Professional Reference Guide.

### 2.1.2 Default Server Configuration

The default configuration from the binary installation sets the default server settings. An example server configuration is shown below:

```
qmgr
Qmgr: print server
#
# Create queues and set their attributes.
# Create and define queue workq
#
create queue workq
set queue workq queue_type = Execution
set queue workq enabled = True
set queue workq started = True
#
# Set server attributes.
#
set server scheduling = True
set server default_queue = workq
set server log_events = 511
set server mail_from = adm
set server query_other_jobs = True
set server resources_default.ncpus = 1
set server scheduler_iteration = 600
set server resv_enable = True
set server node_fail_requeue = 310
set server max_array_size = 10000
set server default_chunk.ncpus=1
```
2.1.3 The Server’s Nodes File

The server creates a file of the nodes managed by PBS. This nodes file is written only by the Server. On startup each MOM sends a time-stamped list of her known vnodes to the Server. The Server updates its information based on that message. If the time stamp on the vnode list is newer than what the Server recorded before in the nodes file, the Server will create any vnodes which were not already defined. If the time stamp in the MOM’s message is not newer, then the Server will not create any missing vnodes and will log an error for any vnodes reported by MOM but not already known.

Whenever new vnodes are created, the Server sends a message to each MOM with the list of MOMs and each vnode managed by the MOMs. The Server will only delete vnodes when they are explicitly deleted via qmgr.

This is different from the nodes file created for each job. See section 12.12 “The Job’s Node File” on page 862.

2.1.4 Server Configuration Attributes


2.1.5 Recording Server Configuration

If you wish to record the configuration of a PBS Server for re-use later, you may use the print subcommand of qmgr(8B). For example,

```
qmgr -c "print server" > /tmp/server.out
qmgr -c "print node @default" > /tmp/nodes.out
```

will record in the file /tmp/server.out the qmgr subcommands required to recreate the current configuration including the queues. The second file generated above will contain the vnodes and all the vnode properties. The commands could be read back into qmgr via standard input:

```
qmgr < /tmp/server.out
qmgr < /tmp/nodes.out
```
2.1.6 Server Support for Globus

Two hosts may be defined with the same hosts name. One may be a Globus host (MOM), and the other a non-Globus host.

Globus support is not available on Windows.

2.1.7 Configuring the Server for Licensing

The PBS server must be configured for licensing. You must set the location where PBS will look for the license server, by setting the server attribute \texttt{pbs-license-info}. The other server licensing attributes have defaults, but you may wish to set them as well. See section 4.4 “Configuring PBS for Licensing” on page 105 in the PBS Professional Installation & Upgrade Guide.

You may also wish to have redundant license servers. See the Altair License Management System Installation and Operations Guide, available at \url{www.pbsworks.com}.

2.1.8 Configuring Mail

You can configure the account that is used as the address to both send and receive administrative mail. These are the same account. For example, when failover occurs, an email is sent to and from the account defined in the server’s \texttt{mail-from} attribute, saying that failover has occurred.

Use the \texttt{qmgr} command to set the \texttt{mail-from} server attribute to an address that is monitored regularly:

\begin{verbatim}
 Qmgr: s server mail_from=<address>
\end{verbatim}

You cannot configure which mail server PBS uses. PBS uses the default mail server. On UNIX/Linux, this is \texttt{/usr/lib/sendmail}.

On Windows, PBS uses sendmail on the host specified in the server’s \texttt{mail-from} attribute. For example, if you set \texttt{mail-from} to \texttt{admin_acct@host1.example.com}, PBS uses \texttt{sendmail} on host1.
2.1.8.1 Mail Caveats

If you do not set the `mail_from` attribute on Windows, PBS will not be able to send mail.

2.2 Queues

When a job is submitted to PBS and accepted, it is placed in a queue. Despite the fact that the name implies first-in, first-out ordering of jobs, this is not the case. Job submission order does not determine job execution order. See Chapter 4, "Scheduling", on page 71.

You can create different queues for different purposes: queues for certain kinds of jobs, queues for specific groups, queues for specific vnodes, etc. You can tell PBS how to automatically route jobs into each queue. PBS has a default execution queue named `workq`, where jobs are placed when no queue is requested. You can specify which queue should be the default. See section 2.2.14 “Specifying Default Queue” on page 38.

2.2.1 Kinds of Queues

2.2.1.1 Execution and Routing Queues

There are two main types of PBS queues: `routing` and `execution`.

- A routing queue is used only to move jobs to other queues. These destination queues can be routing or execution queues, and can be located at different PBS servers.

- An execution queue is used as the home for a waiting or running job. A job must reside in an execution queue to be eligible to run. The job remains in the execution queue during the time it is running.

For more information on creating and using routing queues, see section 2.2.6 “Routing Queues” on page 26.

For information on configuring routing queues and failover, see section 10.2.6.1 “Configuring Failover to Work with Routing Queues” on page 730.
2.2.1.2 Available Kinds of Queues

PBS supplies the following kinds of execution and routing queues:

**Table 2-1: Kinds of Queues**

<table>
<thead>
<tr>
<th>Kind of Queue</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing queues</td>
<td>Used for moving jobs to another queue</td>
<td>See section 2.2.6 “Routing Queues” on page 26</td>
</tr>
</tbody>
</table>
### Table 2-1: Kinds of Queues

<table>
<thead>
<tr>
<th>Kind of Queue</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution queues</td>
<td>Reservation queues Created for reservation.</td>
<td>See section 2.2.5.2.4 “Reservation Queues” on page 25</td>
</tr>
<tr>
<td>Dedicated time queues</td>
<td>Holds jobs that run only during dedicated time.</td>
<td>See section 2.2.5.2.1 “Dedicated Time Queues” on page 25</td>
</tr>
<tr>
<td>Primetime queues</td>
<td>Holds jobs that run only during primetime.</td>
<td>See section 2.2.5.2.2 “Primetime and Non-Primetime Queues” on page 25</td>
</tr>
<tr>
<td>Non-primetime queues</td>
<td>Holds jobs that run only during non-primetime.</td>
<td>See section 2.2.5.2.2 “Primetime and Non-Primetime Queues” on page 25</td>
</tr>
<tr>
<td>Anytime queues</td>
<td>Queue with no dedicated time or primetime restric-</td>
<td>See section 2.2.5.2.3 “Anytime Queues” on page 25</td>
</tr>
<tr>
<td></td>
<td>tions</td>
<td></td>
</tr>
<tr>
<td>Express queues</td>
<td>High-priority queue; priority is set to the level</td>
<td>See section 2.2.5.3.1 “Express Queues” on page 26</td>
</tr>
<tr>
<td></td>
<td>signifying that it is an express queue</td>
<td></td>
</tr>
<tr>
<td>Anti-express queue</td>
<td>Low-priority queue designed for work that should run only when no other jobs need the resources</td>
<td>See section 4.8.1 “Anti-Express Queues” on page 140</td>
</tr>
</tbody>
</table>
2.2.2 Basic Queue Use

The simplest form of PBS uses just one queue. The queue is an execution queue named `workq`. This queue is always created, enabled, and started for you during installation. After a basic installation, this queue is ready to hold jobs submitted by users.

2.2.3 Creating Queues

To create a queue, use the `qmgr` command to create it and set its `queue_type` attribute:

```
Qmgr: create queue <queue name>
Qmgr: set queue <queue_name> queue_type = <execution or route>
```

For example, to create an execution queue named `exec_queue`, set its type, start it, and enable it:

```
Qmgr: create queue exec_queue
Qmgr: set queue exec_queue queue_type = execution
Qmgr: set queue exec_queue enabled = True
Qmgr: set queue exec_queue started = True
```

Now we will create a routing queue, which will send jobs to our execution queue:

```
Qmgr: create queue routing_queue
Qmgr: set queue routing_queue queue_type = route
Qmgr: set queue routing_queue route_destinations = exec_queue
```
2.2.4 Enabling, Disabling, Starting, and Stopping Queues

When you enable a queue, you allow it to accept jobs, meaning that jobs can be enqueued in the queue. When you disable a queue, you disallow it from accepting jobs. Queues are disabled by default. You enable a queue by setting its enabled attribute to True:

Qmgr: set queue <queue name> enabled = True

When you start a queue, you allow the jobs in the queue to be executed. Jobs are selected to be run according to the scheduling policy. When you stop a queue, you disallow jobs in that queue from running, regardless of scheduling policy. Queues are stopped by default. You start a queue by setting its started attribute to True:

Qmgr: set queue <queue name> started = True

2.2.5 Execution Queues

Execution queues are used to run jobs; jobs must be in an execution queue in order to run. PBS does not route from execution queues.

2.2.5.1 Where Execution Queues Get Their Jobs

By default, PBS allows jobs to be moved into execution queues via the qmove command, by hooks, from routing queues, and by being submitted to execution queues. You can specify that an execution queue should accept only those jobs that are routed from a routing queue by PBS, by setting the queue’s from_route_only attribute to True:

Qmgr: set queue <queue name> from_route_only = True
2.2.5.2 Execution Queues for Specific Time Periods

PBS provides a mechanism that allows you to specify that the jobs in an execution queue can run only during specific time periods. PBS provides a different kind of execution queue for each kind of time period. The time periods you can specify are the following:

Advance or Standing Reservations
You can create an advance or standing reservation. An advance reservation is a reservation for specified resources for a specified time period with a defined beginning and end. A standing reservation is a series of recurring advance reservations.

Dedicated time
Dedicated time is a period of time with a defined beginning and end. You can define multiple dedicated times.

Primetime
Primetime is a recurring time period with a defined beginning and end. You can define primetime to be different for each day of the week.

Non-primetime
Non-primetime is a recurring time period with a defined beginning and end. Non-primetime begins when primetime ends, and vice versa.

Holidays
Holidays are dates defined in the PBS_HOME/sched_priv/holidays file. PBS has a default set of holidays, and you can define your own holidays. Holiday time is treated like non-primetime, meaning jobs in non-primetime queues run during holiday time.

Anytime queue
The term “anytime queue” means a queue that is not a primetime or a non-primetime queue.
2.2.5.2.1 Dedicated Time Queues

The jobs in a dedicated time execution queue can run only during dedicated time. Dedicated time is defined in PBS_HOME/sched_priv/dedicated_time. See section 4.8.10 “Dedicated Time” on page 171.

To specify that a queue is a dedicated time queue, you prefix the queue name with the dedicated time keyword. This keyword defaults to “ded”, but can be defined in the dedicated_prefix scheduler parameter in PBS_HOME/sched_priv/sched_config. See “dedicated_prefix” on page 330 of the PBS Professional Reference Guide.

2.2.5.2.2 Primetime and Non-Primetime Queues

The jobs in a primetime queue run only during primetime, and the jobs in a non-primetime queue run only during non-primetime. Primetime and non-primetime are defined in PBS_HOME/sched_priv/holidays. See section 4.8.34 “Using Primetime and Holidays” on page 274.

To specify that a queue is a primetime or non-primetime queue, you prefix the queue name with the primetime or non-primetime keyword. For primetime, this keyword defaults to “p_”, and for non-primetime, the keyword defaults to “np_”, but these can be defined in the primetime_prefix and nonprimetime_prefix scheduler parameters in PBS_HOME/sched_priv/sched_config. See “Scheduler Parameters” on page 327 of the PBS Professional Reference Guide.

2.2.5.2.3 Anytime Queues

An anytime queue is a queue whose jobs can run at any time. An anytime queue is simply a queue that is not a dedicated time, primetime, or non-primetime queue.

2.2.5.2.4 Reservation Queues

When the pbs_rsub command is used to create a reservation or to convert a job into a reservation job, PBS creates a reservation queue. Jobs in the queue run only during the reservation. See section 4.8.37 “Advance and Standing Reservations” on page 283.
2.2.5.3 Prioritizing Execution Queues

You can set the priority of each execution queue as compared to the other queues in this complex by specifying a value for the `priority` queue attribute:

```
Qmgr: set queue <queue name> priority = <value>
```

A higher value for priority means the queue has greater priority. See “Queue Attributes” on page 407 of the PBS Professional Reference Guide.

For how queue priority is used in scheduling, see section 4.8.36 “Queue Priority” on page 282.

2.2.5.3.1 Express Queues

A queue is an `express queue` if its priority is greater than or equal to the value that defines an express queue. This value is set in the `preempt_queue_prio` parameter in PBS_HOME/sched_priv/sched_config. The default value for `preempt_queue_prio` is 150.

You do not need to set `by_queue` or `sort_queues` to `True` in order to use express queues.

For how express queues can be used, see section 4.8.17 “Express Queues” on page 190.

2.2.6 Routing Queues

A routing queue is used only to route jobs; jobs cannot run from a routing queue. A routing queue has the following properties:

- Can route to multiple destinations
- Tries destinations in round-robin fashion, in the order listed
- Can route to execution queues
- Can route to other routing queues
- Can route to queues in other complexes (at other servers)
Destinations can be specified in the following ways:

```
route_destinations = Q1
route_destinations = Q1@Server1
route_destinations = "Q1, Q2@Server1, Q3@Server2"
route_destinations += Q1
route_destinations += "Q4, Q5@Server3"
```

### 2.2.6.1 How Routing Works

Whenever a job enters a routing queue, PBS immediately attempts to route the job to a destination queue. The result is one of the following:

- The job is routed to one of the destination queues.
- The attempt to route is permanently rejected by each destination, and the job is deleted.
- Every destination rejects the job, but at least one rejection is temporary. In this case, the destination is tried again later.

If there are multiple routing queues containing jobs to be routed, the routing queues are processed in the order in which they are displayed in the output of a `qstat -Q` command.

When PBS routes a job, it tries each destination in the order listed. The job’s destination is the first queue that accepts it.

Queue priority does not play a role in routing jobs.

### 2.2.6.2 Requirements for Routing Queues

- A routing queue’s destination queues must be created before being specified in the routing queue’s `route_destinations` attribute.
- A routing queue’s `route_destinations` attribute must be specified before enabling and starting the routing queue.

### 2.2.6.3 Caveats and Advice for Routing Queues

- Routing loops should be avoided. If a job makes more than 20 routing hops, it is discarded, and mail may be sent. Avoid setting a routing
queue’s destination to be the routing queue itself.

- When routing to a complex that is using failover, it's a good idea to include the names of both primary and secondary servers in a routing destination:

  ```
  route_destinations = "destQ@primary_server, destQ@secondary_server"
  ```

- When routing a job between complexes, the job’s owner must be able to submit a job to the destination complex.

- When routing to a destination in another complex, the source and destination complexes should use the same version of PBS. If not, you may need a submission hook to modify incoming jobs.

- It is recommended to list the destination queues in order of the most restrictive first, because the first queue which meets the job’s requirements and is enabled will be its destination.

### 2.2.6.4 Using Resources to Route Jobs Between Queues

You can use resources to direct jobs to the desired queues. The server will automatically route jobs that are in routing queues, based on job resource requests. The destination queue can be at the local server or at another server. If you have more than one PBS complex, you may want to route jobs between the complexes, depending on the resources available at each complex.

You can set up queues for specific kinds of jobs, for example jobs requesting very little memory, a lot of memory, or a particular application. You can then route jobs to the appropriate queues.

A routing queue tests destination queues in the order listed in the queue’s `route_destinations` attribute. The job is placed in the first queue that meets the job’s request and is enabled.

#### 2.2.6.4.1 How Queue Limits Are Applied

You can set a minimum and a maximum for each resource at each queue using the `resources_min.<resource>` and `resources_max.<resource>` queue attributes. Any time a job is considered for entry into a queue, the job's resource request is tested against `resources_min.<resource>` and `resources_max.<resource>` for that queue. The job’s resource request
must be greater than or equal to the value specified in
resources_min.<resource>, and less than or equal to the value specified
in resources_max.<resource>.

The job is tested only against existing resources_min.<resource> and
resources_max.<resource> for the queue.

Only those resources that are specified in the job’s resource request are
tested, so if a job does not request a particular resource, and did not inherit
a default for that resource, the minimum and maximum tests for that
resource are not applied to the job.

If you want jobs requesting only a specific value for a resource to be
allowed into a queue, set the queue’s resources_min.<resource> and
resources_max.<resource> to the same value. This resource can be
numeric, string, string array, or Boolean.

If you limit queue access using a string array, a job must request one of the
values in the string array to be allowed into the queue. For example, if you
set resources_min.strarr and resources_max.strarr to “blue,red,black”,
jobs can request –l strarr=blue, -l strarr=red, or –l
strarr=black to be allowed into the queue.

2.2.6.4.2 Resources Used for Routing and Admittance

When jobs are routed using a chunk-level resource, routing is based on the
sum of that resource across all chunks.

You can route jobs using the following kinds of resources:

- Any server-level or queue-level (job-wide) built-in or custom resource,
  whether it is numeric, string, or Boolean, for example ncpus and software
- The following built-in chunk-level resources:
  ncpus
  netwins
  mem
  vmem
- Custom vnode-level (chunk-level) resources that are global and have
the `n`, `q`, or `f` flags set

- Any resource in the job’s `Resource_List` attribute. For string or string array resources, see section 2.2.6.4.4 “Using String, String Array, and Boolean Values for Routing” on page 30.

2.2.6.4.3 Routing and Admittance Limitations

You cannot route jobs using time-based resources such as `walltime`. Setting `resources_available.walltime` at the server or queue has no effect.

2.2.6.4.4 Using String, String Array, and Boolean Values for Routing

The `resources_min` and `resources_max` attributes work as expected with numeric values. In addition, they can be used with string and Boolean values to force an exact match; this is done by setting both to the same value. For example, to limit jobs entering `queue big` to those that specify `arch=unicos8`, or that do not specify a value for `arch`:

```
Qmgr: set q App1Queue resources_max.software=App1
Qmgr: set q App1Queue resources_min.software=App1
```

2.2.6.4.5 Examples of Routing Jobs

You can force all jobs into a routing queue, or you can allow users to request some queues but not others. If you set up the default queue be a routing queue, and make all execution queues accept jobs only from routing queues, all jobs are initially forced into a routing queue.

Alternatively, you can set up one routing queue and a couple of execution queues which accept jobs only from routing queues, but add other queues which can be requested. Or you could allow jobs to request the execution queues, by making the execution queues also accept jobs that aren’t from routing queues.
Example 1: Jobs can request one execution queue named \textit{WorkQ}. All jobs that do not request a specific queue are routed according to their wall-time:

- Create a routing queue \textit{RouteQ} and make it the default queue:
  
  \begin{verbatim}
  Qmgr: create queue RouteQ queue_type = route  
  Qmgr: set server default_queue = RouteQ
  \end{verbatim}

- Create two execution queues, \textit{LongQ} and \textit{ShortQ}. One is for long-running jobs, and one is for short-running jobs:
  
  \begin{verbatim}
  Qmgr: create queue LongQ queue_type = execution  
  Qmgr: create queue ShortQ queue_type = execution
  \end{verbatim}

- Set \texttt{resources\_min.walltime} and \texttt{resources\_max.walltime} on these queues:
  
  \begin{verbatim}
  Qmgr: set queue LongQ resources_min.walltime = 5:00:00  
  Qmgr: set queue ShortQ resources_max.walltime = 4:59:00
  \end{verbatim}

- For \textit{LongQ} and \textit{ShortQ}, disallow jobs that are not from a route queue:
  
  \begin{verbatim}
  Qmgr: set queue LongQ from_route_only = True  
  Qmgr: set queue ShortQ from_route_only = True
  \end{verbatim}

- Set the destinations for \textit{RouteQ} to be \textit{LongQ} and \textit{ShortQ}:
  
  \begin{verbatim}
  Qmgr: set queue RouteQ route_destinations = “ShortQ, LongQ”
  \end{verbatim}

- Create a work queue that can be requested:
  
  \begin{verbatim}
  Qmgr: create queue WorkQ queue_type = execution
  \end{verbatim}

- Enable and start all queues:
  
  \begin{verbatim}
  Qmgr: active queue RouteQ,LongQ,ShortQ,WorkQ  
  Qmgr: set queue enabled = True  
  Qmgr: set queue started = True
  \end{verbatim}

- Set default for \texttt{walltime} at the server so that jobs that don’t request it inherit the default, and land in \textit{ShortQ}:
  
  \begin{verbatim}
  Qmgr: set server resources_default.walltime = 4:00:00
  \end{verbatim}
Example 2: Jobs are not allowed to request any queues. All jobs are routed to one of three queues based on the job's walltime request:

- Create a routing queue `RouteQ` and make it the default queue:
  
  ```
  Qmgr: create queue RouteQ queue_type = route
  Qmgr: set server default_queue = RouteQ
  ```

- Create three execution queues, `LongQ`, `MedQ`, and `ShortQ`. One is for long-running jobs, one is for medium jobs, and one is for short-running jobs:
  
  ```
  Qmgr: create queue LongQ queue_type = execution
  Qmgr: create queue MedQ queue_type = execution
  Qmgr: create queue ShortQ queue_type = execution
  ```

- Set `resources_min.walltime` and `resources_max.walltime` on these queues:
  
  ```
  Qmgr: set queue LongQ resources_min.walltime = 10:00:00
  Qmgr: set queue MedQ resources_max.walltime = 9:59:00
  Qmgr: set queue MedQ resources_min.walltime = 5:00:00
  Qmgr: set queue ShortQ resources_max.walltime = 4:59:00
  ```

- For `LongQ`, `MedQ`, and `ShortQ`, disallow jobs that are not from a route queue:
  
  ```
  Qmgr: set queue LongQ from_route_only = True
  Qmgr: set queue MedQ from_route_only = True
  Qmgr: set queue ShortQ from_route_only = True
  ```

- Set the destinations for `RouteQ` to be `LongQ`, `MedQ` and `ShortQ`:
  
  ```
  Qmgr: set queue RouteQ route_destinations = "ShortQ, MedQ, LongQ"
  ```

- Enable and start all queues:
  
  ```
  Qmgr: active queue RouteQ, LongQ, ShortQ, MedQ
  Qmgr: set queue enabled = True
  Qmgr: set queue started = True
  ```
2.2.6.4.6 Caveats for Queue Resource Limits

If a job is submitted without a request for a particular resource, and no defaults for that resource are set at the server or queue, and either the server or queue has `resources_max.<resource>` set, the job inherits that maximum value. If the queue has `resources_max.<resource>` set, the job inherits the queue value, and if not, the job inherits the server value.

2.2.6.5 Using Access Control to Route Jobs

You can route jobs based on job ownership by setting access control limits at destination queues. A queue’s access control limits specify which users or groups are allowed to have jobs in that queue. Default behavior is to disallow an entity that is not listed, so you need only list allowed entities.

To set the list of allowed users at a queue:

```
Qmgr: set queue <queue name> acl_users =
   "User1@*.example.com, User2@*.example.com"
```

To enable user access control at a queue:

```
Qmgr: set queue <queue name> acl_user_enable = True
```

To set the list of allowed groups at a queue:

```
Qmgr: set queue <queue name> acl_groups =
   "Group1@*.example.com, Group2@*.example.com"
```

To enable group access control at a queue:

```
Qmgr: set queue <queue name> acl_group_enable = True
```

For a complete explanation of access control, see section 9.3 “Using Access Control” on page 659.
2.2.6.6 Allowing Routing of Held or Waiting Jobs

By default, PBS will not route jobs that are held. You can allow a routing queue to route held jobs by setting the queue’s `route_held_jobs` attribute to `True`:

```
Qmgr: set queue <queue name> route_held_jobs = True
```

By default, PBS will not route jobs whose `execution_time` attribute has a value in the future. You can allow a routing queue to route jobs whose start time is in the future by setting the queue’s `route_waiting_jobs` attribute to `True`:

```
Qmgr: set queue <queue name> route_waiting_jobs = True
```

2.2.6.7 Setting Routing Retry Time

The default time between routing retries is 30 seconds. To set the time between routing retries, set the value of the queue’s `route_retry_time` attribute:

```
Qmgr: set queue <queue name> route_retry_time = <value>
```

2.2.6.8 Specifying Job Lifetime in Routing Queue

By default, PBS allows a job to exist in a routing queue for an infinite amount of time. To change this, set the queue’s `route_lifetime` attribute:

```
Qmgr: set queue <queue name> route_lifetime = <value>
```

2.2.7 Queue Requirements

- Each queue must have a unique name. The name must be alphanumeric, and must begin with an alphabetic character
- A Server may have multiple queues of either or both types, but there must be at least one execution queue defined.
2.2.8 Queue Configuration Attributes

Queue configuration attributes fall into three groups:

- Those which apply to both types of queues
- Those which apply only to execution queues
- Those which apply only to routing queues

If an “execution queue only” attribute is set for a routing queue, or vice versa, it is ignored. However, as this situation might indicate the administrator made a mistake, the Server will write a warning message on stderr about the conflict. The same message is written when the queue type is changed and there are attributes that do not apply to the new type.

See “Queue Attributes” on page 407 of the PBS Professional Reference Guide for a table of queue attributes.

2.2.9 Viewing Queue Status

To see the status of a queue, including values for attributes, use the `qstat` command:

```
qstat -Qf <queue name>
```

To see the status of all queues:

```
qstat -Qf
```

The status of the queue is reported in the **State** field. The field shows two letters. One is either *E* (enabled) or *D* (disabled.) The other is *R* (running, same as started) or *S* (stopped.) Attributes with non-default values are displayed. See “qstat” on page 228 of the PBS Professional Reference Guide.

The following queue attributes contain queue status information:
2.2.10 Deleting Queues

Use the `qmgr` command to delete queues.

```
Qmgr: delete queue <queue name>
```

2.2.10.1 Caveats for Deleting Queues

- A queue that has queued or running jobs cannot be deleted.
- A queue that is associated with a vnode via that vnode’s `queue` attribute cannot be deleted. To remove the association, save the output of `pbs-nodes -a` to a file and search for the queue. Unset the `queue` attribute for each associated vnode.

2.2.11 Defining Queue Resources

For each queue, you can define the resources you want to have available at that queue. To set the value for an existing resource, use the `qmgr` command:

```
Qmgr: set queue <queue name> 
resources_available.<resource name> = <value>
```

For example, to set the value of the Boolean resource `RunsMyApp` to `True` at `QueueA`:

```
Qmgr: set queue QueueA 
resources_available.QueueA.RunsMyApp = True
```

For information on how to define a new resource at a queue, see section 5.14 “Custom Resources” on page 352.
For information on defining default resources at a queue, see section 5.9.3.3 “Specifying Job-wide Default Resources at Queue” on page 340 and section 5.9.3.4 “Specifying Chunk Default Resources at Queue” on page 340.

### 2.2.12 Setting Queue Resource Defaults

The jobs that are placed in a queue inherit the queue’s defaults for any resources not specified by the job’s resource request. You can specify each default resource for each queue. This is described in section 5.9.3 “Specifying Job Default Resources” on page 338. Jobs inherit default resources according to the rules described in section 5.9.4 “Allocating Default Resources to Jobs” on page 342.

### 2.2.13 How Default Server and Queue Resources Are Applied When Jobs Move

When a job is moved from one server to another, the following changes happen:

- Any default resources that were applied by the first server are removed
- Default resources from the new server are applied to the job

When a job is moved from one queue to another, the following changes happen:

- Any default resources that were applied by the first queue are removed
- Default resources from the new queue are applied to the job

For more details on how default resources are inherited when a job is moved, see section 5.9.4.2 “Moving Jobs Between Queues or Servers Changes Defaults” on page 344.
2.2.14 Specifying Default Queue

PBS has a default execution queue named \textit{workq}, where jobs are placed when no queue is requested. You can specify which queue should be the default. To specify the queue which is to accept jobs when no queue is requested, set the server’s \texttt{default\_queue} attribute to the name of the queue:

\texttt{Qmgr: set server default\_queue = \langle queue name\rangle}

2.2.15 Associating Queues and Vnodes

You can set up vnodes so that they accept jobs only from specific queues. See section 4.8.2 “Associating Vnodes with Queues” on page 141.

2.2.16 Configuring Access to Queues

You can configure each queue so that only specific users or groups can submit jobs to the queue. See section 9.3 “Using Access Control” on page 659.

2.2.17 Setting Limits on Usage at Queues

You can set limits on different kinds of usage at each queue:

- You can limit the size of a job array using the \texttt{max\_array\_size} queue attribute
- You can limit the number of jobs or the usage of each resource by each user or group, or overall. See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410

2.2.18 Additional Queue Information

For a description of each queue attribute, see “Queue Attributes” on page 407 of the PBS Professional Reference Guide.
For information on using queues for scheduling, see section 4.5 “Using Queues in Scheduling” on page 132.
Chapter 2 Configuring the Server and Queues
Chapter 3

Configuring MOMs and Vnodes

The installation process creates a basic MOM and vnode configuration which contains the minimum necessary in order to run PBS jobs. This chapter describes how to customize your MOM and vnode configuration.

3.1 Vnodes: Virtual Nodes

A virtual node, or vnode, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. PBS views hosts as being composed of one or more vnodes.
Each vnode has an associated set of attributes and resources. Vnode attributes are listed and described in “Vnode Attributes” on page 420 of the PBS Professional Reference Guide. Vnode resources can be built-in or custom (defined by you.) See Chapter 5, "PBS Resources", on page 317. Rules for setting values for attributes and resources are given in section 3.5.2 “Choosing Configuration Method” on page 58.

### 3.1.1 Vnode State

The state of each vnode is controlled by its state attribute. The state of the vnode publishes whether the vnode can accept new jobs, what it is doing, and whether it is usable. The state attribute can take zero or more of the values listed in “Vnode States” on page 469 of the PBS Professional Reference Guide. The state of a vnode can be set by PBS or in a hook. A vnode’s state can be set to offline using the qmgr command; no other values can be set using qmgr.

### 3.1.2 Relationship Between Hosts, Nodes, and Vnodes

A host is any computer. Execution hosts used to be called nodes. However, some machines such as the Altix can be treated as if they are made up of separate pieces containing CPUs, memory, or both. Each piece is called a vnode. See “Vnodes: Virtual Nodes” on page 41. Some hosts have a single vnode and some have multiple vnodes. PBS treats all vnodes alike in most respects.

### 3.1.3 Natural Vnodes

For machines that have more than one vnode, there is a vnode called the natural vnode. A natural vnode does not correspond to any actual hardware. The natural vnode is used to define any placement set information that is invariant for a given host. See section 4.8.32 “Placement Sets” on page 237. The natural vnode is also used to define dynamic host-level resources, and can be used to define shared resources. On a multi-vnoded machine which has a natural vnode, anything set in the mom_resources line in PBS_HOME/sched_priv/sched_config is shared by all of
that machine’s vnodes. See section 5.14.5.1 “Dynamic Host-level
Resources” on page 374 and section 5.4.7 “Shared and Non-shared Vnode
Resources” on page 328.

3.1.4 Breaking Chunks Across Vnodes

Chunks can be broken up across vnodes that are on the same host. This is
generally used for jobs requesting a single chunk. On the Altix, the sched-
uler will share memory from a chunk even if all the CPUs are used by other
jobs. It will first try to put a chunk entirely on one vnode. If it can, it will
run it there. If not, it will break the chunk up across any vnode it can get
resources from, even for small amounts of unused memory.

3.1.4.1 Restrictions on Natural Vnode on cpuset
Machines

On a machine that has cpusets, the natural vnode should not have its sched-
ulable resources (ncpus, mem, vmem) set. Leave these resources unset.
If these are set by the administrator, their values are retained across restarts
until they are changed again or until the vnode is re-created. Setting the
values via qmgr will lead the Server and the MOM to disagree on the val-
ues.

3.1.5 Creating Vnodes

3.1.5.1 Creating Vnodes on Single-vnode Machines

For a machine which will have a single vnode, use the qmgr command to
create the vnode:

qmgr: create node <vnode name>
3.1.5.2 Creating Vnodes on Multi-vnode Machines

For a machine which will have more than one vnode, you create the natural vnode, but PBS handles creation of the other vnodes. Use the qmgr command to create the natural vnode:

```
Qmgr: create node <natural vnode name>
```

For machines such as an Altix, you must start PBS on the multi-vnode host using the PBS start/stop script. See section 6.1.3 “The PBS Start/Stop Script” on page 208 in the PBS Professional Installation & Upgrade Guide. After you create the natural vnode, the other vnodes become available for use. Follow the rules for configuring these machines in section 3.5.2 “Choosing Configuration Method” on page 58. See section 11.7.1 “Configuring MOM for an Altix Running Supported Versions of ProPack or SGI Performance Suite” on page 829 and section 11.7.6 “Configuring MOM on SGI ICE with ProPack and SGI Performance Suite” on page 835.

Here is an example of the vnode definition for a natural vnode on an Altix:

```
altix03: pnames = cbrick, router
altix03: sharing = ignore_excl
altix03: resources_available.ncpus = 0
altix03: resources_available.mem = 0
altix03: resources_available.vmem = 0
```

For machines such as a Cray, creation of vnodes other than the natural vnode is handled by MOM. You create the natural vnode using qmgr:

```
Qmgr: create node <natural vnode name>
```

3.1.5.2.1 Caveat for Creating Vnodes on the Cray

When creating a vnode to represent a login node, use the short name returned by the gethostname command on the login node. For example, if `gethostname` returns `HostA`, do the following:

```
Qmgr: create node HostA
```

If you create a vnode with a different name from the short name returned by gethostname, the following happens:

- MOM creates a vnode whose name is the short name returned by

gethostname

• The vnode you created is not recognized by MOM, and is marked *stale*

3.1.6 Deleting Vnodes

3.1.6.1 Deleting the Vnode on a Single-vnode Machine

Use the `qmgr` command to delete the vnode:

```
Qmgr: delete node <vnode name>
```

Optionally, you can stop PBS on the execution host whose vnode was deleted.
3.1.6.2 Deleting Vnodes on a Multi-vnode Machine

As long as there is a configuration file describing vnodes, PBS will believe they exist. Therefore, you must first remove the configuration file. To delete one or more vnodes on a multi-vnode machine, follow these steps:

1. Use the `-s remove` option to the `pbs_mom` command to remove the Version 2 configuration file that describes the vnodes to be removed:

   On UNIX/Linux:
   
   ```
   pbs_mom -s remove <configuration file target>
   ```

   On Windows:
   
   ```
   pbs_mom -N -s remove <configuration file target>
   ```

2. Use the `-s insert` option to the `pbs_mom` command to insert a new Version 2 configuration file describing the vnodes to be retained:

   On UNIX/Linux:
   
   ```
   pbs_mom -s insert <configuration file target> <input file source>
   ```

   On Windows:
   
   ```
   pbs_mom -N -s insert <configuration file target> <input file source>
   ```

3. Restart the MOM:

   ```
   <path to start/stop script>/pbs restart
   ```

4. Use the `qmgr` command to remove the vnodes:

   ```
   qmgr: delete node <vnode name>
   ```

3.1.6.3 Deleting Vnodes on a Cray

For information on deleting vnodes on a Cray, see section 11.5.11.4 “Deleting Vnodes on Cray” on page 823.
3.1.7 Allocating Vnodes to Jobs

PBS can run jobs only on the execution hosts that are managed by the PBS server, and running a MOM.

By default, when the scheduler looks for the vnodes on which to run a job, it goes down the list of hosts in the order in which they appear in the server’s list of hosts, and places the job on the first available vnode or vnodes meeting the job’s requirements. This means that the order of the list of hosts affects default job placement. You can specify more sophisticated choices; see Chapter 4, "Scheduling", on page 71.

The scheduler follows the specified rules for selecting vnodes that match each job’s request. Once the scheduler finds the resources that match a job’s request, it allocates vnodes to the job, according to the value of the vnode’s sharing attribute and the job’s resource request.

3.1.7.1 Sharing Vnodes Among Jobs

Each vnode can be allocated exclusively to one job, or its resources can be shared among jobs. Hosts can also be allocated exclusively to one job, or shared among jobs.

How vnodes are allocated to jobs is determined by a combination of the vnode’s sharing attribute and the job’s resource request. The possible values for the vnode sharing attribute, and how they interact with a job’s placement request, are described in “sharing” on page 426 of the PBS Professional Reference Guide. A description of how resources are allocated is in section 4.8.40 “Shared vs. Exclusive Use of Resources by Jobs” on page 298.

If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes job-exclusive. No other job can use the vnode.

If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either default_exclhost or force_exclhost, all vnodes on that host must have the same value for the sharing attribute. When the MOM starts or restarts,
if any vnode on a host is set to either default_exclhost or force_exclhost, and another vnode is set to a different value, the MOM will exit and log the following error message at event class 0x0001:

   It is erroneous to mix sharing= <sharing val> for vnode <name> with sharing= <force_exclhost|default_exclhost> which is set for other vnodes on host <host>

### 3.1.7.2 Placing Jobs on Vnodes

Jobs can be placed on vnodes according to the job’s placement request. Each chunk from a job can be placed on a different host, or a different vnode. Alternatively, all chunks can be taken from a single host, or from chunks sharing the same value for a specified resource. The job can request exclusive use of each vnode, or shared use with other jobs. The job can request exclusive use of its hosts. For details, see section 3.6, "Placing Jobs on Vnodes", on page 50 of the PBS Professional User’s Guide.

### 3.2 MOMs

A MOM daemon runs on each execution host and manages the jobs on that execution host. The `pbs_mom` command starts the PBS job monitoring and execution daemon, called MOM. The `pbs_mom` daemon starts jobs on the execution host, monitors and reports resource usage, enforces resource usage limits, and notifies the server when the job is finished. The MOM also runs any prologue scripts before the job runs, and runs any epilogue scripts after the job runs.

When the MOM starts a job, she creates a new session that is as identical to the user’s login session as is possible. For example, under UNIX, if the user’s login shell is `csh`, then MOM creates a session in which `.login` is run as well as `.cshrc`. MOM returns the job’s output to the user.

The MOM performs any communication with job tasks and with other MOMs. The MOM on the first vnode on which a job is running manages communication with the MOMs on the remaining vnodes on which the job runs. The MOM on the first vnode is called *Mother Superior*. 
The MOM log file is in PBS_HOME/mom_logs. The MOM writes an error message in its log file when it encounters any error. The MOM also writes other miscellaneous information to its log file. If it cannot write to its log file, it writes to standard error.

The executable for pbs_mom is in PBS_EXEC/sbin, and can be run only by root.

See section 6.1.4.3 “Manually Starting MOM” on page 210 in the PBS Professional Installation & Upgrade Guide for information on starting and stopping MOM.

3.2.1 Single-vnode, Multi-vnode, and Cpusetted Systems

For systems that can be subdivided into more than one virtual node, or vnode, PBS manages each vnode much as if it were a host. On each machine, the MOM manages the vnodes. PBS may treat a host such as an Altix as a set of virtual nodes, in which case one MOM manages all of the host's vnodes. For details about vnodes, see section 3.1 “Vnodes: Virtual Nodes” on page 41.

The pbs_mom you select to run a machine depends on the type of machine and the way you want it managed. The MOM that manages a system without cpusets is pbs_mom.standard. This MOM can manage a single-vnoded or a multi-vnoded, non-cpusetted system. The MOM that has extensions to manage a cpusetted machine such as the Altix is pbs_mom.cpuset. The appropriate MOM is copied to pbs_mom. See the PBS Professional Installation and Upgrade Guide.

The following sections describe configuration files and methods for all MOMs and vnodes. See section 11.6 “Configuring MOM for Machines with cpusets” on page 827 for information that is specific to systems with cpusets.
3.3 Files and Directories Used by MOM

If `PBS_MOM_HOME` is present in the `pbs.conf` file, `pbs_mom` will use that directory for its “home” instead of `PBS_HOME`. Under UNIX/Linux, all files and directories that MOM uses must be owned by root. Under Windows, these directories must have at least Full Control permission for the local Administrators group. MOM uses the following files and directories:

UNIX:

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Description</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>aux</td>
<td>Directory</td>
<td>0755</td>
</tr>
<tr>
<td>checkpoint</td>
<td>Directory</td>
<td>0700</td>
</tr>
<tr>
<td>checkpoint script</td>
<td>File</td>
<td>0755</td>
</tr>
<tr>
<td>mom_logs</td>
<td>Directory</td>
<td>0755</td>
</tr>
<tr>
<td>mom_priv</td>
<td>Directory</td>
<td>0751</td>
</tr>
<tr>
<td>mom_priv/jobs</td>
<td>Directory</td>
<td>0751</td>
</tr>
<tr>
<td>mom_priv/config</td>
<td>File</td>
<td>0644</td>
</tr>
<tr>
<td>mom_priv/prologue</td>
<td>File</td>
<td>0755</td>
</tr>
<tr>
<td>mom_priv/epilogue</td>
<td>File</td>
<td>0755</td>
</tr>
<tr>
<td>pbs_environment</td>
<td>File</td>
<td>0644</td>
</tr>
<tr>
<td>spool</td>
<td>Directory</td>
<td>1777 (drwxrwxrwt)</td>
</tr>
<tr>
<td>undelivered</td>
<td>Directory</td>
<td>1777 (drwxrwxrwt)</td>
</tr>
<tr>
<td>Version 2 configuration files</td>
<td>Files</td>
<td>0755</td>
</tr>
<tr>
<td>PBS reserved configuration files</td>
<td>Files</td>
<td>----</td>
</tr>
</tbody>
</table>
### Table 3-1: MOM Files and Directories Under UNIX

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Description</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>auxiliary</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>checkpoint</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group</td>
</tr>
<tr>
<td>checkpoint script</td>
<td>File</td>
<td>At least Full Control permission for the local Administrators group</td>
</tr>
<tr>
<td>mom_logs</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>mom_priv</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>mom_priv/jobs</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>mom_priv/config</td>
<td>File</td>
<td>At least Full Control permission for the local Administrators group</td>
</tr>
<tr>
<td>pbs_environment</td>
<td>File</td>
<td>At least Full Control permission for the local Administrators group and read-only to Everyone</td>
</tr>
<tr>
<td>spool</td>
<td>Directory</td>
<td>Full access to Everyone</td>
</tr>
<tr>
<td>undelivered</td>
<td>Directory</td>
<td>Full access to Everyone</td>
</tr>
</tbody>
</table>

### Table 3-2: MOM Files and Directories Under Windows

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Description</th>
<th>Ownership/Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job temporary directory</td>
<td>Directory</td>
<td>1777</td>
</tr>
</tbody>
</table>

Windows:

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Description</th>
<th>Ownership/Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>auxiliary</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>checkpoint</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group</td>
</tr>
<tr>
<td>checkpoint script</td>
<td>File</td>
<td>At least Full Control permission for the local Administrators group</td>
</tr>
<tr>
<td>mom_logs</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>mom_priv</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>mom_priv/jobs</td>
<td>Directory</td>
<td>At least Full Control permission for the local Administrators group and read-only access to Everyone</td>
</tr>
<tr>
<td>mom_priv/config</td>
<td>File</td>
<td>At least Full Control permission for the local Administrators group</td>
</tr>
<tr>
<td>pbs_environment</td>
<td>File</td>
<td>At least Full Control permission for the local Administrators group and read-only to Everyone</td>
</tr>
<tr>
<td>spool</td>
<td>Directory</td>
<td>Full access to Everyone</td>
</tr>
<tr>
<td>undelivered</td>
<td>Directory</td>
<td>Full access to Everyone</td>
</tr>
</tbody>
</table>
Chapter 3 Configuring MOMs and Vnodes

3.4 MOM and Vnode Configuration Files

The behavior of each MOM is controlled through its configuration files. You configure MOMs by specifying values for parameters in configuration files.

Vnodes are controlled through the values of their attributes. You configure vnodes by specifying values for vnode attributes, either by setting attribute values using the qmgr command or by creating configuration files using the pbs_mom -s insert command (pbs_mom -N -s insert on Windows).

The method to use to configure MOMs and vnodes depends on the machine being configured. The methods used are described in section 3.5 “How to Configure MOMs and Vnodes” on page 56.

3.4.1 Types of MOM and Vnode Configuration Files

MOM and vnode configuration information can be contained in configuration files of three types:

- Version 1
- PBS reserved
- Version 2
3.4.1.1 Version 1 Configuration Files

You edit the Version 1 configuration file directly. The Version 1 configuration file is usually \texttt{PBS\_HOME/mom\_priv/config}. This file contains the parameters that control MOM’s behavior.

The Version 1 configuration file must be secure. It must be owned by a user ID and group ID both less than 10 and must not be world-writable.

For a complete description of the syntax and contents of the Version 1 configuration file, see “MOM Parameters” on page 311 of the PBS Professional Reference Guide.

3.4.1.2 PBS Reserved Configuration Files

PBS reserved configuration files are created by PBS and are prefixed with "PBS". These files are created by PBS and are not configurable. Do not attempt to edit these files. An attempt to create or remove a file with the "PBS" prefix will result in an error.

3.4.1.3 Version 2 Configuration Files

Version 2 configuration files are those created by the site administrator. These files can contain vnode attribute settings. Do not attempt to edit these files directly. Instead of editing these directly, you create a local file and give it as an argument to the \texttt{pbs\_mom \(-s\) insert} option (\texttt{pbs\_mom \(-N\) \(-s\) insert} on Windows), and PBS creates a new configuration file for you. See section 3.5.3 “Creating Version 2 MOM Configuration Files” on page 60. Their syntax is called “Version 2” in order to differentiate it from the syntax of the Version 1 configuration files.

You can list, add, delete and display Version 2 configuration files using the \texttt{pbs\_mom \(-s\)} option (\texttt{pbs\_mom \(-N\) \(-s\) on Windows). See “\texttt{pbs\_mom}” on page 66 of the PBS Professional Reference Guide for information about \texttt{pbs\_mom} options.

3.4.1.3.1 Removing Version 2 Configuration Files

You can remove a Version 2 configuration file using the \texttt{pbs\_mom \(-s\) remove} option (\texttt{pbs\_mom \(-N\) \(-s\) remove} on Windows). See “\texttt{pbs\_mom}” on page 66 of the PBS Professional Reference Guide.
3.4.2 Location of MOM’s Configuration Files

The Version 1 configuration file is usually PBS_HOME/mom_priv/config. It can be in a different location; in that case, MOM must be started with the -c option. See “pbs_mom” on page 66 of the PBS Professional Reference Guide.

PBS places PBS reserved and Version 2 configuration files in an area that is private to each installed instance of PBS.

3.4.3 Listing and Viewing PBS Reserved and Version 2 Configuration Files

You can list and view the PBS reserved configuration files and the Version 2 configuration files using the pbs_mom -s list and pbs_mom -s show options (pbs_mom -N -s list and show on Windows). See “pbs_mom” on page 66 of the PBS Professional Reference Guide.

3.4.4 Caveats and Restrictions for Configuration Files

- Do not attempt to directly create PBS reserved or Version 2 configuration files; instead, use the pbs_mom -s option (pbs_mom -N -s on Windows).
- Note that the -d option to pbs_mom changes where MOM looks for PBS_HOME, and using this option will change where MOM looks for all configuration files. If you use the -d option, MOM will look in the new location for any PBS reserved and Version 2 files.
- The -c option will change which Version 1 configuration file MOM reads.
- Do not move PBS reserved configuration files.
- If you set a value using qmgr, this value overrides the value specified in a configuration file.
- Do not mix the configuration file contents or syntax. Each type must
use its own syntax, and contain its own type of information.

- When you create a Version 2 configuration file for a pre-existing vnode, make sure it specifies all of the information about the vnode, such as resources and attribute settings. The creation of the configuration file overrides previous settings, and if the new file contains no specification for a resource or attribute, that resource or attribute becomes unset.

- Version 2 configuration files can be moved from one installed instance of PBS to another. To move a set of Version 2 configuration files from one installed instance of PBS to another:
  1. Use the `-s list` directive with the "source" instance of PBS to enumerate the Version 2 files.
  2. Use the `-s show` directive with each Version 2 file of the "source" instance of PBS to save a copy of that file.
  3. Use the `-s insert` directive with each file at the "target" instance of PBS to create a copy of each Version 2 configuration file.

### 3.4.4.1 When MOM Reads Configuration Files

MOM reads the configuration files at startup and reinitialization. On UNIX, this is when `pbs_mom` receives a SIGHUP signal or is started or restarted, and on Windows, when MOM is started or restarted. In order for any configuration changes to take effect, MOM must be HUPed.

If you make changes to the hardware or a change occurs in the number of CPUs or amount of memory that is available to PBS, such as a non-PBS process releasing a cpuset, you should restart PBS, by typing the following:

```
<path-to-script>/pbs restart
```

The MOM daemon is normally started by the PBS start/stop script.

When MOM is started, it will open its Version 1 configuration file, `mom_priv/config`, in the path specified in `pbs.conf`, if the file exists. If it does not, MOM will continue anyway. The `config` file may be placed elsewhere or given a different name, by starting `pbs_mom` using the `-c` option with the new file and path specified. See section 6.1.4.3 “Manually Starting MOM” on page 210 in the PBS Professional Installation & Upgrade Guide.
The files are processed in this order:

1. Version 1 configuration file
2. PBS reserved configuration files
3. Version 2 configuration files

Within each category, the files are processed in lexicographic order.

The contents of a file that is read later will override the contents of a file that is read earlier.

### 3.5 How to Configure MOMs and Vnodes

#### 3.5.1 Configuration Methods

The method you use to configure MOMs and vnodes depends upon the machine being configured. The methods are the following:

<table>
<thead>
<tr>
<th>Method</th>
<th>When Method Changes MOM Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the <code>qmgr</code> command to set attribute values</td>
<td>Immediately</td>
</tr>
<tr>
<td>Editing the Version 1 configuration file <code>PBS_HOME/mom_priv/config</code></td>
<td>When MOM is restarted</td>
</tr>
<tr>
<td>Using the <code>pbs_mom -s insert</code> command to create a configuration file</td>
<td>When MOM is restarted</td>
</tr>
<tr>
<td>(<code>pbs_mom -N -s insert</code> on Windows)</td>
<td></td>
</tr>
<tr>
<td>Using the <code>pbsnodes</code> command to change the state of a vnode</td>
<td>Immediately</td>
</tr>
</tbody>
</table>
3.5.1.1 The qmgr Command

You use the qmgr command to set attribute values. You can use the qmgr command to set attribute values for individual vnodes where those vnodes are part of a multi-vnode machine.

To set a vnode’s attribute, the format is the following:

```
qmgr -c 'set node <vnode name> <attribute> = <value>'
```

or start qmgr, and use the following:

```
set node <vnode name> <attribute> = <value>
```

The qmgr command is described in “qmgr” on page 178 of the PBS Professional Reference Guide.

If you set a value using qmgr, this value overrides the value specified in a configuration file.

3.5.1.2 Editing Version 1 Files

Use your favorite text editor to edit Version 1 configuration files.

3.5.1.3 Using the pbs_mom -s insert Command

You use the pbs_mom -s insert command (pbs_mom -N -s insert on Windows) to create all Version 2 configuration files. First, you create a script which is to be the contents of the configuration file. Then, you insert the script using the pbs_mom -s insert command:

UNIX/Linux:

```
pbs_mom -s insert <script> <configuration file name>
```

Windows:

```
pbs_mom -N -s insert <script> <configuration file name>
```

For a description of the Version 2 syntax, see section 3.5.3 “Creating Version 2 MOM Configuration Files” on page 60.
3.5.1.4 Using the pbsnodes Command

The pbsnodes command is used to set the state of a host to be offline or not offline. To set the state attribute of one or more hosts to offline:

```bash
pbsnodes -o <hostname [hostname ...]>
```

To remove the offline setting from the state attribute of one or more hosts:

```bash
pbsnodes -r <hostname [hostname ...]>
```

Note that the pbsnodes command operates on hosts, not individual vnodes where those vnodes are on multi-vnode machines. To operate on individual vnodes, use the qmgr command.

See “pbsnodes” on page 123 of the PBS Professional Reference Guide.

3.5.2 Choosing Configuration Method

3.5.2.1 Configuring Single-vnode Machines without cpusets

To configure the MOM and vnode on a single-vnode machine without cpusets, do the following:

- To configure MOM, including local resources, edit the Version 1 MOM parameter file
- To configure vnodes, use the qmgr command to set vnode attributes and global resources

3.5.2.1.1 Exceptions

- Use `pbs_mom -s insert` (pbs_mom -N -s insert on Windows) to set the sharing vnode attribute
Configuring Multi-vnode Machines without cpusets

To configure the MOM and vnodes on a multi-vnode machine without cpusets, such as an HPCBP MOM host, do the following:

• To configure MOM, including local resources, edit the Version 1 MOM parameter file
• To configure vnodes, use the `qmgr` command to set vnode attributes and global resources

3.5.2.2.1 Exceptions

• Use the `pbs_mom -s insert` command (`pbs_mom -N -s insert` on Windows) to set the sharing vnode attribute (vnode definition files are not recommended on Cray)
• You can use `pbsnodes` to set the `state` vnode attribute

3.5.2.2.2 Restrictions

• Set the `Mom` vnode attribute for the natural vnode only.
• On non-HPCBP MOM hosts, set values for the `resources_available` vnode attribute on the natural vnode only.
• On the HPCBP MOM host, do not attempt to set values for `resources_available.mem`, `resources_available.vmem`, or `resources_available.ncpus`.

3.5.2.3 Configuring Machines with cpusets

To configure the MOM and vnodes on a machine that has cpusets, do the following:

• To configure MOM, including local resources, edit the Version 1 MOM parameter file
• To configure vnodes, use the `pbs_mom -s insert` command (`pbs_mom -N -s insert` on Windows) to set vnode attributes and global resources.
3.5.2.3.1 Exceptions

• You can use `qmgr` or `pbsnodes` to set the `state` vnode attribute
• Use `qmgr` to set the `priority` vnode attribute

3.5.2.3.2 Restrictions

• Do not use `qmgr` to configure vnodes, especially for sharing, `resources_available.ncpus`, `resources_available.vmem`, and `resources_available.mem`.
• Do not attempt to set values for `resources_available.ncpus`, `resources_available.vmem`, or `resources_available.mem`. These are set by PBS when the topology file is read.
• Set the `Mom` vnode attribute for the natural vnode only. Do not attempt to set it for any other vnodes.

3.5.3 Creating Version 2 MOM Configuration Files

3.5.3.1 Operating on Version 2 Configuration Files

You can list, add, delete and display Version 2 configuration files using the `pbs_mom -s` option (`pbs_mom -N -s` on Windows). See “`pbs_mom`” on page 66 of the PBS Professional Reference Guide for information about `pbs_mom` options.

3.5.3.2 Format of Version 2 Configuration Files

Any Version 2 configuration file must begin with this line:

```
$configversion 2
```

The format of the remaining contents of the file is the following:

```
<vnode ID> : <attribute name> = <attribute value>
```

where

**Table 3-4: Elements in Version 2 Reserved Configuration Files**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;vnode ID&gt;</td>
<td>Sequence of characters not including a colon (&quot;.&quot;), unique per vnode</td>
</tr>
<tr>
<td>&lt;attribute name&gt;</td>
<td>Sequence of characters beginning with alphabets or numerics, with underscore (&quot;_&quot;) and dash (&quot;-&quot;)</td>
</tr>
<tr>
<td>&lt;attribute value&gt;</td>
<td>Sequence of characters not including an equal sign (&quot;=&quot;), unique per vnode</td>
</tr>
</tbody>
</table>

The colon and equal sign may be surrounded by white space.

A vnode's ID is an identifier that will be unique across all vnodes known to a given `pbs_server` and will be stable across reinitializations or invocations of `pbs_mom`. ID stability is important when a vnode's CPUs or memory might change over time and PBS is expected to adapt to such changes by resuming suspended jobs on the same vnodes to which they were originally assigned. Vnodes for which this is not a consideration may simply use IDs of the form "0", "1", etc. concatenated with some identifier that ensures uniqueness across the vnodes served by the `pbs_server`. Vnode attributes cannot be used as vnode names. See “Vnode Attributes” on page 420 of the PBS Professional Reference Guide, where vnode attributes are listed.

### 3.5.3.3 Using the `pbs_mom -s insert` Command

To create a Version 2 configuration file:

1. Create the script that is to be the contents of the configuration file
2. Make this script into a configuration file using the `pbs_mom -s insert` command.

Example 1: If your machine has 4 vnodes, named `BigNode0`, `BigNode1`, `SmallNode0`, and `SmallNode1`, and you want big jobs to have exclusive use of their vnodes, and small jobs to share their vnodes, then set
Configuring MOMs and Vnodes

sharing for big and small vnodes by creating a file "set_sharing" containing the following:

$configversion 2

BigNode0: sharing = default_excl
BigNode1: sharing = default_excl
SmallNode0: sharing = default_shared
SmallNode1: sharing = default_shared

Then use the pbs_mom -s insert <filename> <script> option to create the configuration file:

UNIX/Linux:

```
pbs_mom -s insert sharing_config set_sharing
```

Windows:

```
pbs_mom -N -s insert sharing_config set_sharing
```

The script sharing_config is the new Version 2 configuration file. Its contents will override previously-read sharing settings. You must restart the MOM after changing the configuration file.
Example 2: To change the sharing attribute on the host named `host3`:

1. Check that `pbsnodes` shows `host3` has “sharing = default_shared”:
   ```bash
   pbsnodes host3
   ```
2. Change the setting to be “sharing = force_excl”:
   
   As root, create a script file `/tmp/excl_file` which contains the following:
   ```bash
   $configversion 2
   <host3>: sharing=force_excl
   ```
3. With the `pbs_mom` daemon running, execute the following on `host3`:
   
   UNIX/Linux:
   ```bash
   # $PBS_EXEC/sbin/pbs_mom -s insert excl /tmp/excl_file
   ```
4. Check that this took effect. The following should show "excl".
   
   UNIX/Linux:
   ```bash
   # $PBS_EXEC/sbin/pbs_mom -s list
   ```
5. Restart `pbs_mom`:
   ```bash
   # kill -HUP <PID of pbs_mom>
   ```
6. Check that “`pbsnodes host3`” now shows “sharing = force_excl”

### 3.5.3.4 Caveats and Restrictions for `pbs_mom -s insert`

On Windows, the `pbs_mom -s` option must be used with the `-N` option so that MOM will start in standalone mode.

### 3.5.4 Using `qmgr` to Set Vnode Resources and Attributes

One of the PBS reserved configuration files is `PBSvnodedefs`, which is created by a placement set generation script. You can use the output of the placement set generation script to produce input to `qmgr`. The placement
set generation script normally emits data for the PBSvnodedef file. If the script is given an additional “-v type=q” argument it emits data in a form suitable for input to qmgr:

```
set node <ID> resources_available.<ATTRNAME> = <ATTR-VALUE>
```

where <ID> is a vnode identifier unique within the set of hosts served by a pbs_server. Conventionally, although by no means required, the <ID> above will look like HOST[<localID>] where HOST is the host's FQDN stripped of domain suffixes and <localID> is an identifier whose meaning is unique to the execution host on which the referred to vnode resides. For invariant information, it will look like this:

```
set node <ID> priority = 2
```

### 3.5.5 Caveats and Advice on Configuring MOMs and Vnodes

#### 3.5.5.1 Changing Resource Settings

In general, it is not advisable to set `resources_available.ncpus` or `resources_available.mem` to a value greater than PBS has detected on the machine. This is because you do not want MOM to try to allocate more resources than are available.

In general, it is safe to set `resources_available.ncpus` or `resources_available.mem` to a value less than PBS has detected.
3.6 Configuring MOM and Vnode Features

3.6.1 Configuring MOM’s Polling Cycle

3.6.1.1 Polling on UNIX/Linux

In this section, we describe how to configure MOM’s polling cycle. Please note that polling intervals cannot be considered to be exact:

- The calculation below simply provides a minimum amount of time between one poll and the next.
- The actual time between polls can vary. The actual time taken by MOM also depends on the other tasks MOM is performing, such as starting jobs, running a prologue or epilogue, etc.
- The timing of MOM’s activities is not completely under her control, because she is a user process.
- The finest granularity for calculating polling is in seconds.

MOM’s polling cycle is determined by the values of $min\_check\_poll$ and $max\_check\_poll$. The interval between each poll starts at $min\_check\_poll$ and increases with each cycle until it reaches $max\_check\_poll$, after which it remains the same. The amount by which the cycle increases is the following:

\[
\frac{(\text{max\_check\_poll} - \text{min\_check\_poll} + 19)}{20}
\]

The default value for $max\_check\_poll$ is 120 seconds. The minimum is 1 second. It is not recommended to set $max\_check\_poll$ to less than 30 seconds.

The default value for $min\_check\_poll$ is 10 seconds. The minimum is 1 second. It is not recommended to set $min\_check\_poll$ to less than 10 seconds.

The start of a new job resets the polling for all of the jobs being managed by this MOM.
MOM polls for resource usage for cput, walltime, mem and ncpus. See section 5.15.3 “Placing Resource Limits on Jobs” on page 436.

### 3.6.1.2 Polling on Windows

On Windows, MOM updates job usage at fixed intervals of 10 seconds. The $min\_check\_poll and $max\_check\_poll parameters are not used by MOM on Windows. MOM looks for any job that has exceeded a limit for walltime, mem, or cput, and terminates jobs that have exceeded the limit.

#### 3.6.1.2.1 Windows Polling Caveats

The ncpus resource cannot be tracked in Windows.

### 3.6.1.3 How Polling is Used

Job-wide limits are enforced by MOM using polling. See section 5.15.3.3.1 “Job Memory Limit Enforcement on UNIX” on page 439. MOM can enforce cpuaverage and cpuburst resource usage. See section 5.15.3.4.1 “Average CPU Usage Enforcement” on page 442 and section 5.15.3.4.2 “CPU Burst Usage Enforcement” on page 443.

MOM enforces the $restrict\_user access restrictions on a polling cycle which can be set to a maximum of 10 seconds. See section 3.6.6 “Restricting User Access to Execution Hosts” on page 68.

Cycle harvesting has its own polling interval. See “$kbd\_idle <idle\_wait> <min\_use> <poll\_interval>” on page 317 of the PBS Professional Reference Guide for information on $kbd\_idle.

### 3.6.1.4 Recommendations for Polling Interval

Do not set $max\_check\_poll to less than 30 seconds.

Do not set $min\_check\_poll to less than 10 seconds.

If you have many small jobs, frequent polling can take up a lot of MOM’s cycles. You may want to set $min\_check\_poll and $max\_check\_poll to somewhat higher values.
3.6.2 Configuring Host-level Resources

Before configuring host-level resources, consider how you will use them. When configuring static resources, it is best to configure global static resources. Even though they are global, they can be configured at the host level. Global resources can be operated on via the `qmgr` command and viewed via the `qstat` command. When configuring dynamic resources, if you need the script to run at the execution host, configure local dynamic resources. These resources cannot be operated on via the `qmgr` command or viewed via the `qstat` command.

3.6.2.1 Configuring Global Static Vnode Resources

You can create global custom static host-level resources that can be reported by MOM and used for jobs. Follow the instructions in section 5.14.5.2 “Static Host-level Resources” on page 377.

You can set values for built-in and custom global static vnode resources according to the rules in section 3.5.2 “Choosing Configuration Method” on page 58.

3.6.2.1.1 Configuring Local Dynamic Vnode Resources

You can create local custom dynamic host-level resources. The primary use of this feature is to add site-specific resources, such as software application licenses or scratch space. Follow the instructions in section 5.14.5.1 “Dynamic Host-level Resources” on page 374.

3.6.3 Manual Creation of cpusets Not Managed by PBS

You may wish to create cpusets not managed by PBS on an Altix running supported versions of ProPack or SGI Performance Suite. If you have not started PBS, create these cpusets before starting PBS. If you have started PBS, requeue any jobs, stop PBS, create your cpuset(s), then restart PBS.
3.6.4 Configuring Site-Specific Job Termination

For information on site-specific job termination, see section 12.7.4 “Configuring Site-specific Job Termination” on page 854.

3.6.5 Job Checkpoint and Restart

If you want support for job checkpoint and restart, you can configure MOM to run checkpoint and restart scripts. See section 10.3 “Checkpoint and Restart” on page 735.

3.6.6 Restricting User Access to Execution Hosts

PBS provides a facility to prevent users who are not running PBS jobs from using machines controlled by PBS. You can turn this feature on by using the $restrict_user MOM directive. This directive can be fine-tuned by using the $restrict_user_exceptions and $restrict_user_maxsysid MOM directives. This feature can be set up host by host.

- A user requesting exclusive access to a set of hosts (via place=excl) can be guaranteed that no other user will be able to use the hosts assigned to his job, and PBS will not assign any unallocated resources on the vnode to another job.

- A user requesting non-exclusive access to a set of hosts can be guaranteed that no non-PBS users are allowed access to the hosts.

- A privileged user can be allowed access to the complex such that they can log into a host without having a job active.

- An abusive user can be denied access to the complex hosts.

The administrator can find out when users try to access hosts without going through PBS. The administrator can ensure that application performance is consistent on a complex controlled by PBS. PBS will also be able to clean up any job processes remaining after a job finishes running. The log event class for messages concerning restricting users is 0x0002.
For a vnode with access restriction turned on:

- Any user not running a job who logs in or otherwise starts a process on that vnode will have his processes terminated.

- A user who has logged into a vnode where he owns a job will have his login terminated when the job is finished.

- When MOM detects that a user that is not exempt from access restriction is using the system, that user's processes are killed and a log message is output:

  01/16/2006 22:50:16;0002;pbs_mom;Svr;restrict_user;
  killed uid 1001 pid 13397(bash) with log event class PBSE_SYSTEM.

You can set up a list of users who are exempted from the restriction via the $restrict_user_exceptions directive. This list can contain up to 10 user names.

Example 1: Turn access restriction on for a given node:

$restrict_user on

Example 2: Limit the users affected to those with a user ID greater than 500:

$restrict_user_maxsysid 500

Example 3: Exempt specific users from the restriction:

$restrict_user_exceptions userA, userB, userC

Note that a user who has a job running on a particular host will be able to log into that host.

3.6.6.1 Windows Restriction

The user access restriction feature is not supported on Windows.

3.6.7 Vnode Resources Set by MOM

If the following vnode resources are not explicitly set, they will take the value provided by MOM. But if they are explicitly set, that setting will be carried forth across Server restarts.
They are:

- `resources_available.ncpus`
- `resources_available.arch`
- `resources_available.mem`

### 3.6.8 Vnode Comments

Vnodes have a `comment` attribute which can be used to display information about that vnode. If the `comment` attribute has not been explicitly set by the PBS Manager and the vnode is down, it will be used by the PBS Server to display the reason the vnode was marked down. If the Manager has explicitly set the attribute, the Server will not overwrite the comment. The `comment` attribute may be set via the `qmgr` command:

```
Qmgr: set node pluto comment="node will be up at 5pm"
```

Once set, vnode comments can be viewed via `pbsnodes`, `xpbsmon` (vnode detail page), and `qmgr`. See “pbsnodes” on page 123 of the PBS Professional Reference Guide, and “xpbsmon” on page 292 of the PBS Professional Reference Guide.
Chapter 4

Scheduling

The Scheduling Policy Basics section of this chapter describes what PBS can do, so that you can consider these capabilities when choosing how to schedule jobs. The Choosing a Policy section describes how PBS can meet the scheduling needs of various workloads. The Scheduling Tools section describes each scheduling tool offered by PBS.

4.1 Chapter Contents

4.1 Chapter Contents ............................................ 71
4.2 Scheduling Policy Basics ................................. 74
  4.2.1 How Scheduling Can Be Used ..................... 74
  4.2.2 What is Scheduling Policy? ......................... 74
  4.2.3 Basic PBS Scheduling Behavior .................. 75
  4.2.4 Sub-goals .................................................. 75
  4.2.5 Job Prioritization ....................................... 76
4.2 Scheduling Policy Basics

4.2.1 How Scheduling Can Be Used

You can use the scheduling tools provided by PBS to implement your chosen scheduling policy, to make your jobs run in the way you want.

Your policy can do the following:

- Prioritize jobs according to your specification
- Run jobs according to their relative importance
- Award specific amounts of resources such as CPU time or licenses to users and groups according to rules that you set
- Make sure that resources are not misused
- Optimize how jobs are placed on vnodes, so that jobs run as efficiently as possible
- Use special time slots for particular tasks
- Optimize throughput or turnaround time for jobs

4.2.2 What is Scheduling Policy?

Scheduling policy determines when each job is run and on which resources. In other words, a scheduling policy describes a goal, or intended behavior. For convenience, we describe a scheduling policy as being a combination of sub-goals, for example a combination of how resources should be allocated and how efficiency should be maximized.

You implement a scheduling policy using the tools PBS provides. A scheduling tool is a feature that allows you control over some aspect of scheduling. For example, the job sorting formula is a tool that allows you to define how you want job execution priority to be computed. Some scheduling tools are supplied by the PBS scheduler, and some are supplied by other elements of PBS, such as the server, queues or resources.
4.2.3 Basic PBS Scheduling Behavior

The basic behavior of PBS is that it always places jobs where it finds the resources requested by the job. PBS will not place a job where that job would use more resources than PBS thinks are available. For example, if you have two jobs, each requesting 1 CPU, and you have one vnode with 1 CPU, PBS will run only one job at a time on the vnode. You do not have to configure PBS for this basic behavior.

However, you do have to inform PBS which resources are available and where, how much, and whether they are consumable or not. In addition, in order to ensure that jobs are sent to the appropriate vnodes for execution, you also need to make sure that they request the correct resources. You can do this either by having users submit their jobs with the right resource requests, or by configuring default resources for jobs to inherit.

4.2.4 Sub-goals

Your scheduling policy is the combination that you choose of one or more sub-goals. For example, you might need to meet two particular sub-goals: you might need to prioritize jobs a certain way, and you might need to use resources efficiently. You can choose among various outcomes for each sub-goal. For example, you can choose to prioritize jobs according to size, owner, owner’s usage, time of submission, etc.

In the following sections, we describe the tools PBS offers for meeting each of the following sub-goals.

• Job prioritization; see section 4.2.5 “Job Prioritization” on page 76.
• Resource allocation & limits; see section 4.2.6 “Resource Allocation to Users, Projects & Groups” on page 85.
• Time slot allocation; see section 4.2.7 “Time Slot Allocation” on page 88.
• Job placement optimizations; see section 4.2.8 “Job Placement Optimization” on page 90.
• Resource efficiency optimizations; see section 4.2.9 “Resource Efficiency Optimizations” on page 95.
• Overrides; see section 4.2.10 “Overrides” on page 99.
4.2.5 Job Prioritization

Job prioritization is any technique you use to come up with a ranking of each job’s relative importance. You can specify separate priority schemes for both execution and preemption.

4.2.5.1 Where PBS Uses Job Priority

PBS calculates job priority for two separate tasks: job execution and job preemption. Job execution priority is used with other factors to determine when to run each job. Job preemption priority is used to determine which queued jobs are allowed to preempt which running jobs in order to use their resources and run. These two tasks are independent, and it is important to make sure that you do not make them work at cross-purposes. For example, you do not want to have a class of jobs having high execution priority and low preemption priority; these jobs would run first, and then be preempted first.

Preemption comes into play when the scheduler examines the top job and determines that it cannot run now. If preemption is enabled, the scheduler checks to see whether the top job has sufficient preemption priority to be able to preempt any running jobs, and then if it does, whether preempting jobs would yield enough resources to run the top job. If both are true, the scheduler preempts running jobs and runs the top job.

If you take no action to configure how jobs should be prioritized, they are considered in submission order, one queue at a time. If you don’t prioritize queues, the queues are examined in an undefined order.

4.2.5.2 Overview of Prioritizing Jobs

PBS provides several tools (methods) for setting job execution priority. There are queue-based tools for organizing jobs, moving them around, and specifying the order in which groups of jobs should be examined. There are tools for sorting jobs into the order you want. There is a meta-tool (strict ordering) that allows you to specify that the top job must go next, regardless of whether the resources it requires are available now.
The scheduler can use only one sorting tool at a time. However, you can combine your chosen sorting tool with queue-based tools to give a wide variety of behaviors. Most of the queue-based tools can be used together. The scheduler can treat all jobs as if they are in a single queue, considering them all with respect to each other, or it can examine jobs queue by queue, comparing each job only to other jobs in the same queue.

You can change how execution priority is calculated, depending on which time slot is occurring. You can divide time up into primetime, non-prime-time, and dedicated time.

When the scheduler calculates job execution priority, it uses a built-in system of job classes. PBS runs special classes of jobs before it considers queue membership. These classes are for reservation, starving, suspended, and express jobs. Please see section 4.8.16 “Calculating Job Execution Priority” on page 187. After these jobs are run, the scheduler follows the rules you specify for queue behavior. Within each queue, jobs are sorted according to the sorting tool you choose.

4.2.5.3 Using Queue-based Tools to Prioritize Jobs

4.2.5.3.1 Using Queue Order to Affect Order of Consideration

When the scheduler examines waiting jobs, it can either consider all of the jobs in the complex as a whole, or it can examine jobs in only one queue at a time. When examining jobs one queue at a time, the scheduler can either run all the jobs it can from the first queue before examining the jobs in the next queue (the default behavior), or run one job from each queue before moving to the next queue. These two methods of examining queues individually are incompatible, and both are incompatible with treating all jobs as if they are in a single queue.

The by_queue scheduler parameter controls whether or not the scheduler runs all the jobs it can from the first queue before moving to the next. By default, this parameter is set to True. When the round_robin parameter is True, it overrides by_queue. See section 4.8.4 “Examining Jobs Queue by Queue” on page 151.

The round_robin scheduler parameter controls whether or not the scheduler examines one queue at a time, running one job from each queue before moving on to the next queue. This parameter overrides by_queue. See section 4.8.38 “Round Robin Queue Selection” on page 290.
If you want queues to be considered in a specific order, you must assign a different priority to each queue, and tell PBS to sort the queues according to their priority. To sort the queues according to their priority, set the `sort_queues` scheduler parameter to `True`. See section 4.8.44 “Sorting Queues into Priority Order” on page 306.

Give the queue you want considered first the highest priority, then the next queue the next highest priority, and so on. If the queues don’t have priority assigned to them, the order in which they are considered is undefined. To set a queue’s priority, use the `qmgr` command to assign a value to the `priority` queue attribute. See section 2.2.5.3 “Prioritizing Execution Queues” on page 26.

### 4.2.5.3.2 Using Express Queues in Job Priority Calculation

You can create express queues, and route jobs into them, if you want to give those jobs special priority.

An express queue is a queue whose priority is high enough to qualify as an express queue; the default for qualification is 150, but this can be set using the `preempt_queue_prio` scheduler parameter. For information on configuring express queues, see section 2.2.5.3.1 “Express Queues” on page 26.

When calculating execution priority, the PBS scheduler uses a built-in job class called “Express” which contains all jobs that have a preemption level greater than that of the `normal_jobs` level. By default, those jobs are jobs in express queues. See section 4.8.16 “Calculating Job Execution Priority” on page 187.

You can create preemption levels that include jobs in express queues. Jobs in higher preemption levels are allowed to preempt jobs in lower levels. See section 4.8.33 “Using Preemption” on page 258.

### 4.2.5.3.3 Routing Jobs into Queues

You can configure PBS to automatically put each job in the most appropriate queue. There are several approaches to this. See section 4.8.39 “Routing Jobs” on page 292.
4.2.5.3.4 Using Queue Priority when Computing Job Priority

You can configure the scheduler so that job priority is partly determined by the priority of the queue in which the job resides. See section 4.8.36 “Queue Priority” on page 282.

4.2.5.4 Using Job Sorting Tools to Prioritize Jobs

The scheduler can use only one sorting tool at a time. The sorting tool you choose can be used either on all jobs in the complex together, or within each queue, one queue at a time. The job sorting tools are the following:

- You can create a formula that the scheduler uses to sort jobs. The scheduler applies this formula to all jobs in the complex, using it to calculate a priority for each job. For example, you can specify in the formula that jobs requesting more CPUs have higher priority. If the formula is defined, it overrides fairshare and sorting jobs on keys. See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

- You can use the fairshare algorithm to sort jobs. This algorithm allows you to set a resource usage goal for users or groups. Jobs are prioritized according to each entity’s usage; jobs whose owners have used the smallest percentage of their allotment go first. For example, you can track how much CPU time is being used, and allot each group a percentage of the total. See section 4.8.18 “Using Fairshare” on page 191.

- You can sort jobs according to the same usage allotments you set up for fairshare. In this case, jobs whose owners are given the highest allotment go first. See section 4.8.14 “Sorting Jobs by Entity Shares (Was Strict Priority)” on page 180.

- You can sort jobs on one or more keys, for example, you can sort jobs first by the number of CPUs they request, then by the amount of memory they request. You can specify that either the high or the low end of the resulting sort has higher priority.

    You can create a custom resource, and use a hook to set a value for that resource for each job, and then sort on the resource.

    See section 4.8.42 “Sorting Jobs on a Key” on page 302.

- You can run jobs in the order in which they were submitted. See section
4.8.19 “FIFO Scheduling” on page 203.

- You can run jobs according to the priority requested for each job at submission time. This priority can be modified via a hook. See section 4.8.43 “Sorting Jobs by Requested Priority” on page 305 and “Hooks” on page 461.

### 4.2.5.5 Prioritizing Jobs by Wait Time

You can use the amount of time a job has been waiting to run in the priority calculation. There are two ways to measure wait time:

- Eligible waiting time: how long a job has been waiting to run due to a shortage of resources, rather than because its owner isn’t allowed to run jobs now. See section 4.8.13 “Eligible Wait Time for Jobs” on page 173

- Amount of time waiting in the queue

Both of these ways can be used when computing whether or not a job is starving. You can specify how long a job must be waiting to be considered starving. See section 4.8.45 “Starving Jobs” on page 307.

You can use a job’s eligible waiting time in the job sorting formula. See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

When a job is considered to be starving, it is automatically assigned special execution priority, and placed in the Starving execution priority class; see section 4.8.16 “Calculating Job Execution Priority” on page 187. You can configure preemption levels that include starving jobs; see section 4.8.33 “Using Preemption” on page 258.

### 4.2.5.6 Calculating Preemption Priority

Execution priority and preemption priority are two separate systems of priority.

By default, if the top job cannot run now, and it has high preemption priority, the scheduler will use preemption to run the top job. The scheduler will preempt jobs with lower preemption priority so that it can use the resources to run the top job. The default for jobs with high preemption priority is
jobs in express queues. You can configure many levels of preemption priority, specifying which levels can preempt which other levels. See section 4.8.33 “Using Preemption” on page 258.

4.2.5.7 Meta-priority: Running Jobs Exactly in Priority Order

By default, when scheduling jobs, PBS orders jobs according to execution priority, then considers each job, highest-priority first, and runs the next job that can run now. If a job cannot run now because the resources required are unavailable, the default behavior is to skip the job and move to the next in order of priority.

You can tell PBS to use a different behavior called strict ordering. This means that you tell PBS that it must not skip a job when choosing which job to run. If the top job cannot run, no job runs.

You can see that using strict ordering could lead to decreased throughput and idle resources. In order to prevent idle resources, you can tell PBS to run small filler jobs while it waits for the resources for the top job to become available. These small filler jobs do not change the start time of the top job. See section 4.8.46 “Using Strict Ordering” on page 310 and section 4.8.3 “Using Backfilling” on page 145.

4.2.5.8 Using Different Calculations for Different Time Periods

PBS allows you to divide time into two kinds, called primetime and non-primetime. All time is covered by one or the other of these two kinds of time. The times are arbitrary; you can set them up however you like. You can also choose not to define them, and instead to treat all time the same.

You can configure two separate, independent ways of calculating job priority for primetime and non-primetime. The same calculations are used during dedicated time; dedicated time is a time slot made up of primetime and/or non-primetime. Many scheduler parameters are prime options, meaning that they can be configured separately for primetime and non-primetime. For example, you can configure fairshare as your sorting tool during primetime, but sort jobs on a key during non-primetime.

If you use the formula, it is in force all of the time.
See section 4.8.34 “Using Primetime and Holidays” on page 274.

4.2.5.9 When Priority Isn’t Enough: Overrides

Sometimes, the tools available for setting job priority don’t do everything you need. For example, it may be necessary to run a job right away, regardless of what else is running. Or you may need to put a job on hold. Or you might need to tweak the way the formula works for the next \( N \) jobs. See section 4.8.30 “Overrides” on page 226.

4.2.5.10 Elements to Consider when Prioritizing Jobs

- Whether users, groups, or projects affect job priority: for techniques to use user, group, or project to affect job priority, see section 4.3.3 “Prioritizing Jobs by User, Project or Group” on page 101.

- Starving jobs: PBS has a built-in execution priority for starving jobs, but you can give starving jobs the highest execution priority by giving them the highest preemption priority and enabling preemption. See section 4.8.16 “Calculating Job Execution Priority” on page 187 and section 4.8.33 “Using Preemption” on page 258.

- Express jobs: PBS has a built-in execution priority for express jobs. You can set the preemption priority for express jobs; see section 4.8.33 “Using Preemption” on page 258.

- Suspended jobs: PBS has a built-in execution priority for suspended jobs. You can set the preemption priority for express jobs; see section 4.8.33 “Using Preemption” on page 258.

- Large or small jobs: you may want to give large and/or small jobs special treatment. See section 4.3.5 “Scheduling Jobs According to Size Etc.” on page 106.

- User’s priority request for job: the job submitter can specify a priority for the job at submission. You can sort jobs according to each job’s specified priority. See section 4.8.43 “Sorting Jobs by Requested Priority” on page 305.

- Whether the top job must be the next to run, regardless of whether it can run now; see section 4.8.46 “Using Strict Ordering” on page 310.
4.2.5.10.1 Queue-based Tools for Organizing Jobs

These tools can be used in conjunction with the sorting tools listed above.

- Queue-by-queue: PBS runs all the jobs it can from the first queue before moving to the next queue. Queue order is determined by priority and the value of the `sort_queues` parameter. See section 4.8.4 “Examining Jobs Queue by Queue” on page 151.

- Round-robin job selection: PBS can select jobs from queues in a round-robin fashion. See section 4.8.38 “Round Robin Queue Selection” on page 290.

- Queue priority: Queues can be ordered according to their priority; jobs in higher-priority queues can then be examined before those in lower-priority queues. See section 2.2.5.3 “Prioritizing Execution Queues” on page 26.

- Sorting queues: PBS can sort queues into priority order. See section 4.8.44 “Sorting Queues into Priority Order” on page 306.

- Express queues: Jobs in express queues are assigned increased priority. See section 2.2.5.3.1 “Express Queues” on page 26, and section 4.2.5.3.2 “Using Express Queues in Job Priority Calculation” on page 78.

- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See section 4.8.39 “Routing Jobs” on page 292.

4.2.5.10.2 Job Sorting Tools

You can use only one job sorting tool at a time, and the tool you choose applies to all queues. However, you could use a different sorting tool for primetime and non-primetime. Each of these methods can be used on all jobs in the complex, or one queue at a time.

- Job sorting formula: You create a formula that PBS uses to calculate each job’s priority. See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.


- Sorting jobs on keys: PBS can sort jobs according to one or more keys,
such as requested CPUs or memory; see section 4.8.42 “Sorting Jobs on a Key” on page 302.

- Entity shares (strict priority): Jobs are prioritized according to the owner’s fairshare allocation. See section 4.8.14 “Sorting Jobs by Entity Shares (Was Strict Priority)” on page 180.

- FIFO: Jobs can be run in submission order. See section 4.8.19 “FIFO Scheduling” on page 203.

- Job’s requested priority: you can sort jobs on the priority requested for the job; see section 4.8.43 “Sorting Jobs by Requested Priority” on page 305.

4.2.5.10.3 Other Job Prioritization Tools

- Strict ordering: you can specify that jobs must be run in priority order, so that a job that cannot run because resources are unavailable is not skipped. See section 4.8.46 “Using Strict Ordering” on page 310.

- Waiting time: PBS can assign increased priority to jobs that have been waiting to run. See section 4.8.13 “Eligible Wait Time for Jobs” on page 173, and section 4.8.45 “Starving Jobs” on page 307.

- Setting job execution priority: PBS can set job execution priority according to a set of rules. See section 4.8.16 “Calculating Job Execution Priority” on page 187.

- Preemption: PBS preempts lower-priority jobs in order to run higher-priority jobs. See section 4.8.33 “Using Preemption” on page 258.

- Starving jobs: Jobs that have been waiting for a specified amount of time can be given increased priority. See section 4.8.45 “Starving Jobs” on page 307.

- Behavior overrides: you can intervene manually in how jobs are run. See section 4.8.30 “Overrides” on page 226.
4.2.6 Resource Allocation to Users, Projects & Groups

If you need to ensure fairness, you may need to make sure that resources are allocated fairly. If different users, groups, or projects own or pay for different amounts of hardware or machine time, you may need to allocate resources according to these amounts or proportions.

You can allocate hardware-based resources such as CPUs or memory, and/or time-based resources such as walltime or CPU time, according to to the agreed amounts or proportions. You can also control who starts jobs.

4.2.6.1 Limiting Amount of Resources Used

4.2.6.1.1 Allocation Using Resource Limits

You can use resource limits as a way to enforce agreed allocation amounts. This is probably the most straightforward way, and the easiest to explain to your users. PBS provides a system for limiting the total amount of each resource used by users and groups at the server and at each queue. For example, you can set a limit on the number of CPUs that any generic user can use at one time at QueueA, but set three different individual limits for each of three users that have special requirements, at the same queue. See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.

4.2.6.1.2 Allocation Using Fairshare

The PBS fairshare tool allows you to start jobs according to how much of a particular resource has been used by job owners. You can designate who the valid job owners are, which resource is being tracked, and how much of the resource each owner is allowed to be using. Fairshare uses a moving average of resource usage, so that a user who in the recent past has not used their share can use more now. For example, you can track usage of the cput resource, and give one group 40 percent of usage, one 50 percent, and one group, 10 percent. See section 4.8.18 “Using Fairshare” on page 191.
4.2.6.1.3 Allocation Using Routing

If you do not want to place usage limits directly on users or groups, you can instead route their jobs to specific queues, where those queues have their own resource usage limits.

To route jobs this way, force users to submit jobs to a routing queue, and set access control limits at each execution queue. See section 9.3 “Using Access Control” on page 659. Make the routing queue be the default queue:

```
Qmgr: set server default_queue = <routing queue name>
```

Using this method, you place a limit for total resource usage at each queue, for each resource you care about. See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.

4.2.6.2 Limiting Jobs

4.2.6.2.1 Limiting Number of Jobs per User or Group

You can set limits on the numbers of jobs that can be run by users and groups. You can set these limits for each user and group, and you can set them at the server and at each queue. You can set a generic limit for all users or groups, and individual limits that override the generic limit. For example, you can set a limit that says that no user at the complex can run more than 8 jobs. Then you can set a more specific limit for QueueA, so that users at QueueA can run 4 jobs. Then you can set a limit for User1 and User2 at QueueA, so that they can run 6 jobs. See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.

4.2.6.2.2 Allocation Using Round-robin Queue Selection

PBS can select jobs from queues by examining the queues in round-robin fashion. When using the round-robin method, the scheduler considers the first queue, tries to run a job from that queue, then considers the next queue, tries to run a job from that queue, then considers the next queue, and so on, in a circular fashion. If you want a simple way to control how jobs are started, you can use round-robin where each queue belongs to a different user or group. See section 4.8.38 “Round Robin Queue Selection” on page 290.
4.2.6.2.3 Limiting Resource Usage per Job

If you are having trouble with large jobs taking up too much of a resource, you can limit the amount of the resource being used by individual jobs. You can set these limits for each queue, and at the server. See section 5.15.3 “Placing Resource Limits on Jobs” on page 436.

4.2.6.3 Resource Allocation Tools

The following is a list of scheduling tools that you can use for allocating resources or limiting resources or jobs:

• Matching: PBS places jobs where the available resources match the job’s resource requirements; see section 4.8.28 “Matching Jobs to Resources” on page 222.

• Reservations: Users can create advance and standing reservations for specific resources for specific time periods. See section 4.8.37 “Advance and Standing Reservations” on page 283.


• Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See section 2.2.6 “Routing Queues” on page 26 and section 4.8.39 “Routing Jobs” on page 292.

• Limits on resource usage by users and groups: You can set limits on user and group resource usage. See section 4.8.25 “Limits on User and Group Resource Usage” on page 216.

• Round-robin job selection: PBS can select jobs from queues in a round-robin fashion. See section 4.8.38 “Round Robin Queue Selection” on page 290.

• Sorting queues: PBS can sort queues into priority order. See section 4.8.44 “Sorting Queues into Priority Order” on page 306.

• Limits on number of jobs for users and groups: You can set limits on the numbers of jobs that can be run by users and groups. See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.

• Limits on resources used by each job: You can set limits on the amount of each resource that any job can use. See section 4.8.23 “Limits on
Chapter 4  Scheduling

Per-job Resource Usage” on page 215.

- Limit on number of jobs at each vnode: You can set limits on the number of jobs that can run at each vnode. See section 4.8.26 “Limits on Jobs at Vnodes” on page 216.

- Using custom resources to limit resource usage: You use custom resources to manage usage. See section 4.8.8 “Using Custom and Default Resources” on page 155.

- Gating and admission requirements: You can specify admission requirements for jobs. See section 4.8.21 “Gating Jobs at Server or Queue” on page 213.

- Making jobs inherit default resources: You can use default resources to manage jobs. See section 4.8.8 “Using Custom and Default Resources” on page 155.

4.2.7  Time Slot Allocation

Time slot allocation is the process of creating time slots within which only specified jobs are allowed to run.

4.2.7.1  Why Allocate Time Slots

You may want to set up blocks of time during which only certain jobs are allowed to run. For example, you might need to ensure that specific high-priority jobs have their own time slot, so that they are guaranteed to be able to run and finish before their results are required.

You may want to divide jobs into those that run at night, when no one is around, and those that run during the day, because their owners need the results then.

You might want to run jobs on desktop clusters only at night, when the primary users of the desktops are away.

When you upgrade PBS, a chunk of dedicated time can come in very handy. You set up dedicated time for a time period that is long enough for you to perform the upgrade, and you make sure the time slot starts far enough out that no jobs will be running.
You may want to run different scheduling policies at different times or on different days.

### 4.2.7.2 How to Allocate Time Slots

Time slots are controlled by queues: primetime queues, non-primetime queues, dedicated time queues, and reservation queues. Therefore, you can use your favorite routing method to move jobs into the desired queues. See section 4.8.39 “Routing Jobs” on page 292.

#### 4.2.7.2.1 Allocation Using Primetime and Holidays

You can specify how to divide up days or weeks, and designate each time period to be either primetime or non-primetime. You can use this division in the following ways:

- You can run a different policy during primetime from that of non-primetime
- You can run specific jobs during primetime, and others during non-primetime

See section 4.8.34 “Using Primetime and Holidays” on page 274.

#### 4.2.7.2.2 Allocation Using Dedicated Time

Dedicated time is a time period where the only jobs that are allowed to run are the ones in dedicated time queues. The policy you use during dedicated time is controlled by the normal primetime and non-primetime policies; those times overlap dedicated time.

If you don’t allow any jobs into a dedicated time queue, you can use it to perform an upgrade.

See section 4.8.10 “Dedicated Time” on page 171.

#### 4.2.7.2.3 Allocation Using Reservations

You and any other PBS user can create advance and standing reservations. These are time periods with a defined start and end, for a specific, defined set of resources. Reservations are used to make sure that specific jobs can run on time. See section 4.8.37 “Advance and Standing Reservations” on page 283.
4.2.7.2.4 Allocation Using `cron` Jobs or the Windows Task Scheduler

You can use `cron` or the Windows Task Scheduler to run jobs at specific times. See section 4.8.7 “`cron` Jobs, or the Windows Task Scheduler” on page 155.

4.2.7.3 Time Slot Allocation Tools

The following is a list of scheduling tools that you can use to create time slots:

- Primetime and holidays: You can specify days and times that are to be treated as prime execution time. See section 4.8.34 “Using Primetime and Holidays” on page 274.
- Dedicated time: You can set aside blocks of time reserved for certain operations. See section 4.8.30.6 “Using Dedicated Time” on page 229.
- `cron` jobs and the Windows Task Scheduler: You can use `cron` or the Windows Task Scheduler to run jobs. See section 4.8.30.7 “Using `cron` Jobs or the Windows Task Scheduler” on page 229.
- Reservations: Users can create advance and standing reservations for specific resources for specific time periods. See section 4.8.37 “Advance and Standing Reservations” on page 283.

4.2.8 Job Placement Optimization

PBS automatically places jobs where they can run, but you can refine how jobs are placed.

Optimizations are the techniques you use to increase throughput, turn-around, or efficiency, by taking advantage of where jobs can be run.

PBS places jobs according to placement optimization settings in tools to specify how vnodes should be organized, how jobs should be distributed, and how resources should be used.
4.2.8.1 Why Optimize Placement

PBS automatically places jobs where they can run, matching jobs to resources, so why optimize placement?

• You can help PBS refine its understanding of hardware topology, so that PBS can place jobs where they will run most efficiently.
• If you have some vnodes that are faster than others, you can preferentially place jobs on those vnodes.
• You may need to place jobs according to machine ownership, so that for example only jobs owned by a specific group run on a particular machine.
• You can take advantage of the presence of an HPCBP cluster, and run jobs on it.
• You can take advantage of unused workstation computing capacity.
• You can balance the workload between two or more PBS complexes, trading jobs around depending on the workload on each complex.
• You can specify whether or not certain vnodes should be used for more than one job at a time.
• You can tell PBS to avoid placing jobs on highly-loaded vnodes

4.2.8.2 Matching Jobs to Resources

By default, PBS places jobs where the available resources match the job’s resource requirements. See section 4.8.28 “Matching Jobs to Resources” on page 222.

4.2.8.3 Organizing and Selecting Vnodes

By default, the order in which PBS examines vnodes is undefined. The default setting for vnode sorting is the following:

    node_sort_key: “sort_priority HIGH all”

However, sort_priority means sort on each vnode’s priority attribute, but by default, that attribute is unset.

By default, PBS does not organize vnodes into groups.
By default, when PBS chooses vnodes for a job, it runs down its list of vnodes, searching until it finds vnodes that can supply the job with the requested resources. You can improve this in two ways:

- PBS provides a way to organize your vnodes so that jobs can run on groups of vnodes, where the selected group of vnodes provides the job with good connectivity. This can improve memory access and interprocess communication speed. PBS then searches through these groups of vnodes, called placement sets, looking for the smallest group that satisfies the job’s requirements. Each placement set is a group of vnodes that share a value for a resource. An illustrative example is a group of vnodes that are all connected to the same high speed switch, so that all of the vnodes have the same value for the switch resource. For detailed information on how placement sets work and how to configure them, see section 4.8.32 “Placement Sets” on page 237.

- By default, the order in which PBS examines vnodes, whether in or outside of placement sets, is undefined. PBS can sort vnodes on one or more keys. Using this tool, you can specify which vnodes should be selected first. For information on sorting vnodes on keys, see section 4.8.47 “Sorting Vnodes on a Key” on page 312.

You can sort vnodes in conjunction with placement sets.

4.2.8.4 Distributing Jobs

All of the following methods for distributing jobs can be used together.

4.2.8.4.1 Filtering Jobs to Specific Vnodes

If you want to run certain kinds of jobs on specific vnodes, you can route those jobs to specific execution queues, and tie those queues to the vnodes you want. For example, if you want to route jobs requesting large amounts of memory to your large-memory machines, you can set up an execution queue called LMemQ, and associate that queue with the large-memory vnodes. You can route any kind of job to its own special execution queue. For example, you can route jobs owned by the group that owns a cluster to a special queue which is associated with the cluster. For details on routing jobs, see section 4.8.39 “Routing Jobs” on page 292. For details on associating vnodes and queues, see section 4.8.2 “Associating Vnodes with Queues” on page 141.
4.2.8.4.2 Running Jobs at Least-loaded Complex

You can set up cooperating PBS complexes that automatically run jobs from each other’s queues. This allows you to dynamically balance the workload across multiple, separate PBS complexes. See section 4.8.31 “Peer Scheduling” on page 229.

4.2.8.4.3 Running Jobs on an HPCBP Cluster

You can set PBS up so that it can communicate with one or more HPCBP clusters, sending PBS jobs to an HPCBP cluster to be run. See Chapter 7, "Metascheduling Using HPC Basic Profile", on page 579.

4.2.8.4.4 Using Idle Workstations

You can run jobs on workstations whenever they are not being used by their owners. PBS can monitor workstations for user activity or load, and run jobs when those jobs won’t interfere with the user’s operation. See section 4.8.9 “Using Idle Workstation Cycle Harvesting” on page 159.

4.2.8.4.5 Avoiding Highly-loaded Vnodes

You can tell PBS not to run jobs on vnodes that are above a specified load. This is in addition to the default behavior, where PBS does not run jobs that request more of a resource than it thinks each vnode can supply. See section 4.8.27 “Using Load Balancing” on page 216.

4.2.8.4.6 Placing Job Chunks on Desired Hosts

You can tell PBS to place each job on as few hosts as possible, to place each chunk of a job on a separate host, a separate vnode, or on any vnode. You can specify this behavior for the jobs at a queue and at the server.

You can do the following

- Set default behavior for the queue or server: jobs inherit behavior if they do not request it; see section 5.9.3.6 “Specifying Default Job Placement” on page 341
- Use a hook to set each job’s placement request (Resource_List.place). See Chapter 6, "Hooks", on page 461
For more on placing chunks, see section 4.8.6 “Organizing Job Chunks” on page 154.

For information on how jobs request placement, see section 2.56.2.5 “Requesting Resources and Placing Jobs” on page 248.

4.2.8.5 Shared or Exclusive Resources and Vnodes

PBS can give jobs their own vnodes, or fill vnodes with as many jobs as possible. The scheduler uses a set of rules to determine whether a job can share resources or a host with another job. These rules specify how the vnode sharing attribute should be combined with a job’s placement directive. The vnode’s sharing attribute supersedes the job’s placement request.

You can set each vnode’s sharing attribute so that the vnode or host is always shared, always exclusive, or so that it honors the job’s placement request. See section 4.8.40 “Shared vs. Exclusive Use of Resources by Jobs” on page 298.

4.2.8.6 Tools for Organizing Vnodes

- Placement sets: PBS creates sets of vnodes organized by the values of multiple resources. See section 4.8.32 “Placement Sets” on page 237.
- Sorting vnodes on keys: PBS can sort vnodes according to specified keys. See section 4.8.47 “Sorting Vnodes on a Key” on page 312.

4.2.8.7 Tools for Distributing Jobs

- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See section 2.2.6 “Routing Queues” on page 26 and section 4.8.39 “Routing Jobs” on page 292.
- Associating vnodes with queues: You can specify that jobs in a given queue can run only on specific vnodes, and vice versa. See section 4.8.2 “Associating Vnodes with Queues” on page 141.
- Using High Performance Computing Basic Profile: PBS can send jobs off to an HPCBP cluster for execution. See Chapter 7, "Metascheduling Using HPC Basic Profile", on page 579.
- Idle workstation cycle harvesting: PBS can take advantage of unused workstation CPU time. See section 4.8.9 “Using Idle Workstation
Scheduling

Chapter 4

Cycle Harvesting” on page 159.

- Peer scheduling: PBS complexes can exchange jobs. See section 4.8.31 “Peer Scheduling” on page 229.
- Load balancing: PBS can place jobs so that machines have balanced loads. See section 4.8.27 “Using Load Balancing” on page 216.
- SMP cluster distribution: PBS can place jobs in a cluster as you specify. See section 4.8.41 “SMP Cluster Distribution” on page 300.

4.2.9 Resource Efficiency Optimizations

PBS automatically runs each job where the resources required for the job are available. You can refine the choices PBS makes.

Resource optimizations are the techniques you use to increase throughput, turnaround, or efficiency, by taking advantage of how resources are used.

Before reading this section, please make sure you understand how resources are used by reading section 4.8.28 “Matching Jobs to Resources” on page 222.

4.2.9.1 Why Optimize Use of Resources

You may want to take advantage of the following:

- If you are using strict ordering, you can prevent resources from standing idle while the top job waits for its resources to become available
- PBS can estimate the start times of jobs, so that users can stay informed
- PBS can provision vnodes with the environments that jobs require
- PBS can track resources that are outside of the control of PBS, such as scratch space
- You can take advantage of unused workstation computing capacity.
- You can balance the workload between two or more PBS complexes, trading jobs around depending on the workload on each complex.
- You can specify whether or not certain vnodes should be used for more than one job at a time.
- Users can specify that jobs that are dependent on the output of other
jobs run only after the other jobs complete

- You can tell PBS to avoid placing jobs on highly-loaded vnodes

### 4.2.9.2 How to Optimize Resource Use

#### 4.2.9.2.1 Backfilling Around Top Jobs

PBS creates a list of jobs ordered by priority, and tries to run the jobs in order of priority. You can force all jobs to be run in exact order of their priority, using strict ordering. See section 4.8.46 “Using Strict Ordering” on page 310. However, this can reduce resource utilization when the top job cannot run now and must wait for resources to become available. You can offset this problem by using backfilling, where PBS tries to fit smaller jobs in around the top job that cannot run. The start time of the top job is not delayed. Job walltimes are required in order to use backfilling. You can specify the number of jobs around which to backfill. You can also disable this feature. See section 4.8.3 “Using Backfilling” on page 145.

If you do not use strict ordering, PBS won’t necessarily run jobs in exact priority order. PBS will instead run jobs so that utilization is maximized, while trying to preserve priority order.

#### 4.2.9.2.2 Using Dependencies

Job submitters can specify the dependencies between jobs. For example, if you have an end-of-day accounting job that must run after certain other jobs, you can specify that. See section 4.8.11 “Dependencies” on page 173.

#### 4.2.9.2.3 Estimating Start Time for Jobs

You can tell PBS to estimate start times and execution vnodes for either the number of jobs being backfilled around, or all jobs. Users can then see when their jobs are estimated to start, and the vnodes on which they are predicted to run. See section 4.8.15 “Estimating Job Start Time” on page 181.
4.2.9.2.4 Provisioning Vnodes with Required Environments

PBS can provision vnodes with environments (applications or operating systems) that jobs require. This means that a job can request a particular environment that is not yet on a vnode, but is available to be instantiated there. See section 4.8.35 “Provisioning” on page 281.

4.2.9.2.5 Tracking Dynamic Resources

You can use dynamic PBS resources to represent elements that are outside of the control of PBS, typically for licenses and scratch space. You can represent elements that are available to the entire PBS complex as server-level resources, or elements that are available at a specific host or hosts as host-level resources. For an example of configuring a server-level dynamic resource, see section 5.14.4.1.1 “Example of Configuring Dynamic Server-level Resource” on page 371. For an example of configuring a dynamic host-level resource, see section 5.14.5.1.1 “Example of Configuring Dynamic Host-level Resource” on page 376.

For a complete description of how to create and use dynamic resources, see section 5.14 “Custom Resources” on page 352.

4.2.9.3 Optimizing Resource Use by Job Placement

4.2.9.3.1 Sending Jobs to Complex Having Lightest Workload

You can set up cooperating PBS complexes that automatically run jobs from each other’s queues. This allows you to dynamically balance the workload across multiple, separate PBS complexes. See section 4.8.31 “Peer Scheduling” on page 229.

4.2.9.3.2 Running Jobs on an HPCBP Cluster

You can set PBS up so that it can communicate with one or more HPCBP clusters, sending PBS jobs to an HPCBP cluster to be run. See Chapter 7, "Metascheduling Using HPC Basic Profile", on page 579.
4.2.9.3.3 Using Idle Workstations

You can run jobs on workstations whenever they are not being used by their owners. PBS can monitor workstations for user activity or load, and run jobs when those jobs won’t interfere with the user’s operation. See section 4.8.9 “Using Idle Workstation Cycle Harvesting” on page 159.

4.2.9.3.4 Avoiding Highly-loaded Vnodes

You can tell PBS not to run jobs on vnodes that are above a specified load. This is in addition to the default behavior, where PBS does not run jobs that request more of a resource than it thinks each vnode can supply. See section 4.8.27 “Using Load Balancing” on page 216.

4.2.9.4 Resource Efficiency Optimization Tools

The following is a list of scheduling tools that you can use to optimize how resources are used:

- Backfilling around most important job(s): PBS can place small jobs in otherwise-unused blocks of resources. See section 4.8.3 “Using Backfilling” on page 145.
- Dependencies: Users can specify requirements that must be met by previous jobs in order for a given job to run. See section 4.8.11 “Dependencies” on page 173.
- Estimating start time of jobs: PBS can estimate when jobs will start, so that users can be informed. See section 4.8.15 “Estimating Job Start Time” on page 181.
- Provisioning vnodes with required environments: PBS can provision vnodes with the environments that jobs require. See section 4.8.35 “Provisioning” on page 281.
- Using dynamic resources: PBS can track resources such as scratch space and licenses. See section 4.8.12 “Dynamic Resources” on page 173.
- Idle workstation cycle harvesting: PBS can take advantage of unused workstation CPU time. See section 4.8.9 “Using Idle Workstation Cycle Harvesting” on page 159.
- Peer scheduling: PBS complexes can exchange jobs. See section 4.8.31
“Peer Scheduling” on page 229.

- Load balancing: PBS can place jobs so that machines have balanced loads. See section 4.8.27 “Using Load Balancing” on page 216.

### 4.2.10 Overrides

Overrides are the techniques you use to override the specified scheduling behavior of PBS.

#### 4.2.10.1 Why and How to Override Scheduling

- If you need to run a job immediately, you can tell PBS to run a job now. You can optionally specify the vnodes and resources to run it. See section 4.8.30.1 “Run a Job Manually” on page 226.
- If you need to prevent a job from running, you can tell PBS to place a hold on a job. See section 4.8.30.2 “Hold a Job Manually” on page 227.
- If you need to change how the formula computes job priority, you can make on-the-fly changes to how the formula is computed. See section 4.8.30.5 “Change Formula On the Fly” on page 229.
- If you need a block of time where you can control what’s running, for example for upgrading PBS, you can create dedicated time. See section 4.8.30.6 “Using Dedicated Time” on page 229.
- If you need to submit jobs at a certain time, you can use `cron` or the Windows Task Scheduler to run jobs. See section 4.8.30.7 “Using cron Jobs or the Windows Task Scheduler” on page 229.
- If you need to change job resource requests or attributes, you can use hooks to examine jobs and alter their characteristics. See “Hooks” on page 461.
4.3 Choosing a Policy

4.3.1 Overview of Kinds of Policies

You can tune PBS to produce a wide variation in scheduling behavior. You can choose from a wide variety of behaviors for each sub-goal, resulting in many possible scheduling policies. However, policies can be grouped into the following kinds:

- FIFO, where you essentially run jobs in the order in which they were submitted; see section 4.3.2 “FIFO: Submission Order” on page 100
- According to user or group priority, where the job’s priority is determined by the owner’s priority; see section 4.3.3 “Prioritizing Jobs by User, Project or Group” on page 101
- According to resource allocation rules, where jobs are run so that they use resources following a set of rules for how resources should be awarded to users or groups; see section 4.3.4 “Allocating Resources by User, Project or Group” on page 102
- According to the size of the job, for example measured by CPU or memory request; see section 4.3.5 “Scheduling Jobs According to Size Etc.” on page 106
- By setting up time slots for specific uses; see section 4.3.6 “Scheduling Jobs into Time Slots” on page 109

4.3.2 FIFO: Submission Order

If you want jobs to run in the order in which they are submitted, use FIFO. You can use FIFO across the entire complex, or within each queue.

If it’s important that jobs run exactly in submission order, use FIFO with strict ordering. However, if you don’t want resources to be idle while the top job is stuck, you can use FIFO with strict ordering and backfilling.

To run jobs in submission order across the entire complex, see section 4.8.19.1 “Configuring Basic FIFO Scheduling” on page 203 and section 4.8.19.2 “FIFO for Entire Complex” on page 203.
To run jobs in submission order, selecting queues in order of their priority, see section 4.8.19.1 “Configuring Basic FIFO Scheduling” on page 203 and section 4.8.19.3 “Queue by Queue FIFO” on page 203.

To run jobs in submission order, with strict ordering, see section 4.8.19.4 “FIFO with Strict Ordering” on page 204.

To run jobs in submission order, with strict ordering and backfilling, see section 4.8.19.5 “FIFO with Strict Ordering and Backfilling” on page 205.

### 4.3.3 Prioritizing Jobs by User, Project or Group

If you need to run jobs from some users, groups, or projects before others, you can prioritize the jobs this way.

You can prioritize jobs by user, project, or group, using the following techniques:

- Routing each entity’s jobs to its own execution queue, assigning the queue the desired priority, sorting the queues by priority, and examining jobs queue by queue. See the following:
  - For routing: section 2.2.6 “Routing Queues” on page 26
  - For setting queue priority: section 2.2.5.3 “Prioritizing Execution Queues” on page 26
  - For sorting queues: section 4.8.44 “Sorting Queues into Priority Order” on page 306
  - For examining jobs queue by queue: section 4.8.4 “Examining Jobs Queue by Queue” on page 151

- Routing each entity’s jobs to its own execution queue, where the jobs inherit a custom resource that you use in the job sorting formula. See the following:
  - For routing: section 2.2.6 “Routing Queues” on page 26
  - For inherited resources: section 12.2 “Allocating Resources to Jobs” on page 842
  - For the job sorting formula: section 4.8.20 “Using a Formula for
Computing Job Execution Priority” on page 205

• Using a hook to allocate a custom resource to each job, where the hook sets the value according to the priority of the job’s owner, then using the resource in the job sorting formula. See the following:
  - For hooks: “Hooks” on page 461
  - For custom resources: “Custom Resources” on page 352
  - For the job sorting formula: section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205

• Assigning a greater fairshare allocation in the fairshare tree to the users or groups whose jobs must run first, and running jobs according to entity shares. See the following:
  - For fairshare: section 4.8.18 “Using Fairshare” on page 191
  - For entity shares: section 4.8.14 “Sorting Jobs by Entity Shares (Was Strict Priority)” on page 180

4.3.4 Allocating Resources by User, Project or Group

When you want to divide up hardware usage among users, groups, or projects, you can make sure you allocate resources along those lines. You can do this in the following ways:

• Allocate portions of the entire complex to each entity; see section 4.3.4.1 “ Allocating Portions of Complex” on page 103

• Allocate portions of all machines or clusters to each entity, or use controlled allocation for some hardware, with a free-for-all elsewhere; see section 4.3.4.2 “Allocating Portions of Machines or Clusters” on page 103

• Lock entities into using specific hardware; see section 4.3.4.3 “Locking Entities into Specific Hardware” on page 105
4.3.4.1 Allocating Portions of Complex

4.3.4.1.1 Allocating Specific Amounts

To allocate specific amounts of resources across the entire complex, you can use resource limits at the server. These limits set the maximum amount that can be used, ensuring that users or groups stay within their bounds. You can set a limit for each resource, and make it different for each user and/or group. You can set a different limit for each user and group, for each resource.

For example, you can set a limit of 48 CPUs in use at once by most groups, but give groupA a limit of 96 CPUs. You can give each individual user a limit of 8 CPUs, but give UserA a limit of 10 CPUs, and UserB a limit of 4 CPUs.

To set limits for usage across the entire complex, set the limits at the server. See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.

4.3.4.1.2 Allocating Percentages

To allocate a percentage of the resources being used at the complex, you can use fairshare. Fairshare tracks a moving average of resource usage, so it takes past use into account. You choose which resource to track. You can tune the influence of past usage. To use fairshare, you write a file that contains the percentage of resource usage for each entity.

To use fairshare across the entire complex, make sure that both by_queue and round_robin are False.

Fairshare is described in section 4.8.18 “Using Fairshare” on page 191.

4.3.4.2 Allocating Portions of Machines or Clusters

You can allocate fixed amounts of a machine or groups of machines. You can do this for as many machines as you want. For example, on HostA, you can give GroupA 100 CPUs, GroupB 150 CPUs, and GroupC 50 CPUs, while at HostB, GroupA gets 10, GroupB gets 8, and GroupC gets 25.
To allocate fixed portions of a specific machine or group of machines, you use these tools in combination:

- Create an execution queue for this machine; see section 2.2.3 “Creating Queues” on page 22.
- Route jobs belonging to the users or groups who share this machine into a queue. Each machine or cluster that requires controls gets its own queue. See section 4.8.39 “Routing Jobs” on page 292.
- Associate the queue with the vnodes in question; see section 4.8.2 “Associating Vnodes with Queues” on page 141.
- Set a limit at the queue for each resource that you care about, for each user or group. These limits control use of the vnodes associated with the queue only. See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.

You can prevent unauthorized usage by setting generic user and group limits for the machine’s queue to zero. However, you probably don’t want users to submit their jobs to a queue where they are not allowed to run, only to have those jobs languish. You can avoid this by doing the following:

- Setting up a routing queue; see section 2.2.6 “Routing Queues” on page 26.
- Make the routing queue be the default queue:
  
  `Qmgr: set server default_queue = <routing queue name>`

- Making the routing queue the only queue that accepts job submission: set `from_route_only` to `True` on execution queues tied to hardware. See section 2.2.5.1 “Where Execution Queues Get Their Jobs” on page 23.
- Using queue access control to limit which jobs are routed into the execution queue; see section 2.2.6.5 “Using Access Control to Route Jobs” on page 33.

You can either set up allocations for every machine, or you can set up allocations for only some machines, leaving a free-for-all for the others. If you want access to be unrestricted for some machines, do not set limits at the server.
4.3.4.3 Locking Entities into Specific Hardware

You can send all jobs from some users or groups to designated hardware; you are essentially limiting them to a sandbox. To do this, do the following:

• Create an execution queue for the sandbox hardware; see section 2.2.3 “Creating Queues” on page 22.

• Create at least one other execution queue; see section 2.2.3 “Creating Queues” on page 22.

• Create a routing queue; see section 2.2.3 “Creating Queues” on page 22.

• Make the routing queue be the default queue:

\[ \text{Qmgr: set server default_queue = <routing queue name>} \]

• Force all users to submit jobs to the routing queue: set from_route_only to True on all other queues. See section 2.2.5.1 “Where Execution Queues Get Their Jobs” on page 23.

• Use queue access control: allow jobs from the controlled users or groups into the sandbox queue only. See section 2.2.6.5 “Using Access Control to Route Jobs” on page 33.

• Associate the sandbox queue with the sandbox vnodes. See section 4.8.2 “Associating Vnodes with Queues” on page 141.

Note that you can either allow all users into the sandbox queue, or allow only the controlled users into the sandbox queue. Furthermore, you can either associate only the sandbox queue with the sandbox vnodes, or you an associate both execution queues with the sandbox vnodes. For the latter, you associate the non-sandbox queue with all vnodes, using a different custom resource from the one you use to associate the sandbox queue with the sandbox vnodes.
4.3.5  Scheduling Jobs According to Size Etc.

You may need to treat jobs differently depending on their size or other characteristics. For example, you might want to run jobs differently depending on the number of CPUs or amount of memory requested by the job, or whether the job requests GPUs.

- Give special priority to a group of jobs
- Run a group of jobs on designated hardware
- Run a group of jobs in designated time slots: reservations, dedicated time, and primetime or non-primetime

There are two main approaches to doing this. You can route jobs into queues, or you can use hooks to set values. Here is an outline:

- Route certain kinds of jobs into their own queues, in order to treat each kind differently. This works for priority, hardware, and time slots. See section 4.3.5.1 “Special Treatment via Routing” on page 106
  - Route each kind to its own queue, using queue-based routing or a submission hook;
  - Use queue-based methods to set job priority or to run the jobs on certain hardware or in certain time slots

- Use hooks to set priority for jobs or to set a custom resource that will send jobs to certain hardware. This does not work for time slots. See section 4.3.5.2 “Special Treatment via Hooks” on page 108.
  - Use a submission hook to set each job’s Priority attribute, or set a value for a custom resource used in the job sorting formula
  - Use a submission hook to set a custom host-level resource value for each job, where the value matches the value at the desired hardware

4.3.5.1  Special Treatment via Routing

Use a routing queue or a hook to route jobs into a special queue, where the jobs are given special priority, or are run on special hardware, or are run in special time slots.
4.3.5.1.1 Routing via Queues

• Create your destination queues. See section 2.2.3 “Creating Queues” on page 22.

• Set limits at the destination queues, so that each queue receives the correct jobs. See section 2.2.6.4 “Using Resources to Route Jobs Between Queues” on page 28.

• Create a routing queue, and set its destination queues. See section 2.2.6 “Routing Queues” on page 26.

• Make the routing queue be the default queue:

    `Qmgr: set server default_queue = <routing queue name>`

4.3.5.1.2 Using Hooks to Route Jobs

You can use a submission hook to move jobs into the queues you want. See section 4.8.39.2.2 “Hooks as Mechanism to Move Jobs” on page 295.

4.3.5.1.3 Giving Routed Jobs Special Priority

You can give routed jobs special priority in the following ways:

• Have the jobs inherit a custom resource from the special queue, and use this resource in the job sorting formula.
  - For how to have jobs inherit custom resources, see section 12.2 “Allocating Resources to Jobs” on page 842.
  - For how to use the job sorting formula, see section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

• Give the queue itself special priority, and use queue priority in the job sorting formula.
  - For how to assign priority to queues, see section 2.2.5.3 “Prioritizing Execution Queues” on page 26
  - For how to use the job sorting formula, see section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.
4.3.5.1.4 Running Jobs on Special Vnodes

Now that the special jobs are routed to a special queue, associate that queue with the special vnodes. See section 4.8.2 “Associating Vnodes with Queues” on page 141.

4.3.5.1.5 Running Jobs in Special Time Slots

If you want to run jobs during dedicated time, route the jobs into one or more dedicated time queues. In the same way, for primetime or non-primetime, route jobs into primetime or non-primetime queues. You can also route jobs into reservation queues for reservations that you have created for this purpose.

For using dedicated time, see section 4.8.10 “Dedicated Time” on page 171

For using primetime and non-primetime, see section 4.8.34 “Using Prime-time and Holidays” on page 274

For using reservations, see section 4.8.37 “Advance and Standing Reservations” on page 283

4.3.5.2 Special Treatment via Hooks

4.3.5.2.1 Setting Job Priority Via Hook

Use a job submission hook to set the job priority, by doing one of the following:

- Set a custom numeric resource for the job, and use the resource in the job sorting formula
  - For how to use hooks, see Chapter 6, "Hooks", on page 461
  - For how to use the job sorting formula, see section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

- Set the job’s Priority attribute, and sort jobs on a key, where the key is the job’s Priority attribute.
  - For how to set job attributes, see Chapter 6, "Hooks", on page 461
  - For how to sort jobs on a key, see section 4.8.42 “Sorting Jobs on a Key” on page 302
4.3.5.2.2 Routing Jobs to Hardware via Hook

You can send jobs to particular hardware without using a particular queue, by using a hook. See section 4.8.39.4.1 “Using Hooks to Tag Jobs” on page 297.

4.3.6 Scheduling Jobs into Time Slots

You can schedule jobs in time slots in the following ways:

- Set aside time slots for specific entities; see section 4.3.6.1 “Setting Aside Time Slots for Entities” on page 109
- Lock entities into specific time slots; see section 4.3.6.2 “Locking Entities into Time Slots” on page 110

4.3.6.1 Setting Aside Time Slots for Entities

You can set aside time slots that are reserved exclusively for certain users or groups. You can use reservations, dedicated time, or non-primetime.

4.3.6.1.1 Reservations

Reservations set aside one or more blocks of time on the requested resources. Users can create their own reservations, or you can create them and set their access control to allow only specified users to submit jobs to them. See section 4.8.37 “Advance and Standing Reservations” on page 283.

4.3.6.1.2 Dedicated Time

During dedicated time, the only jobs allowed to run are those in dedicated queues. The drawback to dedicated time is that it applies to the entire complex. If you want to set aside one or more time slots for a user or group, do the following:

- Create a dedicated queue. See section 2.2.5.2.1 “Dedicated Time Queues” on page 25.
- Define dedicated time. See section 4.8.10 “Dedicated Time” on page
171.

- Set access control on the dedicated queue so that only the particular users or groups you want can submit jobs to the queue. See section 2.2.6.5 “Using Access Control to Route Jobs” on page 33.

4.3.6.1.3 Non-primetime

You can set up primetime and non-primetime so that one of them, for example, non-primetime, is used as a special time slot allocated to particular users or groups. The advantage of using non-primetime is that you can set up a separate scheduling policy for it, for example, using fairshare during non-primetime and sorting jobs on a key during primetime. Note that the formula, if defined, is in force all of the time. To use non-primetime, do the following:

- Create a non-primetime queue; see section 2.2.3 “Creating Queues” on page 22 and section 2.2.5.2.2 “Primetime and Non-Primetime Queues” on page 25.
- Define primetime and non-primetime; see section 4.8.34 “Using Primetime and Holidays” on page 274.
- Set access control on the non-primetime queue so that only the particular users or groups you want can submit jobs to the queue. See section 2.2.6.5 “Using Access Control to Route Jobs” on page 33.
- Make sure that the scheduling policy you want is in force during non-primetime. See section 4.8.34.1 “How Primetime and Holidays Work” on page 274.

4.3.6.2 Locking Entities into Time Slots

You can make all jobs from some users or groups run during designated time slots. You can run them during a reservation, dedicated time, or non-primetime.

4.3.6.2.1 Locking Entities into Reservations

To allow a user to submit jobs only into a reservation, do the following:

- Create a reservation for the resources and time(s) you want the controlled user(s) to use. When creating the reservation, set access control to allow the controlled user(s). See section 4.8.37 “Advance and
Standing Reservations” on page 283 and section 9.3.8.1 “Setting Reservation Access” on page 673.

- Set access control on all queues except the reservation’s queue to deny the controlled user(s); see section 2.2.6.5 “Using Access Control to Route Jobs” on page 33.

### 4.3.6.2.2 Locking Entities into Dedicated Time

You can create a dedicated time queue, and send all jobs from controlled users to that queue. You can route their jobs to it, and you can allow them to submit directly to it. To lock one or more users or groups into one or more dedicated time slots, do the following:

- Create a dedicated time queue; see section 2.2.3 “Creating Queues” on page 22 and section 2.2.5.2.1 “Dedicated Time Queues” on page 25.
- Create at least one other execution queue; see section 2.2.3 “Creating Queues” on page 22.
- Create a routing queue; see section 2.2.3 “Creating Queues” on page 22.
- Prevent controlled users from submitting to non-dedicated time execution queues: set from_route_only to True on the non-dedicated time execution queues. See section 2.2.5.1 “Where Execution Queues Get Their Jobs” on page 23.
- Use queue access control to allow jobs from the controlled users or groups into the dedicated time queue only. See section 2.2.6.5 “Using Access Control to Route Jobs” on page 33.
- Make the routing queue be the default queue:

  ```
  Qmgr: set server default_queue = <routing queue name>
  ```

  Note that you can either allow all users into the dedicated time queue, or allow only the controlled users into the dedicated time queue.

### 4.3.6.2.3 Locking Entities into Non-primetime

You can create a non-primetime queue, and send all jobs from controlled users to that queue. You can route their jobs to it, and you can allow them to submit directly to it. To lock one or more users or groups into one or more non-primetime slots, do the following:

- Create a non-primetime queue; see section 2.2.3 “Creating Queues” on
• Create at least one other execution queue; see section 2.2.3 “Creating Queues” on page 22.

• Create a routing queue; see section 2.2.3 “Creating Queues” on page 22.

• Prevent controlled users from submitting to primetime execution queues: set from_route_only to True on the primetime execution queues. See section 2.2.5.1 “Where Execution Queues Get Their Jobs” on page 23.

• Make the routing queue be the default queue:

  Qmgr: set server default_queue = <routing queue name>

• Use queue access control to allow jobs from the controlled users or groups into the non-primetime queue only. See section 2.2.6.5 “Using Access Control to Route Jobs” on page 33.

• Define primetime and non-primetime; see section 4.8.34 “Using Prime-time and Holidays” on page 274.

• Make sure that the scheduling policy you want is in force during non-primetime. See section 4.8.34.1 “How Primetime and Holidays Work” on page 274.

Note that you can either allow all users into the non-primetime queue, or allow only the controlled users into the non-primetime queue.

### 4.3.7 Default Scheduling Policy

The default scheduling policy is determined by the default settings for all of the attributes, parameters, etc. that determine the scheduler’s behavior. For a list of all of these elements, see section 4.4.1 “Configuring the Scheduler” on page 117.

The default behavior of the scheduler is the following:

• The scheduler matches jobs with available resources. This means that the scheduler places each job only where that job has enough resources to run. See section 4.8.28 “Matching Jobs to Resources” on page 222.

• The scheduler will not over-allocate the resources that are listed in the
The scheduler sorts vnodes according to its `node_sort_key` parameter, whose default setting is the following:

```
node_sort_key: "sort_priority HIGH all"
```

This means that vnodes are sorted by the value of their `priority` attribute, with high-priority vnodes used first. The scheduler places jobs first on vnodes that are first in the sorted list.

Note that all vnodes have the same default priority upon creation, so the default sorted order for vnodes is undefined.

See section 4.8.47 “Sorting Vnodes on a Key” on page 312.

- Queues are sorted according to the value of their `priority` attribute, so that queues with a higher priority are considered before those with a lower priority. See section 2.2.5.3 “Prioritizing Execution Queues” on page 26.

- Jobs are considered according to the priority of their queues. The scheduler runs all of the jobs that it can from the highest-priority queue before moving to the next queue, and so on. See section 4.8.4 “Examining Jobs Queue by Queue” on page 151.

- Within each queue, jobs are considered in submission order.

- Starving jobs are given a special priority called `starving`. The default time required to become a starving job is 24 hours. See section 4.8.45 “Starving Jobs” on page 307.

- Jobs in an express queue are placed in the `express_queue` preemption priority level. They are also placed in the `Express` execution priority class. The default priority for a queue to be an express queue is 150. See section 2.2.5.3.1 “Express Queues” on page 26.

- Queued jobs are sorted according to their priority. Special jobs are all prioritized ahead of normal jobs, without regard to the queue in which they reside. The order for job priority for special jobs, highest first, is reservation jobs, jobs in express queues, starving jobs, suspended jobs. After this, the scheduler looks at normal jobs, queue by queue. All jobs in express queues, all starving jobs, and all suspended jobs are consid-
ered before the scheduler looks at the individual queues.

See section 4.8.16 “Calculating Job Execution Priority” on page 187.

• The scheduler will preempt lower-priority jobs in order to run higher-priority jobs (preemptive_sched is True by default). By default, it has two levels of job priority, express_queue, and normal_jobs, where express_queue jobs can preempt normal_jobs. This is set in the scheduler’s preempt_prio parameter.

When the scheduler chooses among jobs of the same priority for a job to preempt, it uses the default setting for preempt_sort, which is min_time_since_start, choosing jobs that have been running for the shortest time.

When the scheduler chooses how to preempt a job, it uses the default setting for its preempt_order parameter, which is SCR, meaning that first it will attempt suspension, then checkpointing, then if necessary requeueing.

See section 4.8.33 “Using Preemption” on page 258.

• The scheduler will do its best to backfill smaller jobs around the job it has decided is the most important job. See section 4.8.3 “Using Backfilling” on page 145.

• Primetime is 6:00 AM to 5:30 PM. Any holiday is considered non-primetime. Standard U.S. Federal holidays for the year are provided in the file PBS_HOME/sched_priv/holidays. These dates should be adjusted yearly to reflect your local holidays. See section 4.8.34 “Using Primetime and Holidays” on page 274.

• The scheduler runs every 10 minutes unless a new job is submitted or a job finishes execution. See section 4.4.4 “The Scheduling Cycle” on page 129.

### 4.3.8 Examples of Workload and Policy

• If you need to have high-priority jobs run soon, and nothing distinguishes the high-priority jobs from the rest:
  - Create advance reservations for the high-priority jobs, and have users submit those jobs to the reservations; see section 4.8.37
“Advance and Standing Reservations” on page 283

- If you want to run jobs in submission order:
  - FIFO; see section 4.8.19 “FIFO Scheduling” on page 203

- If you have low-priority jobs that should run only when other jobs don’t need the resources:
  - Set up an anti-express queue; see section 4.8.1 “Anti-Express Queues” on page 140

- If you have a mix of jobs, and want to run big jobs first:
  - Sort jobs on a key, using `ncpus` as the key, to run big jobs first; see section 4.3.5 “Scheduling Jobs According to Size Etc.” on page 106

- If you have a mix of jobs, and want to give big jobs high priority, but avoid having idle resources:
  - Sort jobs on a key, using `ncpus` as the key, to run big jobs first; see section 4.3.5 “Scheduling Jobs According to Size Etc.” on page 106
  - Use backfilling; see section 4.8.3 “Using Backfilling” on page 145

- If you want to have all users start about the same number of jobs:
  - Use round robin, and give each user their own queue; see section 4.8.38 “Round Robin Queue Selection” on page 290

- If you want to always give each user access to a certain amount of a resource, but allow more if no one else is using it:
  - Use soft limits for the amount each user can use; see section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410 and section 4.8.33 “Using Preemption” on page 258

- If your site has more than one funding source:
  - See section 4.3.4 “Allocating Resources by User, Project or Group” on page 102

- If you have lots of users in a complex:
  - Use resource limits; see section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410, or
  - Use fairshare; see section 4.8.18 “Using Fairshare” on page 191

- If you have jobs that must run at the end of the day:
  - Use dependencies for end-of-day accounting; see section 4.8.11
“Dependencies” on page 173

• If you need to ensure that jobs run in certain hours on desktops:
  - Use cycle harvesting; see section 4.8.9 “Using Idle Workstation Cycle Harvesting” on page 159, or
  - Use primetime & non-primetime for nighttime; see section 4.8.34 “Using Primetime and Holidays” on page 274

• If you want to be sure a job will run:
  - Create an advance reservation; see section 4.8.37 “Advance and Standing Reservations” on page 283

• If you have more than one complex, and you want to balance the workload across the complexes:
  - Use peer scheduling; see section 4.8.31 “Peer Scheduling” on page 229

• If you have some jobs that should prefer to run on one set of vnodes, and other jobs that should prefer to run on another set of vnodes, but if the preferred vnodes are busy, a job can run on the non-preferred vnodes:
  - Use peer scheduling. Set up two complexes, give the pulling queues low priority, and use queue priority in the job sorting formula. See section 4.8.31 “Peer Scheduling” on page 229, section 2.2.5.3 “Prioritizing Execution Queues” on page 26, and section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205. You can use a routing queue to send jobs to the correct complex. See section 2.2.6 “Routing Queues” on page 26

• If you have two (or more) sets of vnodes, and jobs should run on one set or the other, but not both. Additionally, jobs should not have to request where they run. If one set of vnodes is new, and one is old:
  - Use a routing queue and two execution queues. Associate each execution queue with one set of vnodes. Put the execution queue for the preferred set of vnodes first in the routing list, but put a limit on the number of queued jobs in the execution queues, so that both queues will fill up. Otherwise the routing queue will preferentially fill the first in its routing list. See section 2.2.6 “Routing Queues” on page 26, and section 4.8.2 “Associating Vnodes with Queues” on page 141

• If you need to apportion a single vnode or cluster according to owner-
ship:
- See section 4.3.4 “Allocating Resources by User, Project or Group” on page 102

- If you have more than one high-priority queue, and at least one low-priority queue, and you want all jobs in high-priority queues to be considered as one group, and run in submission order:
  - Use the job sorting formula to sort jobs on queue priority:
    \[ \text{set server job_sort_formula} = \text{queue_priority} \]
  - Give all queues whose jobs should be considered together the same priority
  - Set the \text{by_queue} scheduler attribute to \text{False}

- If you want to place jobs on the vnodes with the fewest CPUs first, saving bigger vnodes for larger jobs:
  - Sort vnodes so that those with fewer CPUs come first:
    \[ \text{node_sort_key: “ncpus LOW”} \]

4.4 The Scheduler

The scheduler, \text{pbs_sched}, implements scheduling policy. The scheduler communicates with the MOMs to query the state of host-level resources and with the Server to learn about the availability of jobs to execute and the state of server-level resources. The interface to the server is through the same API as is used by the client commands. The scheduler communicates with the server with the privilege of a PBS Manager.

4.4.1 Configuring the Scheduler

4.4.1.1 Where the Scheduler Gets Its Information

The behavior of the scheduler is controlled by the information in the following attributes and files of parameters and settings:

The scheduler gets its information from the following sources:
PBS_HOME/sched_priv/resource_group
Contains the description of the fairshare tree. Created by you. Can be edited. Read on startup and HUP of scheduler.

PBS_HOME/sched_priv/usage
Contains the usage database. Do not edit. Instead, use the pbsfs command while the scheduler is stopped; see “pbsfs” on page 120 of the PBS Professional Reference Guide. Written every cycle and HUP. Read on startup. Cannot be altered while scheduler is running.

PBS_HOME/sched_priv/sched_config
Contains scheduler configuration options, also called scheduler parameters, e.g. backfill, job_sort_key. Read on startup and HUP.

Can be edited. Each entry must be a single, unbroken line. Entries must be double-quoted if they contain whitespace.

PBS_HOME/sched_priv/dedicated_time
Contains definitions of dedicated time. Can be edited. Read on startup and HUP.

PBS_HOME/sched_priv/holidays
Contains definitions of holidays. Can be edited. Read on startup and HUP.

Options to pbs_sched command
Control some scheduler behavior. Set on invocation.

Scheduler attributes
Control some scheduler behavior. Can be set using qmgr. Read every scheduling cycle.

Server attributes
Several server attributes control scheduler behavior. Can be set using qmgr. The following table lists the server attributes that affect scheduling, along with a brief description. Read every scheduling cycle.

Some limit attributes are marked as “old”. These are incompatible with, and are replaced by, the new limit attributes described in section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.
For a complete description of each attribute, see “Server Attributes” on page 365 of the PBS Professional Reference Guide.

Table 4-1: Server Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>backfill_depth</td>
<td>Modifies backfilling behavior. Sets the number of jobs that are to be backfilled around.</td>
</tr>
<tr>
<td>default_queue</td>
<td>Specifies queue for jobs that don’t request a queue</td>
</tr>
<tr>
<td>eligible_time_enable</td>
<td>Controls starving behavior.</td>
</tr>
<tr>
<td>est_start_time_freq</td>
<td>Interval at which PBS calculates estimated start times and vnodes for all jobs.</td>
</tr>
<tr>
<td>job_sort_formula</td>
<td>Formula for computing job priorities.</td>
</tr>
<tr>
<td>max_group_res</td>
<td>Old. The maximum amount of the specified resource that any single group may consume in this PBS complex.</td>
</tr>
<tr>
<td>max_group_res_soft</td>
<td>Old. The soft limit for the specified resource that any single group may consume in this complex.</td>
</tr>
<tr>
<td>max_group_run</td>
<td>Old. The maximum number of jobs owned by the users in one group allowed to be running within this complex at one time.</td>
</tr>
<tr>
<td>max_group_run_soft</td>
<td>Old. The maximum number of jobs owned by the users in one group allowed to be running in this complex at one time.</td>
</tr>
</tbody>
</table>
### Table 4-1: Server Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_queued</td>
<td>The maximum number of jobs allowed to be queued in or running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_queued_res.&lt;resource&gt;</td>
<td>The maximum amount of the specified resource allowed to be allocated to jobs queued in or running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run</td>
<td>The maximum number of jobs allowed to be running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run_res.&lt;resource&gt;</td>
<td>The maximum amount of the specified resource allowed to be allocated to jobs running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run_res_soft.&lt;resource&gt;</td>
<td>Soft limit on the amount of the specified resource allowed to be allocated to jobs running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_run_soft</td>
<td>Soft limit on the number of jobs allowed to be running in the complex. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td>max_running</td>
<td>Old. The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs.</td>
</tr>
<tr>
<td>max_user_res</td>
<td>Old. The maximum amount within this complex that any single user may consume of the specified resource.</td>
</tr>
</tbody>
</table>
### Table 4-1: Server Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_user_res_soft</td>
<td>Old. The soft limit on the amount of the specified resource that any single user may consume within a complex.</td>
</tr>
<tr>
<td>max_user_run</td>
<td>Old. The maximum number of jobs owned by a single user allowed to be running within the complex at one time.</td>
</tr>
<tr>
<td>max_user_run_soft</td>
<td>Old. The soft limit on the number of jobs owned by a single user that are allowed to be running within this complex at one time.</td>
</tr>
<tr>
<td>node_fail_requeue</td>
<td>Controls whether running jobs are automatically requeued or are deleted when the primary execution vnode fails. Number of seconds to wait after losing contact with Mother Superior before requeueing or deleting jobs.</td>
</tr>
<tr>
<td>node_group_enable</td>
<td>Specifies whether node grouping is enabled.</td>
</tr>
<tr>
<td>node_group_key</td>
<td>Specifies the resource to use for node grouping.</td>
</tr>
<tr>
<td>resources_available</td>
<td>The list of available resources and their values defined on the server.</td>
</tr>
<tr>
<td>resources_max</td>
<td>The maximum amount of each resource that can be requested by any single job in this complex, if there is not a resources_max value defined for the queue at which the job is targeted.</td>
</tr>
</tbody>
</table>
Scheduling

Several vnode attributes control scheduler behavior. Can be set using `qmgr`. The following table lists the vnode attributes that affect scheduling, along with a brief description. Read every scheduling cycle. For a complete description of each attribute, see “Vnode Attributes” on page 420 of the PBS Professional Reference Guide.

**Table 4-2: Vnode Attributes Involved in Scheduling**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_aoe</td>
<td>This attribute identifies the AOE currently instantiated on this vnode.</td>
</tr>
<tr>
<td>max_group_run</td>
<td>The maximum number of jobs owned by any users in a single group allowed to run on this vnode at one time.</td>
</tr>
<tr>
<td>max_running</td>
<td>The maximum number of jobs allowed to be run on this vnode at any given time.</td>
</tr>
<tr>
<td>max_user_run</td>
<td>The maximum number of jobs owned by a single user allowed to run on this vnode at one time.</td>
</tr>
<tr>
<td>no_multinode_jobs</td>
<td>Controls whether jobs which request more than one chunk are allowed to execute on this vnode.</td>
</tr>
</tbody>
</table>
Several queue attributes control scheduler behavior. These are attributes such as `node_group_key`. Can be set using `qmgr`. The following table lists the queue attributes that affect scheduling, along with a brief description. Read every scheduling cycle. For a complete description of each

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>The priority of this vnode compared with other vnodes.</td>
</tr>
<tr>
<td>provision_enable</td>
<td>Controls whether this vnode can be provisioned.</td>
</tr>
<tr>
<td>queue</td>
<td>The queue with which this vnode is associated.</td>
</tr>
<tr>
<td>resources_available</td>
<td>The list of resources and the amounts available on this vnode</td>
</tr>
<tr>
<td>sharing</td>
<td>Specifies whether more than one job at a time can use the resources of the vnode or the vnode’s host.</td>
</tr>
<tr>
<td>state</td>
<td>Shows or sets the state of the vnode.</td>
</tr>
<tr>
<td>pcpus</td>
<td>The number of physical CPUs on the vnode.</td>
</tr>
<tr>
<td>resources_assigned</td>
<td>The total amount of each resource allocated to jobs running on this vnode.</td>
</tr>
</tbody>
</table>
attribute, see “Queue Attributes” on page 407 of the PBS Professional Reference Guide.

Table 4-3: Queue Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Specifies whether this queue accepts new jobs.</td>
</tr>
<tr>
<td>from_route_only</td>
<td>Specifies whether this queue accepts jobs only from routing queues, or from both execution and routing queues.</td>
</tr>
<tr>
<td>max_array_size</td>
<td>The maximum number of subjobs that are allowed in an array job.</td>
</tr>
<tr>
<td>max_group_res</td>
<td>Old. The maximum amount of the specified resource that any single group may consume in this queue.</td>
</tr>
<tr>
<td>max_group_res__soft</td>
<td>Old. The soft limit for the specified resource that any single group may consume in this queue.</td>
</tr>
<tr>
<td>max_group_run</td>
<td>Old. The maximum number of jobs owned by the users in one group allowed to be running within this queue at one time.</td>
</tr>
<tr>
<td>max_group_run__soft</td>
<td>Old. The maximum number of jobs owned by the users in one group allowed to be running in this queue at one time.</td>
</tr>
<tr>
<td>max_queuable</td>
<td>Old. The maximum number of jobs allowed to reside in the queue at any given time.</td>
</tr>
<tr>
<td>max_queued</td>
<td>The maximum number of jobs allowed to be queued in or running from the queue. Can be specified for users, groups, or all.</td>
</tr>
</tbody>
</table>
### Table 4-3: Queue Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>max_queued_res.&lt;resource&gt;</code></td>
<td>The maximum amount of the specified resource allowed to be allocated to jobs queued in or running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td><code>max_run</code></td>
<td>The maximum number of jobs allowed to be running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td><code>max_run_res.&lt;resource&gt;</code></td>
<td>The maximum amount of the specified resource allowed to be allocated to jobs running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td><code>max_run_res_soft.&lt;resource&gt;</code></td>
<td>Soft limit on the amount of the specified resource allowed to be allocated to jobs running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td><code>max_run_soft</code></td>
<td>Soft limit on the number of jobs allowed to be running from the queue. Can be specified for users, groups, or all.</td>
</tr>
<tr>
<td><code>max_running</code></td>
<td>Old. The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs.</td>
</tr>
<tr>
<td><code>max_user_res</code></td>
<td>Old. The maximum amount of the specified resource that the jobs of any single user may consume.</td>
</tr>
<tr>
<td><code>max_user_res_soft</code></td>
<td>Old. The soft limit on the amount of the specified resource that any single user may consume in this queue.</td>
</tr>
</tbody>
</table>
Table 4-3: Queue Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_user_run</td>
<td>Old. The maximum number of jobs owned by a single user allowed to be running from the queue at one time.</td>
</tr>
<tr>
<td>max_user_run_soft</td>
<td>Old. The soft limit on the number of jobs owned by a single user that are allowed to be running from this queue at one time.</td>
</tr>
<tr>
<td>node_group_enable</td>
<td>Specifies whether node grouping is enabled.</td>
</tr>
<tr>
<td>node_group_key</td>
<td>Specifies the resource to use for node grouping.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priority of this queue compared to other queues of the same type in this PBS complex.</td>
</tr>
<tr>
<td>resources_assigned</td>
<td>The total of each type of resource allocated to jobs running in this queue.</td>
</tr>
<tr>
<td>resources_available</td>
<td>The list of available resources and their values defined on the queue.</td>
</tr>
<tr>
<td>resources_max</td>
<td>The maximum amount of each resource that can be requested by any single job in this queue.</td>
</tr>
<tr>
<td>resources_min</td>
<td>The minimum amount of each resource that can be requested by a single job in this queue.</td>
</tr>
<tr>
<td>route_destinations</td>
<td>The list of destinations to which jobs may be routed.</td>
</tr>
<tr>
<td>route_held_jobs</td>
<td>Specifies whether jobs in the held state can be routed from this queue.</td>
</tr>
</tbody>
</table>
List of jobs and server-level resources queried from Server
Cannot be altered. Read every scheduling cycle.

List of host-level resources queried from MOMs
Cannot be altered. Read every scheduling cycle.

### 4.4.1.2 Reference Copies of Files

PBS is installed with a reference copy of the current year’s holidays file, in `PBS_EXEC/etc/pbs_holidays`.

---

### Table 4-3: Queue Attributes Involved in Scheduling

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>route_lifetime</code></td>
<td>The maximum time a job is allowed to reside in a routing queue. If a job cannot be routed in this amount of time, the job is aborted.</td>
</tr>
<tr>
<td><code>route_retry_time</code></td>
<td>Time delay between routing retries. Typically used when the network between servers is down.</td>
</tr>
<tr>
<td><code>route_waiting_jobs</code></td>
<td>Specifies whether jobs whose <code>execution_time</code> attribute value is in the future can be routed from this queue.</td>
</tr>
<tr>
<td><code>started</code></td>
<td>Specifies whether jobs in this queue can be scheduled for execution.</td>
</tr>
<tr>
<td><code>state_count</code></td>
<td>The number of jobs in each state currently residing in this queue.</td>
</tr>
</tbody>
</table>
4.4.2 Making the Scheduler Read its Configuration

If you change the scheduler’s configuration file, the scheduler must re-read it for the changes to take effect. To get the scheduler to re-read its configuration information, without stopping the scheduler, you can HUP the scheduler:

```
kill -HUP <scheduler PID>
```

4.4.3 Starting, Stopping, and Restarting the Scheduler

4.4.3.1 When and How to Start the Scheduler

During normal operation, startup of the scheduler is handled automatically. The PBS daemons are started automatically at bootup by the PBS start/stop script. During failover, the secondary server automatically tries to use the primary scheduler, and if it cannot, it starts its own scheduler.

To start the scheduler by hand:

```
PBS_EXEC/sbin/pbs_sched [options]
```

See “pbs_sched” on page 105 of the PBS Professional Reference Guide.

4.4.3.2 When and How to Stop the Scheduler

You must stop the scheduler for the following operations:

- Using the `pbsfs` command; see “pbsfs” on page 120 of the PBS Professional Reference Guide.
- Upgrading PBS Professional; see section “Upgrading PBS Professional” on page 127 in the PBS Professional Installation & Upgrade Guide.
The scheduler traps signals during the scheduling cycle. You can kill the scheduler at the end of the cycle, or if necessary, immediately. The scheduler does not write the fairshare usage file when it is killed with -9, but it does write the file when it is killed without -9.

You must be root on the scheduler’s host.

To stop the scheduler at the end of a cycle:

```
kill <scheduler PID>
```

To stop the scheduler immediately:

```
kill -9 <scheduler PID>
```

### 4.4.3.3 When and How to Restart the Scheduler

Under most circumstances, when you restart the scheduler, you do not need to specify any options to the `pbs_sched` command. See “pbs_sched” on page 105 of the PBS Professional Reference Guide. Start the scheduler this way:

```
PBS_EXEC/sbin/pbs_sched [options]
```

The PBS start/stop script is not recommended for restarting the scheduler after stopping it, because the script causes any running jobs to be killed.

### 4.4.4 The Scheduling Cycle

The scheduler runs in a loop. Inside each loop, it starts up, performs all of its work, and then stops. The scheduling cycle is triggered by a timer and by several possible events.

When there are no events to trigger the scheduling cycle, it is started by a timer. The time between starts is set in the server’s `scheduler_iteration` server attribute. The default value is 10 minutes.

The maximum duration of the cycle is set in the scheduler’s `sched_cycle_length` attribute. The scheduler will terminate its cycle if the duration of the cycle exceeds the value of the attribute. The default value for the length of the scheduling cycle is 20 minutes.
4.4.4.1 Triggers for Scheduling Cycle

The scheduler starts when the following happen:

- The specified amount of time has passed since the previous start
- A job is submitted
- A job finishes execution.
- A new reservation is created
- A reservation starts
- Scheduling is enabled
- The server comes up
- A job is qrun
- A queue is started
- A job is moved to a local queue
- Eligible wait time for jobs is enabled
- A reservation is re-confirmed after being degraded

4.4.4.1.1 Logging Scheduling Trigger

The server triggers scheduler cycles. The reason for triggering a scheduling cycle is logged by the server. See section 13.4.4.2 “Scheduler Commands” on page 896.
4.4.4.2 Actions During Scheduling Cycle

The following is a list of the scheduler’s actions during a scheduling cycle. The list is not in any special order.

- The scheduler gets the state of the world:
  - The scheduler queries the server for the following:
    - Status of jobs in queues
    - All global server, queue, and host-level resources
    - Server, queue, vnode, and scheduler attribute settings
    - Reservations
  - The scheduler runs dynamic server resource queries for resources listed in “server_dyn_res” line in sched_config
  - The scheduler runs dynamic host-level resource queries for resources listed in “mom_resources” line in sched_config
- The scheduler logs a message at the beginning of each scheduling cycle saying whether it is primetime or not, and when this period of primetime or non-primetime will end. The message is at log event class 0x0100. The message is of this form:
  “It is primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS”
  or
  “It is non-primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS”
- Given scheduling policy, available jobs and resources, and scheduling cycle length, the scheduler examines as many jobs as it can, and runs as many jobs as it can.
4.4.5 How Available Consumable Resources are Counted

When the scheduler checks for available consumable resources, it uses the following calculation:

\[ \text{resources\_available.}<\text{resource}> - \text{total resources assigned for this resource} \]

\text{total resources assigned} is the total amount of \text{resources\_assigned.}<\text{resource}> for all other running jobs.

For example, if the scheduler is calculating available memory, and two other jobs are running, each with 2GB of memory assigned, and \text{resources\_available.mem} is 8GB, the scheduler figures that it has 4GB to work with.

4.5 Using Queues in Scheduling

A queue is a PBS mechanism for holding jobs. PBS has queue-based tools for handling jobs; for example, you can set queue-based limits on resource usage by jobs. PBS uses queues for a variety of purposes. Before reading this section, please familiarize yourself with the mechanics of creating and configuring queues, by reading section 2.2 “Queues” on page 19.

Queues are used in the following ways:

- Holding submitted jobs
- Prioritizing jobs and ordering job selection:
  - PBS provides tools for selecting jobs according to the queue they are in; see section 4.2.5.3 “Using Queue-based Tools to Prioritize Jobs” on page 77
  - Queue priority can be used in calculating job priority; see section 4.8.36 “Queue Priority” on page 282
- Providing tools for managing time slots
  - Reservations: you can reserve specific resources for defined time slots. Queues are used for advance and standing reservations; see
section 4.8.37 “Advance and Standing Reservations” on page 283, and section 7.8 “Advance and Standing Reservation of Resources” on page 181

- Dedicated time; see section 4.8.10 “Dedicated Time” on page 171
- Primetime and holidays; see section 4.8.34 “Using Primetime and Holidays” on page 274

• Routing jobs: Many ways to route jobs are listed in section 4.8.39 “Routing Jobs” on page 292

• Providing tools for managing resources
  - Managing resource usage by users; see section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410
  - Managing resource usage by jobs; see section 5.15.3 “Placing Resource Limits on Jobs” on page 436
  - Setting resource and job limits used for preemption: you can specify how much of a resource or how many jobs a user or group can use before their jobs are eligible to be preempted. See section 5.15.1.3 “Hard and Soft Limits” on page 414 and section 4.8.33 “Using Preemption” on page 258.
  - Assigning default resources to jobs; see section 5.9.4 “Allocating Default Resources to Jobs” on page 342

### 4.6 Scheduling Restrictions and Caveats

#### 4.6.1 One Policy Per PBS Complex

The scheduler runs a single scheduling policy, and applies it to the entire PBS complex. You cannot have two different scheduling policies on two different queues or partitions.
### 4.6.2 Jobs that Cannot Run on Current Resources

The scheduler checks to see whether each job could possibly run now, counting resources as if there were no other jobs, and all current resources could be used by this job. The scheduler counts resources only from those vnodes that are on line. If a vnode is marked `offline`, its resources are not counted.

A job that cannot run on current resources will sit in the queue until it becomes the most deserving job. Whenever this job is considered for being run, and backfilling is being used, the following error message is printed in the scheduler’s log file:

```
"resource request is impossible to solve: job will never run"
```

The scheduler then examines the next job in line to be the most deserving job.

The scheduler determines whether a job cannot run on current resources only when backfilling is used. If backfilling is turned off, then the scheduler won't determine whether or not a job has requested more than can be supplied by current resources. It decides only that it can't run now.

### 4.6.3 Resources Not Controlled by PBS

When the scheduler runs each cycle, it gets the state of its world, including dynamic resources outside of the control of PBS. If non-PBS processes are running on the vnodes PBS uses, it is possible that another process will use enough of a dynamic resource such as scratch space to prevent a PBS job that requested that resource from running.

### 4.6.4 No Pinning of Processes to Cores

PBS does not pin processes to cores. This can be accomplished in the job launch script using, for example, `taskset` or `dplace`. 
4.7 Errors and Logging

4.7.1 Logfile for scheduler

You can set the scheduler’s logging to record different kinds of events. See section 13.4.3.1.3 “Specifying Scheduler Log Events” on page 894.

The server triggers scheduler cycles. The reason for triggering a scheduling cycle is logged by the server. See section 13.4.4.2 “Scheduler Commands” on page 896.

4.8 Scheduling Tools

In this section (all of section 4.8 “Scheduling Tools” on page 135, and its subsections), we describe each scheduling tool, including how to configure it.

The following table lists PBS scheduling tools, with links to descriptions:

<table>
<thead>
<tr>
<th>Scheduling Tool</th>
<th>Incompatible Tools</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anti-express queue</td>
<td>soft queue limits</td>
<td>See section 4.8.1 “Anti-Express Queues” on page 140</td>
</tr>
<tr>
<td>Associating vnodes with queues</td>
<td></td>
<td>See section 4.8.2 “Associating Vnodes with Queues” on page 141</td>
</tr>
<tr>
<td>Backfilling</td>
<td>fairshare or pre-emption w/backfilling+strict ordering</td>
<td>See section 4.8.3 “Using Backfilling” on page 145</td>
</tr>
</tbody>
</table>
Examining jobs queue-by-queue | round robin, queues as fair-share entities | See section 4.8.4 “Examining Jobs Queue by Queue” on page 151
Checkpointing | | See section 4.8.5 “Checkpointing” on page 153
Organizing job chunks | | See section 4.8.6 “Organizing Job Chunks” on page 154
cron jobs, Windows Task Scheduler |  | See section 4.8.7 “cron Jobs, or the Windows Task Scheduler” on page 155
Custom resources | | See section 4.8.8 “Using Custom and Default Resources” on page 155
Cycle harvesting | reservations | See section 4.8.9 “Using Idle Workstation Cycle Harvesting” on page 159
Dedicated time | | See section 4.8.10 “Dedicated Time” on page 171
Default resources | | See section 4.8.8 “Using Custom and Default Resources” on page 155
Dependencies | | See section 4.8.11 “Dependencies” on page 173
Dynamic resources (server & host) | | See section 4.8.12 “Dynamic Resources” on page 173
Eligible wait time for jobs | | See section 4.8.13 “Eligible Wait Time for Jobs” on page 173
<table>
<thead>
<tr>
<th>Scheduling Tool</th>
<th>Incompatible Tools</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity shares (was strict priority)</td>
<td>formula, fair-share, FIFO</td>
<td>See section 4.8.14 “Sorting Jobs by Entity Shares (Was Strict Priority)” on page 180</td>
</tr>
<tr>
<td>Estimating job start time</td>
<td></td>
<td>See section 4.8.15 “Estimating Job Start Time” on page 181</td>
</tr>
<tr>
<td>Calculating job execution priority</td>
<td></td>
<td>See section 4.8.16 “Calculating Job Execution Priority” on page 187</td>
</tr>
<tr>
<td>Express queues</td>
<td></td>
<td>See section 4.8.17 “Express Queues” on page 190</td>
</tr>
<tr>
<td>Fairshare</td>
<td>formula, starving, strict ordering, using the fair_share_perc option to job_sort_key</td>
<td>See section 4.8.18 “Using Fair-share” on page 191</td>
</tr>
<tr>
<td>FIFO</td>
<td></td>
<td>See section 4.8.19 “FIFO Scheduling” on page 203</td>
</tr>
<tr>
<td>Formula</td>
<td>job_sort_key, fairshare</td>
<td>See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205</td>
</tr>
<tr>
<td>Gating jobs at server or queue</td>
<td></td>
<td>See section 4.8.21 “Gating Jobs at Server or Queue” on page 213</td>
</tr>
<tr>
<td>Managing application licenses</td>
<td></td>
<td>See section 4.8.22 “Managing Application Licenses” on page 214</td>
</tr>
<tr>
<td>Limits on per-job resource usage</td>
<td></td>
<td>See section 4.8.23 “Limits on Per-job Resource Usage” on page 215</td>
</tr>
</tbody>
</table>
### Table 4-4: List of Scheduling Tools

<table>
<thead>
<tr>
<th>Scheduling Tool</th>
<th>Incompatible Tools</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limits on user and group jobs</td>
<td></td>
<td>See section 4.8.24 “Limits on User and Group Jobs” on page 215</td>
</tr>
<tr>
<td>Limits on user and group resource usage</td>
<td></td>
<td>See section 4.8.25 “Limits on User and Group Resource Usage” on page 216</td>
</tr>
<tr>
<td>Limits on jobs at vnodes</td>
<td></td>
<td>See section 4.8.26 “Limits on Jobs at Vnodes” on page 216</td>
</tr>
<tr>
<td>Load balancing</td>
<td>node_sort_key using unused or assigned options,</td>
<td>See section 4.8.27 “Using Load Balancing” on page 216</td>
</tr>
<tr>
<td>Matching jobs to resources</td>
<td></td>
<td>See section 4.8.28 “Matching Jobs to Resources” on page 222</td>
</tr>
<tr>
<td>Node grouping</td>
<td></td>
<td>See section 4.8.29 “Node Grouping” on page 225</td>
</tr>
<tr>
<td>Overrides</td>
<td></td>
<td>See section 4.8.30 “Overrides” on page 226</td>
</tr>
<tr>
<td>Peer scheduling</td>
<td></td>
<td>See section 4.8.31 “Peer Scheduling” on page 229</td>
</tr>
<tr>
<td>Placement sets</td>
<td></td>
<td>See section 4.8.32 “Placement Sets” on page 237</td>
</tr>
<tr>
<td>Preemption</td>
<td></td>
<td>See section 4.8.33 “Using Preemption” on page 258</td>
</tr>
<tr>
<td>Primetime and holidays</td>
<td></td>
<td>See section 4.8.34 “Using Primetime and Holidays” on page 274</td>
</tr>
<tr>
<td>Provisioning</td>
<td></td>
<td>See section 4.8.35 “Provisioning” on page 281</td>
</tr>
</tbody>
</table>
## Table 4-4: List of Scheduling Tools

<table>
<thead>
<tr>
<th>Scheduling Tool</th>
<th>Incompatible Tools</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue priority</td>
<td></td>
<td>See section 4.8.36 “Queue Priority” on page 282</td>
</tr>
<tr>
<td>Advance and standing reservations</td>
<td>cycle harvesting</td>
<td>See section 4.8.37 “Advance and Standing Reservations” on page 283</td>
</tr>
<tr>
<td>Round robin queue examination</td>
<td>by_queue</td>
<td>See section 4.8.38 “Round Robin Queue Selection” on page 290</td>
</tr>
<tr>
<td>Routing jobs</td>
<td></td>
<td>See section 4.8.39 “Routing Jobs” on page 292</td>
</tr>
<tr>
<td>Shared or exclusive vnodes and hosts</td>
<td></td>
<td>See section 4.8.40 “Shared vs. Exclusive Use of Resources by Jobs” on page 298</td>
</tr>
<tr>
<td>SMP cluster distribution</td>
<td>avoid_provision</td>
<td>See section 4.8.41 “SMP Cluster Distribution” on page 300</td>
</tr>
<tr>
<td>Sorting jobs using job_sort_key</td>
<td></td>
<td>See section 4.8.42 “Sorting Jobs on a Key” on page 302</td>
</tr>
<tr>
<td>Sorting jobs on job’s requested priority</td>
<td></td>
<td>See section 4.8.43 “Sorting Jobs by Requested Priority” on page 305</td>
</tr>
<tr>
<td>Sorting queues</td>
<td></td>
<td>See section 4.8.44 “Sorting Queues into Priority Order” on page 306</td>
</tr>
<tr>
<td>Starving jobs</td>
<td>fairshare</td>
<td>See section 4.8.45 “Starving Jobs” on page 307</td>
</tr>
<tr>
<td>Strict ordering</td>
<td>Backfilling combined with fairshare</td>
<td>See section 4.8.46 “Using Strict Ordering” on page 310</td>
</tr>
</tbody>
</table>
4.8.1 Anti-Express Queues

An anti-express queue is a preemptable low-priority queue, designed for jobs that should run only when no other jobs need the resources. These jobs are preempted if any other job needs the resources. An anti-express queue has the lowest priority of all queues in the complex. Jobs in this queue have a soft limit of zero, so that any job running from this queue is over its queue soft limit.

See section 4.8.33 “Using Preemption” on page 258.

4.8.1.1 Configuring Anti-express Queues

To configure an anti-express queue, do the following:

- Create an execution queue called lowprio:
  ```
  Qmgr: create queue lowprio
  Qmgr: set queue lowprio queue_type=e
  Qmgr: set queue lowprio started=true
  Qmgr: set queue lowprio enabled=true
  ```
- By default, all new queues have a priority of zero. Make sure all queues have a value set for priority, and that lowprio has the lowest priority:
  ```
  Qmgr: set queue workq priority=10
  ```
- Set the soft limit on the number of jobs that can run from that queue to

<table>
<thead>
<tr>
<th>Scheduling Tool</th>
<th>Incompatible Tools</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting vnodes on a key</td>
<td>smp_cluster_dist set to other than pack, or load balancing, with unused or assigned options to node_sort_key</td>
<td>See section 4.8.47 “Sorting Vnodes on a Key” on page 312</td>
</tr>
</tbody>
</table>

Table 4-4: List of Scheduling Tools
zero for all users:

```
Qmgr: set queue lowprio max_run_soft = "[u:PBS_GENERIC=0]"
```

- Make sure that jobs over their queue soft limits have lower preemption priority than normal jobs. Edit PBS_HOME/sched_priv/sched_config, and do the following:
  - Add “queue_softlimits” after “normal_jobs”. For example:
    ```
    preempt_prio: "express_queue, normal_jobs, queue_softlimits"
    ```
  - Sort queues:
    ```
    sort_queues: True ALL
    ```
  - Use preemption:
    ```
    preemptive_sched True ALL
    ```

### 4.8.1.2 Anti-express Queue Caveats

If you use soft limits on the number of jobs that users can run at other queues, jobs that are over their soft limits at other queues will also have the lowest preemption priority.

### 4.8.2 Associating Vnodes with Queues

You can associate each vnode with one or more queues. When a vnode is associated with a queue, that means it accepts jobs from that queue only. There are two arrangements:

- One or more vnodes associate with one queue
- One or more vnodes associate with multiple queues

These two arrangements require different methods of configuration.

You do not need to associate vnodes with queues in order to have jobs run on the vnodes that have the right application, as long as the application is a resource that can be requested by jobs. See section 5.14.7 “Supplying Application Licenses” on page 386.
4.8.2.1 Associating Vnodes With One Queue

You can associate one or more vnodes with a queue, using the vnode’s queue attribute. Using this method, each vnode can be associated with at most one queue. Each queue can be associated with more than one vnode. If you associate a queue and one or more vnodes using this method, any jobs in the queue can run only on the associated vnodes, and the only jobs that can run on the vnodes are the ones in the queue.

To associate a vnode with a queue, set the vnode’s queue attribute to the name of the queue you want. For example, to associate the vnode named Vnode1 with the queue named Queue1:

```
Qmgr: set node Vnode1 queue=Queue1
```

4.8.2.2 Associating Vnodes With Multiple Queues

You can use custom host-level resources to associate one or more vnodes with more than one queue. The scheduler will use the resources for scheduling just as it does with any resource.

In order to map a vnode to more than one queue, you must define a new host-level string array custom resource. This string array holds a string that has the same value for the queue and vnode you wish to associate. The mechanism of association is that a job that lands in the queue inherits that value for the resource, and then the job can run only on vnodes having a matching value for the resource. You can associate more than one queue with a vnode by setting the resource to the same value at each queue.

In some cases, you can use the same resource to route jobs and to associate vnodes with queues. For the method described here, you use host-level resources to associate vnodes with queues. The rules for which resources can be used for routing are given in section 2.2.6.4.2 “Resources Used for Routing and Admittance” on page 29. How jobs inherit resources is described in section 5.9.4 “Allocating Default Resources to Jobs” on page 342.
4.8.2.2.1 Procedure to Associate Vnodes with Multiple Queues

To associate one or more vnodes with one or more queues, do the following:

1. Define the new host-level resource. Add the new resource to $PBS_HOME/server_priv/resourcedef:
   
   `<new resource> type=string_array flag=h`

2. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

3. Instruct the scheduler to honor the resource. Add the new resource to $PBS_HOME/sched_priv/sched_config:
   
   `resources: "ncpus, mem, arch, host, vnode, <new resource>"`

4. HUP the scheduler:
   
   `kill -HUP <scheduler PID>`

5. Set each queue’s default_chunk for the new resource to the value you are using to associate it with vnodes:
   
   `Qmgr: set queue <queue name> default_chunk.<new resource> = <value>`

   For example, if one queue is “MathQ” and one queue is “SpareQ”, and the new resource is “Qlist”, and you want to associate a set of vnodes and queues based on ownership by the math department, you can make the queue resource value be “math”:

   `Qmgr: set queue MathQ default_chunk.Qlist = math`
   `Qmgr: set queue SpareQ default_chunk.Qlist = math`

6. Set the value for the new resource at each vnode:
   
   `Qmgr: set node <vnode name> resources_available.<new resource> = <associating value>`

   For example, to have the vnode named “Vnode1” associated with the queues owned by the math department:

   `Qmgr: set node Vnode1 resources_available.Qlist = math`
4.8.2.2 Example of Associating Multiple Vnodes with Multiple Queues

Now, as an example, assume you have 2 queues: “PhysicsQ” and “ChemQ”, and you have 3 vnodes: vn[1], vn[2], and vn[3]. You want Physics jobs to run on vn[1] and vn[2], and you want Chem jobs to run on vn[2] and vn[3]. Each department gets exclusive use of one vnode, but both must share a vnode.

To achieve the following mapping:

- PhysicsQ --> vn[1], vn[2]
- ChemQ --> vn[2], vn[3]
Which is the same as:

\[
\begin{align*}
\text{vn[1]} & \leftarrow \text{PhysicsQ} \\
\text{vn[2]} & \leftarrow \text{PhysicsQ, ChemQ} \\
\text{vn[3]} & \leftarrow \text{ChemQ}
\end{align*}
\]

1. Define the new host-level resource. Add the new resource to
   \texttt{$PBS\_HOME/server\_priv/resourcedef}:
   \begin{verbatim}
   Qlist type=string_array flag=h
   \end{verbatim}

2. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

3. Instruct the scheduler to honor the resource. Add the new resource to
   \texttt{$PBS\_HOME/sched\_priv/sched_config}:
   \begin{verbatim}
   resources: "..., Qlist"
   \end{verbatim}

4. HUP the scheduler:
   \begin{verbatim}
   kill -HUP <scheduler PID>
   \end{verbatim}

5. Add queue to vnode mappings:
   \begin{verbatim}
   Qmgr: s n vn[1] resources_available.Qlist="PhysicsQ"
   Qmgr: s n vn[2] resources_available.Qlist="PhysicsQ,ChemQ"
   Qmgr: s n vn[3] resources_available.Qlist="ChemQ"
   \end{verbatim}

6. Force jobs to request the correct Qlist values:
   \begin{verbatim}
   Qmgr: s q PhysicsQ default_chunk.Qlist=PhysicsQ
   Qmgr: s q ChemQ default_chunk.Qlist=ChemQ
   \end{verbatim}

### 4.8.3 Using Backfilling

*Backfilling* means fitting smaller jobs around the higher-priority jobs that
the scheduler is going to run next, in such a way that the higher-priority
jobs are not delayed.

Backfilling allows you to keep resources from becoming idle when the top
job cannot run.
Backfilling is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

### 4.8.3.1 Terminology

**Top job**  
Higher-priority (“most deserving”) job which is next to run according to execution priority. This job runs next when backfilling is not being used.

**Filler job**  
Smaller job that fits around top jobs. This job runs next only when backfilling is being used.

### 4.8.3.2 How Backfilling Works

The scheduler makes a list of jobs to run in order of priority. This list is composed according to execution priority described in section 4.8.16 “Calculating Job Execution Priority” on page 187.

If you use backfilling, the scheduler looks for smaller jobs that can fit into the usage gaps around the highest-priority jobs in the list. The scheduler looks in the prioritized list of jobs and chooses the highest-priority smaller jobs that fit. Filler jobs are run only if they will not delay the start time of top jobs.

The scheduler re-creates the list of highest-priority jobs at every scheduling cycle.

Backfilling is useful in the following circumstances:

- When the **strict_ordering** scheduler parameter is turned on, filler jobs are fitted around higher-priority jobs. Without backfilling, no job runs if the top job cannot run. See section 4.8.46 “Using Strict Ordering” on page 310
- When the **help_starving_jobs** scheduler parameter is turned on, filler jobs are fitted around starving jobs. See section 4.8.45 “Starving Jobs” on page 307
4.8.3.3 Backfilling Around N Jobs

You can configure the number of top jobs that PBS backfills around by setting the value of the `backfill_depth` server attribute. For example, if you set `backfill_depth` to 3, PBS backfills around the top 3 jobs. See “Server Attributes” on page 365 of the PBS Professional Reference Guide. Setting the `backfill_depth` parameter is effective only when `backfill` is set to `True`.

4.8.3.4 Configuring Backfilling

To configure backfilling, do the following:

1. Choose how many jobs to backfill around. If you want to backfill around more than 1 job, set the `backfill_depth` server attribute to the desired number. The default is 1. Set this parameter to less than 100.

2. Choose whether to use backfilling during primetime, non-primetime, or all of the time. If you want separate primetime and non-primetime behavior, specify the `backfill` parameter twice, once for each. The default is `all`.

3. Make sure that the `backfill` scheduler parameter is `True` for the time you want it. The default is `True`. For example:

   backfill True prime
   backfill False non_prime

4. Make sure that jobs request `walltime` by making them inherit a `walltime` resource if they don’t explicitly request it. For options, see section 4.8.3.8.1 “Ensure Jobs Are Eligible for Backfilling” on page 149.

4.8.3.5 Backfilling and Strict Ordering

When you use strict ordering, the scheduler runs jobs in exactly the order of their priority. If backfilling is turned off and the top job cannot run, no job is able to run. Backfilling can prevent resources from standing idle while the top job waits for its resources to become available. See section 4.8.46 “Using Strict Ordering” on page 310.
4.8.3.6 Backfilling and Starving Jobs

When you take starving jobs into consideration, by setting the help_starving_jobs scheduler parameter to True, starving jobs can become the top jobs. They can continue to wait for resources once they are the top job, blocking other jobs from running. Backfilling can prevent resources from standing idle while the top job waits for its resources to become available. See section 4.8.45 “Starving Jobs” on page 307.

4.8.3.7 Attributes and Parameters Affecting Backfilling

backfill
Scheduler parameter. Controls whether or not PBS uses backfilling. Scheduler will backfill when either strict_ordering is True or help_starving_jobs is True. See “backfill” on page 329 of the PBS Professional Reference Guide.

backfill_depth
Server attribute. Modifies backfilling behavior. Sets the number of jobs that are to be backfilled around. See “Server Attributes” on page 365 of the PBS Professional Reference Guide.

The configuration parameters backfill_prime and prime_exempt_anytime_queues do not relate to backfilling. They control the time boundaries of regular jobs with respect to primetime and non-primetime. See section 4.8.34 “Using Primetime and Holidays” on page 274.
4.8.3.8 Backfilling Recommendations and Caveats

4.8.3.8.1 Ensure Jobs Are Eligible for Backfilling

When calculating backfilling, PBS treats a job that has no walltime specified as if its walltime is 5 years. The scheduler will never use one of these jobs as a filler job. You can avoid this by ensuring that each job has a realistic walltime, by using the following methods:

- At qsub time via a hook
- By setting the queue’s `resources_default.walltime` attribute
- By setting the server’s `resources_default.walltime` attribute
- At qsub time via the server’s `default_qsub_arguments`

4.8.3.8.2 Number of Jobs to Backfill Around

The more jobs being backfilled around, the longer the scheduling cycle takes.

4.8.3.8.3 Dynamic Resources and Backfilling

Using dynamic resources and backfilling may result in some jobs not being run even though resources are available. This may happen when a job requesting a dynamic resource is selected as the top job. The scheduler must estimate when resources will become available, but it can only query for resources available at the time of the query, not resources already in use, so it will not be able to predict when resources in use become available. Therefore the scheduler won’t be able to schedule the job. In addition, since dynamic resources are outside of the control of PBS, they may be consumed between the time the scheduler queries for the resource and the time it starts a job.

4.8.3.8.4 Avoid Using Strict Ordering, Backfilling, and Fairshare

It is inadvisable to use strict ordering and backfilling with fairshare.

The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group will be chosen as the small job. The usage from these small jobs will lower the priority of the most deserving job.
For example, if a user has a large job that is the most deserving but cannot run, smaller jobs owned by that user will chew up the user's usage, and prevent the large job from ever being likely to run. Also, if the small jobs are owned by a user in one area of the fairshare tree, no large jobs owned by anyone else in that section of the fairshare tree are likely to be able to run.

### 4.8.3.8.5 Avoid Using Preemption, Strict Ordering, and Backfilling

Using preemption with strict ordering and backfilling may change the top job.

### 4.8.3.8.6 Warning About Backfilling and Provisioning

The scheduler will not run a job requesting an AOE on a vnode that has a top job scheduled on it in the future.

The scheduler will not use a job requesting an AOE as a top job.

### 4.8.3.8.7 Jobs That Cannot Run on Current Resources

The scheduler checks to see whether each job could possibly run now, counting resources as if there were no other jobs, and all current resources could be used by this job. The scheduler counts resources only from those vnodes that are on line. If a vnode is marked offline, its resources are not counted.

When the scheduler is using backfilling, if the top job cannot run on current resources, the following error message is printed in the scheduler's log file:

```
"resource request is impossible to solve: job will never run"
```

The scheduler only determines whether or not a job can run on current resources when backfilling is used. If backfilling is turned off, then the scheduler won't determine this. It just decides it can't run now.

The scheduler logs this message for the top job only.

### 4.8.3.8.8 Backfilling and Estimating Job Start Time

When the scheduler is backfilling around jobs, it estimates the start times and `exec_vnode` for the top jobs being backfilled around. See section 4.8.15 “Estimating Job Start Time” on page 181.
4.8.3.8.9 Using Strict Ordering and Backfilling with Only One of Primetime or Non-primetime

When PBS is using strict ordering and backfilling, the scheduler saves a spot for each high-priority job around which it is backfilling. If you configure PBS to use strict ordering and backfilling for only one of primetime or non-primetime, and you have large jobs that must wait a long time before enough resources are available, the saved spots can be lost in the transition.

4.8.4 Examining Jobs Queue by Queue

When the scheduler examines waiting jobs, it can either consider all of the jobs in the complex as a whole, or it can consider jobs queue by queue. When considering jobs queue by queue, the scheduler runs all the jobs it can from the first queue before examining the jobs in the next queue, and so on. This behavior is controlled by the by_queue scheduler parameter.

When the by_queue scheduler parameter is set to True, jobs in the first queue are evaluated as a group, then jobs in the next queue are evaluated. In this case, PBS runs all the jobs it can from each queue before moving to the next queue, with the following exception: if there are jobs in the Reservation, Express, Starving, or Suspended job execution classes, those are considered before any queue. These classes are described in section 4.8.16 “Calculating Job Execution Priority” on page 187.

The by_queue parameter applies to all of the queues in the complex. This means that either all jobs are scheduled as if they are in one large queue, or jobs are scheduled queue by queue.

If you want jobs to be examined queue by queue, in order of queue priority, you must also sort the queues and specify a different priority for each queue. To sort queues, set the sort_queues scheduler parameter to True. To set queue priority, set each queue’s priority attribute to the desired value. A queue with a higher value is examined before a queue with a lower value. If you do not assign priorities to queues, their ordering is undefined. See “sort_queues” on page 343 of the PBS Professional Reference Guide, and “Queue Attributes” on page 407 of the PBS Professional Reference Guide.
The by_queue parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

See “by_queue” on page 330 of the PBS Professional Reference Guide.

### 4.8.4.1 Configuring PBS to Consider Jobs Queue by Queue

- Set the by_queue scheduler parameter to True
- Set the sort_queues scheduler parameter to True
- Assign a priority to each queue
- Choose whether you want queue by queue during primetime, non-primetime, or both. If you want separate behavior for primetime and non-primetime, list by_queue twice. For example:
  
  ```
  by_queue True prime
  by_queue False non_prime
  ```

### 4.8.4.2 Parameters and Attributes Affecting Queue by Queue

- The by_queue scheduler parameter; see “by_queue” on page 330 of the PBS Professional Reference Guide.
- The sort_queues scheduler parameter; see “sort_queues” on page 343 of the PBS Professional Reference Guide.
- The priority queue attribute; see “Queue Attributes” on page 407 of the PBS Professional Reference Guide.

### 4.8.4.3 Caveats and Advice for Queue by Queue

- The by_queue scheduler parameter is overridden by the round_robin scheduler parameter when round_robin is set to True.
- When by_queue is True, queues cannot be designated as fairshare entities, and fairshare will work queue by queue instead of on all jobs at once.
- When by_queue is True, job execution priority may be affected. See
section 4.8.16 “Calculating Job Execution Priority” on page 187.

- The by_queue parameter is not required when using express queues.
- You can have FIFO scheduling for all your jobs across the complex, if you are using a single execution queue or have by_queue set to False. However, you can have FIFO scheduling for the jobs within each queue if you set by_queue to True and specify a different priority for each queue. See section 4.8.19 “FIFO Scheduling” on page 203.

### 4.8.5 Checkpointing

You can use checkpointing as a scheduling tool, by including it as a preemption method, an aid in recovery, and when using the qhold command.

For a complete description of how to use and configure checkpointing, see section 10.3 “Checkpoint and Restart” on page 735.

#### 4.8.5.1 Checkpointing as a Preemption Method

When a job is preempted via checkpointing, MOM runs the checkpoint_abort script, and PBS kills and requeues the job. When the scheduler elects to run the job again, the MOM runs the restart script to restart the job from where it was checkpointed. See section 4.8.33 “Using Preemption” on page 258.

#### 4.8.5.2 Checkpointing When Using the qhold Command

When the qhold command is used to hold a checkpointable job, MOM runs the checkpoint_abort script, and PBS kills, requeues, and holds the job. A job with a hold on it must have the hold released via the qrls command in order to be eligible to run. For a discussion of the use of checkpointing for the qhold command, see section 10.3.7.6 “Holding a Job” on page 759. For a description of the qhold command, see “qhold” on page 175 of the PBS Professional Reference Guide.
4.8.6 Organizing Job Chunks

You can specify how job chunks should be organized onto hosts or vnodes. Jobs can request their placement arrangement, and you can set defaults at queues and at the server to be inherited by jobs that do not request a placement. You can tell PBS to do the following:

• Put all chunks from a job onto a single host using the `place=pack` statement.

• Put each chunk on a separate host using the `place=scatter` statement. The number of chunks must be fewer than or equal to the number of hosts.

• Put each chunk on a separate vnode using the `place=vscatter` statement. The number of chunks must be fewer than or equal to the number of vnodes.

• Put each chunk anywhere using the `place=free` statement.

To specify a placement default, set `resources_default.place=<arrangement>`, where `arrangement` is `pack`, `scatter`, `vscatter`, or `free`. For example, to have the default at QueueA be `pack`:

```
Qmgr: set queue QueueA
    resources_default.place=pack
```

You can specify that job chunks must be grouped in a certain way. For example, to require that chunks all end up on a shared router, use this:

`place=group=router`

For more about jobs requesting placement, see “Requesting Resources and Placing Jobs” on page 248 of the PBS Professional Reference Guide.

4.8.6.1 Caveats for Organizing Job Chunks

A placement specification for arrangement, sharing, and grouping is treated as one package by PBS. This means that if a job requests only one, any defaults set for the others are not inherited. For example, if you set a default of `place=pack:excl:group=router`, and a job requests only `place=pack`, the job does not inherit `excl` or `group=router`. 
4.8.7  **cron** Jobs, or the Windows Task Scheduler

You can use **cron** jobs or the Windows Task Scheduler to make time-dependent modifications to settings, where you are scheduling according to time slots. For example, you can change settings for primetime and non-primetime configurations, making the following changes:

- Set nodes *offline* or not *offline*
- Change the number of *ncpus* on workstations
- Change the priority of queues, for example to change preemption behavior
- Start or stop queues
- Set primetime & non-primetime options

**4.8.7.1 Caveats for cron Jobs and the Windows Task Scheduler**

Make sure that your **cron** jobs or Windows Task Scheduler behave correctly when PBS is not running.

Be careful when changing available resources, such as when offlining vnodes. You might prevent jobs from running that would otherwise run. For details, see section 4.6.2 “Jobs that Cannot Run on Current Resources” on page 134.

**4.8.8 Using Custom and Default Resources**

The information in this section relies on understanding how jobs are allocated resources via inheriting defaults or via hooks. Before reading this section, please read section 12.2 “Allocating Resources to Jobs” on page 842.

For complete details of how to configure and use custom resources, please see section 5.14 “Custom Resources” on page 352.
You can use custom and default resources for several purposes:

- Routing jobs to the desired vnodes; see section 4.8.8.2 “Using Custom Resources to Route Jobs” on page 156
- Assigning execution priority to jobs; see section 4.8.8.3 “Using Custom Resources to Assign Job Execution Priority” on page 157
- Tracking and controlling the allocation of resources; see section 4.8.8.4 “Using Custom Resources to Track and Control Resource Allocation” on page 158
- Representing elements such as GPUs, FPGAs, and switches; see section 4.8.8.5 “Using Custom Resources to Represent GPUs, FPGAs, Switches, Etc.” on page 158
- Allowing users to request platform-specific resources, for example Cray-specific resources; see section 4.8.8.6 “Using Custom Resources to Allow Platform-specific Resource Requests” on page 158
- Allowing users to submit jobs that run on a Cray as they would if using the aprun command; see section 4.8.8.7 “Using Custom Resources to Allow Platform-specific Behavior” on page 159

4.8.8.1 Techniques for Allocating Custom Resources to Jobs

In addition to using custom resources to represent physical elements such as GPUs, you can use custom resources as tags that you attach to jobs in order to help schedule the jobs. You can make these custom resources into tools that can be used only for managing jobs, by making them unalterable and unrequestable, and if desired, invisible to users.

For how to assign custom and default resources to jobs, see section 12.2 “Allocating Resources to Jobs” on page 842.

4.8.8.2 Using Custom Resources to Route Jobs

You can use several techniques to route jobs to the desired queues and/or vnodes. Depending on your site’s configuration, you may find it helpful to use custom resources with one or more of these techniques.

- You can force users to submit jobs to the desired queues by setting resource limits at queues. You can use custom resources to represent
arbitrary elements, for example, projects. In this case you could limit the project at each queue. You can set a default value for the project at the server, or create a hook that assigns a value for the project.

For how queue resource limits are applied to jobs, see section 2.2.6.4.1 “How Queue Limits Are Applied” on page 28

- Upon submission, send jobs to routing queues, then route to other queues inside or outside the PBS complex. Again, custom resources can represent arbitrary elements, and can be inherited by the job or assigned by a hook.

For how routing queues work, see section 2.2.6 “Routing Queues” on page 26

- Use peer scheduling to send jobs between PBS complexes. You can set resource limits on the furnishing queue in order to limit the kinds of jobs that are peer scheduled. You can assign custom resources to jobs to represent arbitrary elements, for example peer queueing only those jobs from a specific project. You can assign the custom resource by having the job inherit it or via a hook.

For how to set up peer scheduling, see section 4.8.31 “Peer Scheduling” on page 229

- You can route jobs from specific queues to the desired vnodes, by associating the vnodes with the queues. See section 4.8.2 “Associating Vnodes with Queues” on page 141.

- You can create placement sets so that jobs are placed according to resource values. Placement sets are created where vnodes share a value for a resource; you can use custom resources to create the placement sets you want. See section 4.8.32 “Placement Sets” on page 237.

### 4.8.8.3 Using Custom Resources to Assign Job Execution Priority

You can use custom resources as coefficients in the job sorting formula. You can assign custom resources to jobs using the techniques listed in section 12.2 “Allocating Resources to Jobs” on page 842. The value of each custom resource can be based on a project, an application, etc.
For example, you can create a custom resource called “ProjPrio”, and the jobs that request the “Bio” project can be given a value of 5 for ProjPrio, and the jobs that request the “Gravel” project can be given a value of 2 for ProjPrio. You can assign this value in a hook or by routing the jobs through special queues from which the jobs inherit the value for ProjPrio.

For information on using the job sorting formula, see section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

### 4.8.8.4 Using Custom Resources to Track and Control Resource Allocation

You can use resources to track and control usage of things like hardware and licenses. For example, you might want to limit the number of jobs using floating licenses or a particular vnode. See section 5.10 “Using Resources to Track and Control Allocation” on page 347.

### 4.8.8.5 Using Custom Resources to Represent GPUs, FPGAs, Switches, Etc.

You can use custom resources to represent GPUs, FPGAs, high performance switches, etc. For examples, see section 5.14.8 “Using GPUs” on page 402, section 5.14.9 “Using FPGAs” on page 406, and section 11.1.7.3 “Using IBM’s poe” on page 791.

### 4.8.8.6 Using Custom Resources to Allow Platform-specific Resource Requests

PBS is integrated with Cray, and provides special custom resources to represent Cray resources. You can create other custom resources to represent other platform-specific elements. For an example, see section 11.5.7.13 “Allowing Users to Request Login Node Groups” on page 817.
4.8.8.7 Using Custom Resources to Allow Platform-specific Behavior

You can create custom resources that allow Cray users to run jobs that behave the same way they would if the user had used the aprun command. For examples, see section 11.5.7.11 “Allowing Users To Reserve NUMA Nodes Per Compute Node” on page 812 and section 11.5.7.12 “Allowing Users To Reserve Specific NUMA Nodes” on page 814.

4.8.9 Using Idle Workstation Cycle Harvesting

You can configure workstations at your site so that PBS can run jobs on them when their “owners” are away and they are idle. This is called idle workstation cycle harvesting. This can give your site additional resources to run jobs during nights and weekends.

You can configure PBS to use the following methods to decide when a workstation is not being used by its owner:

- Keyboard/mouse activity
- X-Window monitoring
- Load average (not recommended)

On some systems cycle harvesting is simple to implement, because the console, keyboard, and mouse device access times are periodically updated by the operating system. The PBS MOM process can track this information, and mark the vnode busy if any of the input devices is in use. On other systems, however, this data is not available. On these machines, PBS can monitor the X-Window system in order to obtain interactive idle time.

Jobs on workstations that become busy are not migrated; they remain on the workstation until they complete execution, are rerun, or are deleted.
4.8.9.1 Platforms Supporting Cycle Harvesting

Due to different operating system support for tracking mouse and keyboard activity, the availability and method of support for cycle harvesting varies based on the computer platform in question. The following table lists the method and support for each platform.

**Table 4-5: Cycle Harvesting Support Methods**

<table>
<thead>
<tr>
<th>System</th>
<th>Status</th>
<th>Method</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>supported</td>
<td>pbs_idle</td>
<td>“Cycle Harvesting by Monitoring X-Windows” on page 163.</td>
</tr>
<tr>
<td>HP-UX 11</td>
<td>supported</td>
<td>keyboard/mouse</td>
<td>section 4.8.9.3 “Cycle Harvesting Based on Keyboard/Mouse Activity” on page 161</td>
</tr>
<tr>
<td>Linux</td>
<td>supported</td>
<td>keyboard/mouse</td>
<td>section 4.8.9.3 “Cycle Harvesting Based on Keyboard/Mouse Activity” on page 161</td>
</tr>
<tr>
<td>Solaris</td>
<td>supported</td>
<td>keyboard/mouse</td>
<td>section 4.8.9.3 “Cycle Harvesting Based on Keyboard/Mouse Activity” on page 161</td>
</tr>
<tr>
<td>Windows Professional class</td>
<td>supported</td>
<td>operating system</td>
<td>section 4.8.9.3 “Cycle Harvesting Based on Keyboard/Mouse Activity” on page 161</td>
</tr>
<tr>
<td>Windows Server class</td>
<td>supported</td>
<td>operating system</td>
<td>section 4.8.9.3 “Cycle Harvesting Based on Keyboard/Mouse Activity” on page 161</td>
</tr>
</tbody>
</table>

4.8.9.2 The $kbd_idle MOM Configuration Parameter

Cycle harvesting based on keyboard/mouse activity and X-Windows monitoring is controlled by the $kbd_idle MOM configuration parameter in PBS_HOME/mom_priv/config on the workstation in question. This parameter has the following format:

```
$kbd_idle <idle_wait> <min_use> <poll_interval>
```

Declares that the vnode will be used for batch jobs during periods when the keyboard and mouse are not in use. Not supported in the HPCBP MOM.

```
idle_wait
```
Scheduling

Time, in seconds, that the workstation keyboard and mouse must be idle before being considered available for batch jobs.

Must be set to value greater than 0 for cycle harvesting to be enabled.

Format: Integer

No default

\[ \text{min\_use} \]

Time, in seconds, during which the workstation keyboard or mouse must continue to be in use before the workstation is determined to be unavailable for batch jobs.

Format: Integer

Default: 10

\[ \text{poll\_interval} \]

Interval, in seconds, at which MOM checks for keyboard and mouse activity.

Format: Integer

Default: 1

### 4.8.9.3 Cycle Harvesting Based on Keyboard/Mouse Activity

PBS can monitor a workstation for keyboard and mouse activity, and run batch jobs on the workstation when the keyboard and mouse are not being used. PBS sets the state of the vnode to either \textit{free} or \textit{busy}, depending on whether or not there is keyboard or mouse activity, and runs jobs only when the state of the vnode is \textit{free}. PBS sets the state of the vnode to \textit{free} when the vnode’s mouse and keyboard have shown no activity for the specified amount of time. If PBS determines that the vnode is being used, it sets the state of the vnode to \textit{busy} and suspends any running jobs, setting their state to \textit{U (user busy)}. 
4.8.9.3.1 Configuring Cycle Harvesting Using Keyboard/Mouse Activity

To configure cycle harvesting using keyboard and mouse activity, do the following:

1. Set the $kbd_idle MOM configuration parameter by editing the $kbd_idle parameter in PBS_HOME/mom_priv/config on the workstation.

2. HUP the MOM on the workstation:

   ```
   kill -HUP <pbs_mom PID>
   ```

4.8.9.3.2 Example of Cycle Harvesting Using Keyboard/Mouse Activity

The following is an example setting for the parameter:

   ```
   $kbd_idle 1800 10 5
   ```

This setting for the parameter in MOM’s config file specifies the following:

- PBS marks the workstation as free if the keyboard and mouse are idle for 30 minutes (1800 seconds)
- PBS marks the workstation as busy if the keyboard or mouse is used for 10 consecutive seconds
- The states of the keyboard and mouse are to be checked for activity every 5 seconds

Here, we walk through how this example would play out, to show the roles of the arguments to the $kbd_idle parameter:

Let’s start with a workstation that has been in use for some time by its owner. The workstation is in state busy.

Now the owner goes to lunch. After 1800 seconds (30 minutes), PBS changes the workstation’s state to free and starts a job on the workstation.

Some time later, someone walks by and moves the mouse or enters a command. Within the next 5 seconds (idle poll period), pbs_mom...
notes the activity. The job is suspended and placed in state \( U \), and the workstation is marked \( busy \).

If 10 seconds pass and there is no additional keyboard/mouse activity, the job is resumed and the workstation again is either \( free \) (if any CPUs are available) or \( job-busy \) (if all CPUs are in use.)

However, if keyboard/mouse activity continues during that 10 seconds, the workstation remains \( busy \) and the job remains suspended for at least the next 1800 seconds.

4.8.9.3.3 Caveats for Cycle Harvesting Using Keyboard/Mouse Activity

- There is no default for \( idle\_wait \); you must set it to a value greater than 0 in order to enable cycle harvesting using keyboard/mouse activity.
- Cycle harvesting using keyboard/mouse activity is not supported on HPCBP MOM machines.

4.8.9.4 Cycle Harvesting by Monitoring X-Windows

On machines where the OS does not periodically update console, keyboard, and mouse device access times, PBS can monitor X-Window activity instead. PBS uses an X-Window monitoring process called \( pbs\_idled \). This process runs in the background and monitors X and reports to the \( pbs\_mom \) whether or not the vnode is idle. \( pbs\_idled \) is located in \$PBS\_EXEC/sbin.

To configure PBS for cycle harvesting by monitoring X-Windows, perform the following steps:

1. Create a directory for \( pbs\_idled \). This directory must have the same permissions as \( /tmp \) (i.e. mode 1777). This will allow the
pbs_idled program to create and update files as the user, which is necessary because the program runs as the user. For example:

```
mkdir PBS_HOME/spool/idledir
chmod 1777 PBS_HOME/spool/idledir
```

2. Turn on keyboard idle detection in the MOM config file:

```
$kbd_idle <idle wait value>
```

3. Include pbs_idled as part of the X-Windows startup sequence.

   The best and most secure method of starting pbs_idled is via the system-wide Xsession file. This is the script which is run by xdm (the X login program) and sets up each user's X-Windows environment.

   You **must** place the startup line for pbs_idled before that of the window manager.

   You **must** make sure that pbs_idled runs in the background.

   On systems that use Xsession to start desktop sessions, insert a line invoking pbs_idled near the top of the file.

   For example, insert the following line in a Linux Xsession file:

   ```
   /usr/pbs/sbin/pbs_idled &
   ```

   If access to the system-wide Xsession file is not available, you can add pbs_idled to every user's personal .xsession or .xinitrc file, depending on the local OS requirements for starting X-windows programs upon login.

4.8.9.5 Cycle Harvesting Based on Load Average

Cycle harvesting based on load average means that PBS monitors each workstation’s load average, runs jobs where workstations have loads below a specified level, and suspends any batch jobs on workstations whose load has risen above the limit you set. When a workstation’s owner uses the machine, the workstation’s load rises.

When you configure cycle harvesting based on load average, you are performing the same configuration as for load balancing using load average. For a complete description of load balancing, see section 4.8.27 “Using Load Balancing” on page 216.
4.8.9.5.1 Attributes and Parameters Affecting Cycle Harvesting Based on Load Average

load_balancing
Scheduler parameter. When set to True, the scheduler places jobs only where the load average is below the specified limit.

Format: Boolean
Default: False all

$ideal_load <load>
MOM parameter. Defines the load below which the vnode is not considered to be busy. Used with the $max_load directive. Not supported in the HPCBP MOM.

Example:
$ideal_load 1.8

Format: Float
No default

$max_load <load> [suspend]
MOM parameter. Defines the load above which the vnode is considered to be busy. Used with the $ideal_load directive. No new jobs are started on a busy vnode.

The optional suspend directive tells PBS to suspend jobs running on the node if the load average exceeds the $max_load number, regardless of the source of the load (PBS and/or logged-in users). Without this directive, PBS will not suspend jobs due to load.

We recommend setting this to a slightly higher value than your target load (which is typically the number of CPUs), for example .25 + ncpus.

Not supported in the HPCBP MOM.

Example:
$max_load 3.25

Format: Float
Default: number of CPUs
**resv_enable**  
Vnode attribute. Controls whether the vnode can be used for advance and standing reservations. When set to `True`, this vnode can be used for reservations.

Format: *Boolean*  
Default: `True`

**no_multinode_jobs**  
Vnode attribute. Controls whether jobs which request more than one chunk are allowed to execute on this vnode. When set to `True`, jobs requesting more than one chunk are not allowed to execute on this vnode.

Format: *Boolean*  
Default: `False`

### 4.8.9.5.2 How Cycle Harvesting Based on Load Average Works

Cycle harvesting based on load average means that PBS monitors the load average on each machine. When the load on a workstation is below what is specified in the `$ideal_load` MOM parameter, PBS sets the state of the workstation to `free`. The scheduler will run jobs on vnodes whose state is `free`. When the load on a workstation exceeds the setting for `$max_load`, PBS sets the state of the workstation to `busy`, and suspends jobs running on the workstation. PBS does not start jobs on a vnode whose state is `busy`. When the load drops below the setting for `$ideal_load`, PBS sets the state to `free`, and resumes the jobs that were running on the workstation.

PBS thinks that a 1-CPU job raises a vnode’s load by 1. On machines being used for cycle harvesting, values for `$max_load` and `$ideal_load` are set to reasonable limits. On other machines, these are set to values that will never be exceeded, so that load is effectively ignored.

On machines where these parameters are unset, the vnode’s state is not set according to its load, so jobs are not suspended because a vnode is busy. However, if `$max_load` and `$ideal_load` are unset, they are treated as if they have the same value as `resources_available.ncpus`, and because there is usually a small background load, PBS will lose the use of a CPU’s worth of load. The scheduler won’t place a job where the anticipated load would exceed `$max_load`, so if a machine has a load of 1.25, is running a 1-CPU job, and has 2 CPUs, PBS won’t place another 1-CPU job there.
4.8.9.5.3 Configuring Cycle Harvesting Based on Load Average

To set up cycle harvesting for idle workstations based on load average, perform the following steps:

1. If PBS is not already installed on the target execution workstations, do so now, selecting the execution-only install option. See the PBS Professional Installation & Upgrade Guide.

2. Edit the PBS_HOME/mom_priv/config configuration file on each target execution workstation, adding the $max_load and $ideal_load configuration parameters. Make sure they have values that will not interfere with proper operation. See section 4.8.9.5.5 “Caveats for Cycle Harvesting Based on Load Average” on page 168.

   $max_load <load limit that allows jobs to run>
   $ideal_load <load at which to start jobs>

3. Edit the PBS_HOME/mom_priv/config configuration file on each machine where you are not using cycle harvesting, adding the $max_load and $ideal_load configuration parameters. Make sure they have values that will never be exceeded.

   $max_load <load limit that will never be exceeded>
   $ideal_load <load limit that will never be exceeded>

4. HUP the MOM:

   kill -HUP <pbs_mom PID>

5. Edit the PBS_HOME/sched_priv/sched_config configuration file to direct the Scheduler to perform scheduling based on load_balancing.

   load_balancing: True   ALL

6. If you wish to oversubscribe the vnode’s CPU(s), set its resources_available.ncpus to a higher number. Do this only on single-vnode machines. You must be cautious about matching ncpus and $max_load. See section 4.8.9.5.5, "Caveats for Cycle Harvesting
Based on Load Average" on page 168 in the PBS Professional Administrator’s Guide.

7. HUP the scheduler:

```
kill -HUP <pbs_sched PID>
```

8. Set the vnode’s resv_enable attribute to False, to prevent the workstation from being used for reservations.

```
Qmgr: set node <vnode name> resv_enable = False
```

9. Set the vnode’s no_multinode_jobs attribute to True, to prevent the workstation from stalling multichunk jobs.

```
Qmgr: set node <vnode name> no_multinode_jobs = True
```

4.8.9.5.4 Viewing Load Average Information

You can see the state of a vnode using the `pbsnodes -a` command.

4.8.9.5.5 Caveats for Cycle Harvesting Based on Load Average

- Be careful with the settings for $ideal_load and $max_load. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MOM to report being busy, and that PBS does not start any new jobs while the user is working.

- For information about keeping your site running smoothly using $max_load and $ideal_load, see section 10.4.5 “Managing Load Levels on Vnodes” on page 768

- If you set ncpus higher than the number of actual CPUs, and set $max_load higher to match, keep in mind that the workstation user could end up with an annoyingly slow workstation. This can happen when PBS runs jobs on the machine, but the combined load from the jobs and the user is insufficient for MOM to report being busy.

4.8.9.6 Cycle Harvesting and File Transfers

The cycle harvesting feature interacts with file transfers in one of two different ways, depending on the method of file transfer:

- If the user’s job includes file transfer commands (such as rcp or scp)
within the job script, and such a command is running when PBS
decides to suspend the job on the vnode, then the file transfer is sus-
pended as well.

• If the job has PBS file staging parameters (i.e. stagein=, stage-
out=file1...), and the load goes above $max_load, the file transfer is
not suspended. This is because the file staging is not part of the job
script execution, and is not subject to suspension. See section 7.6.7,
"Detailed Description of Job’s Lifecycle", on page 176 of the PBS Pro-
fessional User’s Guide.

4.8.9.7 Cycle Harvesting on Windows

On Windows, configure cycle harvesting using keyboard/mouse activity.

Under Windows, when a machine becomes busy because the keyboard is
being used, the effect on the job is different. Instead of being suspended,
the job has its priority lowered from Normal to Low. For example, you
submit a job and it begins to run on a workstation, and the CPU loading on
that machine goes to 100%. Then you move the mouse, and you see that
the CPU loading is still 100%. This is because the job has lower priority,
but is not suspended. If you use qstat, you’ll see that the job’s state is U,
because PBS has marked the job as suspended. Local activity on the
machine will have higher priority.

4.8.9.8 Parallel Jobs With Cycle Harvesting

Cycle harvesting is not recommended for hosts that will run multi-host
jobs. However, you may find that your site benefits from using cycle har-
vesting on these machines. We provide advice on how to prevent cycle
harvesting on these machines, and advice on how to accomplish it.

4.8.9.8.1 General Advice: Parallel Jobs Not Recommended

Cycle harvesting is somewhat incompatible with multi-host jobs. If one of
the hosts being used for a parallel job running on several hosts is being
used for cycle harvesting, and the user types at the keyboard, job execution
will be delayed for the entire job because the tasks running on that host will
be suspended.
To prevent a machine which is being used for cycle harvesting from being assigned a multi-host job, set the vnode’s no_multinode_jobs attribute to True. This attribute prevents a host from being used by jobs that span multiple hosts.

### 4.8.9.8.2 How to Use Cycle Harvesting with Multi-host Jobs

When a single-host job is running on a workstation configured for cycle harvesting, and that host becomes busy, the job is suspended. However, suspending a multi-host parallel job may have undesirable side effects because of inter-process communications. For a job which uses multiple hosts when one or more of the hosts becomes busy, the default action is to leave the job running.

However, you can specify that the job should be requeued and subsequently re-scheduled to run elsewhere when any of the hosts on which the job is running becomes busy. To enable this action, add the following parameter to MOM’s configuration file:

```
$action multinodebusy 0 requeue
```

where `multinodebusy` is the action to modify; “0” (zero) is the action timeout value (it is ignored for this action); and `requeue` is the new action to perform. The only action that can be performed is requeueing.

Multi-host jobs which are not rerunnable (i.e. those submitted with the `qsub -rn` option) will be killed if the `requeue` argument is configured for the `multinodebusy` action and a vnode becomes busy.

### 4.8.9.9 Cycle Harvesting Caveats and Restrictions

#### 4.8.9.9.1 Cycle Harvesting and Multi-host Jobs

Cycle harvesting is not recommended for hosts that will run multi-host jobs. See section 4.8.9.8.1 “General Advice: Parallel Jobs Not Recommended” on page 169.
4.8.9.9.2 Cycle Harvesting and Reservations

Cycle harvesting is incompatible with jobs in reservations. Reservations should not be made on a machine used for cycle harvesting, because the user may appear during the reservation period and use the machine’s keyboard. This will suspend the jobs in the reservation, defeating the purpose of making a reservation.

To prevent a vnode which is being used for cycle harvesting from being used for reservations, set the vnode’s resv_enable attribute to False. This attribute controls whether the vnode can be used for reservations.

4.8.9.9.3 File Transfers with Cycle Harvesting

File transfers behave differently depending on job details. See section 4.8.9.6 “Cycle Harvesting and File Transfers” on page 168.

4.8.9.9.4 Cycle Harvesting on Windows

Jobs are not suspended under Windows when the workstation becomes busy. See section 4.8.9.7 “Cycle Harvesting on Windows” on page 169.

4.8.10 Dedicated Time

PBS provides a feature called dedicated time which allows you to define times during which the only jobs that can run are the ones in dedicated queues. You can use dedicated time for things like upgrades.

You can define multiple dedicated times. Any job in a dedicated time queue must have a walltime in order to run. Jobs without walltimes will never run. PBS won’t let a reservation conflict with dedicated time. Hooks should not access or modify the dedicated time file.

For information on configuring dedicated time queues, see section 2.2.5.2.1 “Dedicated Time Queues” on page 25.
4.8.10.1 Dedicated Time File

You define dedicated time by adding one or more time slots in the file PBS_HOME/sched_priv/dedicated_time. A time slot is a start date and start time and an end date and end time. Format:

\[<\text{start date}> \ <\text{start time}> \ <\text{end date}> \ <\text{end time}>\]

expressed as

\[\text{MM/DD/YYYY} \ \text{HH:MM} \ \text{MM/DD/YYYY} \ \text{HH:MM}\]

Any line whose first non-whitespace character is a pound sign ("#") is a comment.

Example:

\[
\#\text{Dedicated time for maintenance} \\
04/15/2007 \ 12:00 \ 04/15/2007 \ 15:30
\]

A sample dedicated time file (PBS_HOME/sched_priv/dedicated_time) is included in the installation.

The dedicated time file is read on startup and HUP.

4.8.10.2 Steps in Defining Dedicated Time

You define dedicated time by performing the following steps:

1. Edit the file PBS_HOME/sched_priv/dedicated_time and add one or more time slots.

2. HUP the scheduler:

   UNIX/Linux:
   ```
   kill -HUP <pbs_sched PID>
   ```

   Windows:
   ```
   net stop pbs_sched
   net start pbs_sched
   ```
4.8.11 Dependencies

PBS allows job submitters to specify dependencies between jobs, for example specifying that job J2 can only run if job J1 finishes successfully. You can add dependencies to jobs via a hook, default arguments to `qsub`, or via the `qalter` command.

For a description of how job dependencies work, see section 7.4, "Specifying Job Dependencies", on page 161 of the PBS Professional User’s Guide.

For how to use hooks, see section “Hooks” on page 461.

For how to add default `qsub` arguments, see “Server Attributes” on page 365 of the PBS Professional Reference Guide.

For how to use the `qalter` command, see “qalter” on page 152 of the PBS Professional Reference Guide.

4.8.12 Dynamic Resources

You can use dynamic PBS resources to represent elements that are outside of the control of PBS, typically for licenses and scratch space. You can represent elements that are available to the entire PBS complex as server-level resources, or elements that are available at a specific host or hosts as host-level resources. For an example of configuring a server-level dynamic resource, see section 5.14.4.1.1 “Example of Configuring Dynamic Server-level Resource” on page 371. For an example of configuring a dynamic host-level resource, see section 5.14.5.1.1 “Example of Configuring Dynamic Host-level Resource” on page 376.

For a complete description of how to create and use dynamic resources, see section 5.14 “Custom Resources” on page 352.

4.8.13 Eligible Wait Time for Jobs

PBS provides a method for tracking how long a job that is eligible to run has been waiting to run. By “eligible to run”, we mean that the job could run if the required resources were available. The time that a job waits while it is not running can be classified as “eligible” or “ineligible”. Roughly speaking, a job accrues eligible wait time when it is blocked due
to a resource shortage, and accrues ineligible wait time when it is blocked due to user or group limits. A job can only accrue one kind of wait time at a time, and cannot accrue wait time while it is running.

### 4.8.13.1 Types of Time Accrued

**eligible_time**

Job attribute. The amount of wall clock wait time a job has accrued because the job is blocked waiting for resources, or any other reason not covered by ineligible_time. For a job currently accruing eligible_time, if we were to add enough of the right type of resources, the job would start immediately. Viewable via qstat -f by job owner, Manager and Operator. Settable by Operator or Manager.

**ineligible_time**

The amount of wall clock time a job has accrued because the job is blocked by limits on the job’s owner or group, or because the job is blocked because of its state.

**run_time**

The amount of wall clock time a job has spent running.

**exiting**

The amount of wall clock time a job has spent exiting.

**initial_time**

The amount of wall clock wait time a job has accrued before the type of wait time has been determined.

### 4.8.13.2 How Eligible Wait Time Works

A job accrues ineligible_time while it is blocked by user or group limits, such as:

```plaintext
max_run
max_run_soft
max_run_res.<resource>
max_run_res_soft.<resources>
```
A job also accrues ineligible_time while it is blocked due to a user hold or while it is waiting for its start time, such as when submitted via

```
qsub -a <run-after> ...
```

A job accrues eligible_time when it is blocked by a lack of resources, or by anything not qualifying as ineligible_time or run_time. A job’s eligible_time will only increase during the life of the job, so if the job is requeued, its eligible_time is preserved, not set to zero. The job’s eligible_time is not recalculated when a job is qmoved or moved due to peer scheduling.

The kind of time a job is accruing is sampled periodically, with a granularity of seconds.

A job’s eligible_time attribute can be viewed via `qstat -f`.

### 4.8.13.3 Configuring Eligible Wait Time

To enable using eligible time as the job’s wait time, set the eligible_time_enable server attribute to True.

### 4.8.13.4 How Eligible Wait Time Is Used

- You can choose to use each job’s eligible wait time as the amount of time it is starving. See section 4.8.45 “Starving Jobs” on page 307.
- When a job is requeued, for example being checkpointed and aborted or preempted, its accumulated queue waiting time depends on how that time is calculated:
  - If you are using eligible time, the accumulated waiting time is preserved
  - If you are not using eligible time, the accumulated waiting time is lost

See section 10.3 “Checkpoint and Restart” on page 735 and section 4.8.33 “Using Preemption” on page 258.
4.8.13.5 Altering Eligible Time

You can set the value for a job’s `eligible_time` attribute using the `qalter` command, for example:

```
qalter -W eligible_time=<time> <job ID>
```

4.8.13.6 Attributes Affecting Eligible Time

`eligible_time_enable`

Server attribute. Enables accumulation of eligible time for jobs. Controls whether a job’s `eligible_time` attribute is used as its starving time. See section 4.8.45 “Starving Jobs” on page 307.

On an upgrade from versions of PBS prior to 9.1 or on a fresh install, `eligible_time_enable` is set to `False` by default.

When `eligible_time_enable` is set to `False`, PBS does not track `eligible_time`. Whether `eligible_time` continues to accrue for a job or not is undefined. The output of `qstat -f` does not include `eligible_time` for any job.

Accounting logs do not show `eligible_time` for any job submitted before or after turning `eligible_time_enable` off. Log messages do not include accrual messages for any job submitted before or after turning `eligible_time_enable` off.

If the scheduling formula includes `eligible_time`, `eligible_time` evaluates to 0 for all jobs.

When `eligible_time_enable` is changed from `False` to `True`, jobs accrue `eligible_time` or `ineligible_time` or `run_time` as appropriate. A job’s `eligible_time` is used for starving calculation starting with the next scheduling cycle; changing the value of `eligible_time_enable` does not change the behavior of an active scheduling cycle.
accrue_type
Job attribute. Indicates what kind of time the job is accruing.

Table 4-6: Job’s accrue_type Attribute

<table>
<thead>
<tr>
<th>Type</th>
<th>Numeric Representation</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB_INITIAL</td>
<td>0</td>
<td>initial_time</td>
</tr>
<tr>
<td>JOB_INELIGIBLE</td>
<td>1</td>
<td>ineligible_time</td>
</tr>
<tr>
<td>JOB_ELIGIBLE</td>
<td>2</td>
<td>eligible_time</td>
</tr>
<tr>
<td>JOB_RUNNING</td>
<td>3</td>
<td>run_time</td>
</tr>
<tr>
<td>JOB_EXIT</td>
<td>4</td>
<td>exit_time</td>
</tr>
</tbody>
</table>

The job’s accrue_type attribute is visible via qstat only by Manager, and is set only by the server.

eligible_time
Job attribute. The amount of wall clock wait time a job has accrued because the job is blocked waiting for resources, or any other reason not covered by ineligible_time. For a job currently accruing eligible_time, if we were to add enough of the right type of resources, the job would start immediately. Viewable via qstat -f by job owner, Manager and Operator. Settable by Operator or Manager.

4.8.13.7 Logging

The server prints a log message every time a job changes its accrue_type, with both the new accrue_type and the old accrue_type. These are logged at the 0x0400 event class.

Server logs for this feature display the following information:

- Time accrued between samples
- The type of time in the previous sample, which is one of initial time, run time, eligible time or ineligible time
• The next type of time to be accrued, which is one of run time, eligible time or ineligible time

• The eligible time accrued by the job, if any, until the current sample

Example:

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 0 secs of initial_time, new
  accrue_type=eligible_time, eligible_time=00:00:00

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 1821 secs of eligible_time, new
  accrue_type=ineligible_time, eligible_time=01:20:22

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 2003 secs of ineligible_time, new
  accrue_type=eligible_time, eligible_time=01:20:22

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 61 secs of eligible_time, new accrue_type=run_time,
  eligible_time=01:21:23

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 100 secs of run_time, new
  accrue_type=ineligible_time, eligible_time=01:21:23

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 33 secs of ineligible_time, new
  accrue_type=eligible_time, eligible_time=01:21:23

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 122 secs of eligible_time, new accrue_type=run_time,
  eligible_time=01:23:25

08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 1210 secs of run_time, new accrue_type=exiting,
  eligible_time=01:23:25
The example shows the following changes in time accrual:

- initial to eligible
- eligible to ineligible
- ineligible to eligible
- eligible to running
- running to ineligible
- ineligible to eligible
- eligible to running
- running to exiting

The server also logs the change in accrual when the job’s eligible_time attribute is altered using qalter. For example, if the job’s previous eligible time was 123 seconds, and it has been altered to be 1 hour and 1 minute:

```
Accrue type is eligible_time, previous accrue type was eligible_time for 123 secs, due to qalter total eligible_time=01:01:00
```

### 4.8.13.8 Accounting

Each job’s eligible_time attribute is included in the “E” records in the PBS accounting logs.

Example:

```
08/07/2007 19:34:06;E;182.Host1;user=user1 group=user1 jobname=STDIN queue=workq ctime=1186494765 qtime=1186494765 etime=1186494765 start=1186494767 exec_host=Host1/0 exec_vnode=(Host1:ncpus=1) Resource_List.ncpus=1 Resource_List.nodect=1 Resource_List.place=pack Resource_List.select=1:ncpus=1 session=4656 end=1186495446 Exit_status=-12 resources_used.cpu-percent=0 resources_used.cput=00:00:00 resources_used.mem=3072kb resources_used.ncpus=1 resources_used.vmem=13356kb resources_used.wall-time=00:11:21 eligible_time=00:10:00
```
4.8.14 Sorting Jobs by Entity Shares (Was Strict Priority)

You can sort jobs according to how much of the fairshare tree is allocated to the entity that owns the job. The fairshare percentages in the fairshare tree describe each entity’s share. Using entity shares is sorting jobs on a key, using the `fair_share_perc` option to the `job_sort_key` scheduler parameter.

Using entity shares, the jobs from an entity with greater allocation in the fairshare tree run before the jobs with a smaller allocation.

4.8.14.1 Configuring Entity Shares

To configure entity shares, do the following:

- Define fairshare tree entity allocation in `PBS_HOME/sched_priv/resource_group`. See section 4.8.18 “Using Fairshare” on page 191. You can use a simple fairshare tree, where every entity’s `parent_group` is `root`.
  - Give each entity shares according to desired priority, with higher-priority entities getting larger allocations.
  - Set the `unknown_shares` scheduler parameter to 1. This causes any entity not in your list of approved entities to have a tiny allocation, and the lowest priority.

For example:

```
usr1   60   root   5
usr2   61   root  15
usr3   62   root  15
usr4   63   root  10
usr5   64   root  25
usr6   65   root  30
```

- Set `fair_share_perc` as the option to `job_sort_key`, for example:

  `job_sort_key: “fair_share_perc HIGH all”`
4.8.14.2  Viewing Entity Shares

When you are root, you can use the `pbsfs` command to view the fairshare tree allocations.

4.8.15  Estimating Job Start Time

PBS can estimate when jobs will run, and which vnodes each job will use. PBS estimates job start times and vnodes for all jobs using an asynchronous process, not the PBS server, scheduler, or MOM daemons.

Jobs have an attribute called `estimated` for reporting estimated start time and estimated vnodes. This attribute reports the values of two read-only built-in resources, `start_time` and `exec_vnode`. Each job’s estimated start time is reported in `estimated.start_time`, and its estimated vnodes are reported in `estimated.exec_vnode`.

PBS automatically sets the value of each job’s `estimated.start_time` value to the estimated start time for each job.

4.8.15.1  Configuring Start Time Estimation

PBS estimates values for `start_time` and `exec_vnode` for jobs in the following ways:

- When `est_start_time_freq` is set to a value greater than zero, PBS estimates values for all jobs at the specified interval
- When `est_start_time_freq` is set to zero, PBS estimates values for all jobs after each scheduling cycle
- When the scheduler is backfilling around top jobs, it estimates the start times and `exec_vnode` for those jobs being backfilled around

If you want PBS to estimate the start time for all jobs, either set the `est_start_time_freq` server attribute to the interval at which you want PBS to make the calculation, or set it to `zero`, and the calculation will be made every scheduling cycle.
You can choose to have estimated start times for just the jobs being backfilled around. You set the number of jobs to be backfilled around by setting the server’s `backfill_depth` attribute to the desired number. See section 4.8.3 “Using Backfilling” on page 145.

You can also choose to have `start_time` and `exec_vnode` estimated both for jobs being backfilled around and for all jobs.

Example 1: To estimate start times for the top 5 jobs every scheduling cycle, and for all jobs every 3 hours:

```
qmgr -c 'set server backfill_depth=5'
qmgr -c 'set server est_start_time_freq = 3:00:00'
```

### 4.8.15.2 Controlling User Access to Start Times and Vnode List

#### 4.8.15.2.1 Making Start Time or Vnodes Invisible

You can make job estimated start times and vnodes invisible to unprivileged users by adding resource permission flags to the `start_time` or `exec_vnode` resources. To do this, add the resource to the `resource-def` file, and include the `i` flag, in the same way you would for a custom resource being made invisible. Restart the server; see section 5.14.3.1 “Restarting the Server” on page 367.

Example of making `start_time` and `exec_vnode` invisible to users:

```
start_time       type=long     flag=i
exec_vnode      type=string   flag=i
```

You can always make the start time and vnodes visible again to unprivileged users by removing the entries (or flags) in the `resource-def` file and restarting the server.

See section 5.14.2.5 “Resource Permission Flags” on page 362.
4.8.15.2.2 Allowing Users to See Only Their Own Job Start Times

If you want users to be able to see the start times for their own jobs, but not those of other users, set the server’s `query_other_jobs` attribute to `False`, and do not set the `i` or `r` permission flags. Setting the server’s `query_other_jobs` attribute to `False` prevents a user from seeing anything about other users’ jobs.

4.8.15.3 Attributes and Parameters Affecting Job Start Time Estimation

`backfill`

Scheduler parameter.

Toggle that controls whether PBS uses backfilling. If this is set to `True`, the scheduler attempts to schedule smaller jobs around higher-priority jobs when using `strict_ordering`, as long as running the smaller jobs won't change the start time of the jobs they were scheduled around. The scheduler chooses jobs in the standard order, so other high-priority jobs will be considered first in the set to fit around the highest-priority job.

If this parameter is `True`, the scheduler backfills around starving jobs when `help_starving_jobs` is `True`.

PBS calculates `estimated.start_time` and `estimated.exec_vnode` for these jobs at each scheduler iteration.

Can be used with `strict_ordering` and `help_starving_jobs`.

Format: `Boolean`

Default: `True all`

See “backfill” on page 329 of the PBS Professional Reference Guide.

`backfill_depth`

Server attribute.

Modifies backfilling behavior. Only used when server’s `backfill` attribute is `True`. Sets the number of jobs that are to be backfilled around.
Chapter 4  

Scheduling

Recommendation: set this to less than 100.

Format: Integer

Default: 1

See “Server Attributes” on page 365 of the PBS Professional Reference Guide.

estimated

List of values associated with job's estimated start time. Used to report jobs's start_time and exec_vnode information. Can be set in a hook or via qalter, but PBS will overwrite the values. Allowable values: start_time, exec_vnode

Format: estimated.start_time, estimated.exec_vnode

Default: Unset

Python attribute value type: Dictionary: estimated.<resource name>==<value> where <resource name> is any resource

est_start_time_freq

Server attribute.

Interval at which PBS calculates estimated start times and vnodes for all jobs.

Best value is workload-dependent. Recommendation: set this to two hours.

When set to 0, PBS estimates start times for all jobs.

Format: Duration

Default: Unset

See “Server Attributes” on page 365 of the PBS Professional Reference Guide.

help_starving_jobs

Scheduler parameter.

Setting this option enables starving job support. Once jobs have waited for the amount of time given by max_starve they are considered starving. If a job is considered starving, then no lower-priority jobs will run until the starving job can be run, unless backfilling is also specified. To use this
option, the max_starve configuration parameter needs to be set as well. See also backfill, max_starve, and the server’s eligible_time_enable attribute.

Format: Boolean
Default: True all

strict_ordering

Specifies that jobs must be run in the order determined by whatever sorting parameters are being used. This means that a job cannot be skipped due to resources required not being available. If a job due to run next cannot run, no job will run, unless backfilling is used, in which case jobs can be backfilled around the job that is due to run next.

Example line in PBS_HOME/sched_priv/sched_config:

strict_ordering: True ALL

Format: Boolean.
Default: False all

4.8.15.4 Viewing Estimated Start Times

You can view the estimated start times and vnodes of jobs using the qstat command. If you use the -T option to qstat when viewing job information, the Est Start field is displayed. Running jobs are shown above queued jobs.

See “qstat” on page 228 of the PBS Professional Reference Guide.

If the estimated start time or vnode information is invisible to unprivileged users, no estimated start time or vnode information is available via qstat.

Example output:

```
qstat -T

Job ID  Username  Queue  Jobname  SessID  NDS  TSK  Memory  Time  S  Start
-------  --------  -----  --------  -----  ---  ---   ------  -----  -  -----  
5.host1  user1    workq  foojob  12345  1    1    128mb  00:10  R  --
9.host1  user1    workq  foojob  --    1    1    128mb  00:10  Q  11:30
```
4.8.15.5 Selecting Jobs By Estimated Start Time

You can use the `qselect` command to select jobs according to their start times by using the `-t` suboption to the `-t` option. This selects jobs according to the value of the `estimated.start_time` attribute. See “`qselect`” on page 215 of the PBS Professional Reference Guide.

4.8.15.6 Logging

Whenever the scheduler estimates the start time of a job, it logs the start time. The scheduler does not log the estimated `exec_vnode` of a job.

4.8.15.7 Caveats and Advice

- The `estimated.start_time` of a job array is the time calculated for the first queued subjob only.

- Cached estimated start times are only as fresh as the last time PBS calculated them. This should be taken into account when setting the values of `est_start_time_freq` and `backfill_depth`.

- The frequency of calculating start times is a trade-off between having more current start time information and using fewer computing cycles for non-job work. The background task of calculating start times can be computationally intensive. This should be taken into account when setting the value of `est_start_time_freq`. Depending on the size of your site, it is probably a good idea not to set it to less than 10 minutes.

- The best value for `est_start_time_freq` is workload dependent, but we recommend setting it to two hours as a starting point.

- If your site has short scheduling cycles of a few minutes, and can use backfilling (and at least one of strict ordering or starving jobs), you can have the start times for all jobs calculated at each scheduling cycle. To do this, set `backfill_depth` to a value greater than the number of jobs.
the site will ever have, and do not set `est_start_time_freq`.

• We recommend setting `backfill_depth` to a value that is less than 100.

## 4.8.16 Calculating Job Execution Priority

When the scheduler examines jobs, either at the whole complex or within a queue, it gives each job an execution priority, and then uses this job execution priority to select which job(s) to run. Job execution priority is mostly independent of job preemption priority. We discuss only job execution priority in this section.

Some of the scheduler’s policy for determining job execution priority is built into PBS, but you can specify how execution priority is determined for most of the policy.

First, the scheduler divides queued jobs into classes. Then it sorts the jobs within each class.

### 4.8.16.1 Dividing Jobs Into Classes

PBS has special classes that supersede queue order, meaning that whether or not queues are being examined separately, the jobs in each of those classes are treated as if they are in a single queue. They are not ordered according to which queue they reside in. For example, all starving jobs are treated as if they are in a single queue. PBS has one class for all non-special jobs. This class typically contains most PBS jobs. Queue order is imposed on this class, meaning that if queues are ordered and sorted, jobs are examined queue by queue.

Job execution classes have a built-in order of precedence. All jobs in the highest class are considered before any jobs in the next class, and so on. Classes are listed in the following table, highest first:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Sort Applied Within Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservation</td>
<td>Jobs submitted to an advance or standing reservation</td>
<td>Formula or job sort key</td>
</tr>
</tbody>
</table>
4.8.16.2 Selecting Job Execution Class

The scheduler places each job in the highest-priority class into which the job can fit. So, for example, if a job is both in a reservation and is starving, the job is placed in the Reservation class.

4.8.16.3 Sorting Jobs Within Classes

Jobs within each class are sorted according to rules specific to each class. The sorting applied to each class is listed in Table 4-7, “Job Execution Classes,” on page 187.

- The **Express** class is made up of all the jobs that have a higher priority than “normal_jobs” in the preempt_prio scheduler parameter. The Express class is sorted first by applying the rules for preemption prior-
ity you set in the scheduler’s `preempt_prio` parameter, making pre-emption priority the first sort key. The second sort key is the formula or `job_sort_key`, depending on which is defined, or if neither is defined, job submission time. See section 4.8.33 “Using Preemption” on page 258. Please note that execution priority classes are distinct from preemption levels, and are used for different purposes.

For example, if `preempt_prio` is the following:

```plaintext
preempt_prio: “express_queues, starving_jobs,
express_queue, normal_jobs”
```

The `Express` class contains all jobs that have preemption priority that is greater than that of normal jobs. In this example, the `Express` class is prioritized with top priority for express queue jobs, followed by starving jobs.

Since preemption levels are applied so that a job is put into the highest preemption level possible, in this example, all starving jobs end up in the `Express` class.

- The `Starving` class is sorted only according to the amount of time that counts toward starving for each job. You can use queue wait time or eligible time as starving time. See section 4.8.45 “Starving Jobs” on page 307.

- The `Reservation` class is sorted within each reservation, according to the formula or `job_sort_key`, depending on which is defined. If neither is defined, jobs are sorted according to their submission time.

- The `Suspended` class is sorted according to the formula or `job_sort_key`, depending on which is defined. If neither is defined, jobs are sorted according to their submission time. All jobs in the `Suspended` class are sorted as if they are in a single queue.

- For `Normal` jobs, queue order is the first sort key, and, if defined, the formula, fairshare, or `job_sort_key` is the second sort key. If `job_sort_formula`, `fair_share`, and `job_sort_key` are not defined, the second sort key is job submission time.

One class is not used to divide or sort another. For example, if there are jobs in the `Express` class that are also starving or suspended, those factors are not used to sort the jobs in the `Express` class.

While the last sort key does sort jobs at the finest granularity, most of the work of sorting jobs is done at this level.
4.8.16.3.1 Precedence of Sort Method Used Within Class

If the formula is defined, it overrides fairshare and the job sort key. If fair share is defined, it overrides the job sort key. If none are defined, jobs are ordered by their arrival time in the queue.

For the job sorting formula, see section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

For fairshare, see section 4.8.18 “Using Fairshare” on page 191.

For sorting jobs on a key, see section 4.8.42 “Sorting Jobs on a Key” on page 302.

4.8.16.4 Execution Priority Caveats

- Limits are not taken into account when prioritizing jobs for execution. Limits are checked only after setting priority, when selecting a job to run. The only exception is in the Express class, where soft limits may be taken into account, because execution priority for Express class jobs is calculated using preemption priority. For details, see section 4.8.33 “Using Preemption” on page 258.

- When you issue “qrun <job ID>”, without the -H option, the selected job has execution priority between Reservation and Express.

4.8.17 Express Queues

You can use express queues as tools to manage job execution and preemption priority:

- You can set up execution priority levels that include jobs in express queues. For information on configuring job priorities in the scheduler, see section 4.8.16 “Calculating Job Execution Priority” on page 187.

- You can set up preemption levels that include jobs in express queues. For information on preemption, see section 4.8.33 “Using Preemption” on page 258.
An express queue is a queue whose priority is high enough to qualify as an express queue; the default for qualification is 150, but this can be set using the `preempt_queue_prio` scheduler parameter. For information on configuring express queues, see section 2.2.3.1 “Express Queues” on page 26.

The term “express” is also used in calculating execution priority to mean all jobs that have a preemption level greater than that of the `normal_jobs` level.

### 4.8.18 Using Fairshare

Fairshare provides a way to enforce a site's resource usage policy. It is a method for ordering the start times of jobs based on two things: how a site's resources are apportioned, and the resource usage history of site members. Fairshare ensures that jobs are run in the order of how deserving they are. The scheduler performs the fairshare calculations each scheduling cycle. If fairshare is enabled, all jobs have fairshare applied to them and there is no exemption from fairshare.

The administrator can employ basic fairshare behavior, or can apply a policy of the desired complexity.

The `fair_share` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

#### 4.8.18.1 Outline of How Fairshare Works

The owner of a PBS job can be defined for fairshare purposes to be a user, a group, the job’s accounting string, etc. For example, you can define owners to be groups, and can explicitly set each group’s relationship to all the other groups by using the tree structure. You can define one group to be part of a larger department.

The usage of exactly one resource is tracked for all job owners. So if you defined job owners to be groups, and you defined `cput` to be the resource that is tracked, then only the `cput` usage of groups is considered. PBS tries to ensure that each owner gets the amount of resources that you have set for it.
4.8.18.2 The Fairshare Tree

Fairshare uses a tree structure, where each vertex in the tree represents some set of job owners and is assigned usage shares. Shares are used to apportion the site’s resources. The default tree always has a root vertex and an unknown vertex. The default behavior of fairshare is to give all users the same amount of the resource being tracked. In order to apportion a site's resources according to a policy other than equal shares for each user, the administrator creates a fairshare tree to reflect that policy. To do this, the administrator edits the file PBS_HOME/sched_priv/resource_group, which describes the fairshare tree.

4.8.18.3 Enabling Basic Fairshare

If the default fairshare behavior is enabled, all users with queued jobs will get an equal share of CPU time. The root vertex of the tree will have one child, the unknown vertex. All users will be put under the unknown vertex, and appear as children of the unknown vertex.

Basic fairshare is enabled by doing the following:

• In PBS_HOME/sched_priv/sched_config, set the scheduler configuration parameter fair_share to True
• Uncomment the unknown_shares setting so that it is set to unknown_shares: 10
• Specify how you want fairshare to work with primetime and non-prime-time. If you want separate behavior for primetime and non-primetime, list the fair_share parameter twice, once for each time slot. The default is both. For example:

    fair_share True prime
    fair_share False non_prime

Note that a variant of basic fairshare has all users listed in the tree as children of root. Each user can be assigned a different number of shares. This must be explicitly created by the administrator.
4.8.18.4 Using Fairshare to Enforce Policy

The administrator sets up a hierarchical tree structure made up of interior vertices and leaves. Interior vertices are *departments*, which can contain both departments and leaves. Leaves are for *fairshare entities*, defined by setting `fairshare_entity` to one of the following: `euser`, `egroup`, `egroup:euser`, `Account_Name`, or `queue`. Apportioning of resources for the site is among these entities. These entities' usage of the designated resource is used in determining the start times of the jobs associated with them. All fairshare entities must be the same type. If you wish to have a user appear in more than one department, you can use `egroup:euser` to distinguish between that user's different resource allotments.

**Table 4-8: Using Fairshare Entities**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Fairshare Entities</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>euser</code></td>
<td>Username</td>
<td>Individual users are allotted shares of the resource being tracked. Each username may only appear once, regardless of group.</td>
</tr>
<tr>
<td><code>egroup</code></td>
<td>OS group name</td>
<td>Groups as a whole are allotted shares of the resource being tracked.</td>
</tr>
<tr>
<td><code>egroup:euser</code></td>
<td>Combinations of username and group name</td>
<td>Useful when a user is a member of more than one group, and needs to use a different allotment in each group.</td>
</tr>
<tr>
<td><code>Account_Name</code></td>
<td>Account IDs</td>
<td>Shares are allotted by account string (<code>Account_Name</code> job attribute).</td>
</tr>
<tr>
<td><code>queues</code></td>
<td>Queues</td>
<td>Shares are allotted between queues.</td>
</tr>
</tbody>
</table>
4.8.18.4.1 Shares in the Tree

The administrator assigns shares to each vertex in the tree. The actual number of shares given to a vertex or assigned in the tree is not important. What is important is the ratio of shares among each set of sibling vertices. Competition for resources is between siblings only. The sibling with the most shares gets the most resources.

4.8.18.4.2 Shares Among Unknown Entities

The root vertex always has a child called unknown. Any entity not listed in PBS_HOME/sched_priv/resource_group will be made a child of unknown, designating the entity as unknown. The shares used by unknown entities are controlled by two parameters in PBS_HOME/sched_priv/sched_config: unknown_shares and fairshare_enforce_no_shares.

The parameter unknown_shares controls how many shares are assigned to the unknown vertex. The default sched_config file contains this line:

```
#unknown_shares 10
```

If you leave unknown_shares commented out, the unknown vertex will have 0 shares. If you simply remove the “#”, the unknown vertex's shares default to 10. The children of the unknown vertex have equal amounts of the shares assigned to the unknown vertex.

The parameter fairshare_enforce_no_shares controls whether an entity without any shares can run jobs. If fairshare_enforce_no_shares is True, then entities without shares cannot run jobs. If it is set to False, entities without any shares can run jobs, but only when no other entities’ jobs are available to run.

4.8.18.4.3 Format for Describing the Tree

The file describing the fairshare tree contains four columns to describe the vertices in the tree. Here is the format for the columns:

```
<Vertex name>  <vertex fairshare ID>  <parent of vertex>  <#shares>
```
The columns are for a vertex's name, its fairshare ID, the name of its parent vertex, and the number of shares assigned to that vertex. Vertex names and IDs must be unique. Vertex IDs are integers. The top row in resource_group contains information for the first vertex, rather than column labels.

Neither the root vertex nor the unknown vertex is described in PBS_HOME/sched_priv/resource_group. They are always added automatically. Parent vertices must be listed before their children.

For example, we have a tree with two top-level departments, Math and Phys. Under Math are the users Bob and Tom as well as the department Applied. Under Applied are the users Mary and Sally. Under Phys are the users John and Joe. Our PBS_HOME/sched_priv/resource_group looks like this:

<table>
<thead>
<tr>
<th>Vertex</th>
<th>ID</th>
<th>Parent</th>
<th>Shares</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>100</td>
<td>root</td>
<td>30</td>
</tr>
<tr>
<td>Phys</td>
<td>200</td>
<td>root</td>
<td>20</td>
</tr>
<tr>
<td>Applied</td>
<td>110</td>
<td>Math</td>
<td>20</td>
</tr>
<tr>
<td>Bob</td>
<td>101</td>
<td>Math</td>
<td>20</td>
</tr>
<tr>
<td>Tom</td>
<td>102</td>
<td>Math</td>
<td>10</td>
</tr>
<tr>
<td>Mary</td>
<td>111</td>
<td>Applied</td>
<td>1</td>
</tr>
<tr>
<td>Sally</td>
<td>112</td>
<td>Applied</td>
<td>2</td>
</tr>
<tr>
<td>John</td>
<td>201</td>
<td>Phys</td>
<td>2</td>
</tr>
<tr>
<td>Joe</td>
<td>202</td>
<td>Phys</td>
<td>2</td>
</tr>
</tbody>
</table>

If you wish to use egroup:euser as your entity, and Bob to be in two UNIX/Windows groups pbsgroup1 and pbsgroup2, and Tom to be in two groups pbsgroup2 and pbsgroup3:

<table>
<thead>
<tr>
<th>Vertex</th>
<th>ID</th>
<th>Parent</th>
<th>Shares</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>100</td>
<td>root</td>
<td>30</td>
</tr>
<tr>
<td>Phys</td>
<td>200</td>
<td>root</td>
<td>20</td>
</tr>
<tr>
<td>Applied</td>
<td>110</td>
<td>Math</td>
<td>20</td>
</tr>
<tr>
<td>pbsgroup1:Bob</td>
<td>101</td>
<td>Phys</td>
<td>20</td>
</tr>
<tr>
<td>pbsgroup2:Bob</td>
<td>102</td>
<td>Math</td>
<td>20</td>
</tr>
<tr>
<td>pbsgroup2:Tom</td>
<td>103</td>
<td>Math</td>
<td>10</td>
</tr>
<tr>
<td>pbsgroup3:Tom</td>
<td>104</td>
<td>Applied</td>
<td>10</td>
</tr>
</tbody>
</table>
A user’s egroup, unless otherwise specified, will default to their primary UNIX/Windows group. When a user submits a job using 
-Wgroup_list=<group>, the job’s egroup will be <group>. For example, user Bob is in pbsgroup1 and pbsgroup2. Bob uses “qsub 
-Wgroup_list=pbsgroup1” to submit a job that will be charged to pbsgroup1, and “qsub -Wgroup_list=pbsgroup2” to submit a job that will be charged to pbsgroup2.

The first and third fields are alphanumeric. The second and fourth fields are numeric. Fields can be separated by spaces and tabs.

4.8.18.4.4 Computing How Much Each Vertex Deserves

How much resource usage each entity deserves is its portion of all the shares in the tree, divided by its past and current resource usage.

A vertex's portion of all the shares in the tree is called tree percentage. It is computed for all of the vertices in the tree. Since the leaves of the tree represent the entities among which resources are to be shared, their tree percentage sums to 100 percent.

The scheduler computes the tree percentage for the vertices this way:

First, it gives the root of the tree a tree percentage of 100 percent. It proceeds down the tree, finding the tree percentage first for immediate children of root, then their children, ending with leaves.

1. For each internal vertex A:
   
   sum the shares of its children;

2. For each child J of vertex A:

   divide J's shares by the sum to normalize the shares;

   multiply J's normalized shares by vertex A's tree percentage to find J's tree percentage.

4.8.18.5 Tracking Resource Usage

The administrator selects exactly one resource to be tracked for fairshare purposes by setting the scheduler configuration parameter fairshare_usage_res in PBS_HOME/sched_priv/sched_config.
Fairshare is effective only when the resource used is a time-based resource. The time-based resources are cput and the exact string "ncpus*walltime" which multiplies the number of CPUs requested by the walltime in seconds. The default for the tracked resource is cput, CPU time.

To use CPU time as the resource to be tracked, put this line in PBS_HOME/sched_priv/sched_config:

    fairshare_usage_res: cput

To use ncpus multiplied by walltime as the resource to be tracked, put this line in PBS_HOME/sched_priv/sched_config:

    fairshare_usage_res: ncpus*walltime

An entity's usage always starts at 1. Resource usage tracking begins when the scheduler is started.

Each entity's current usage of the designated resource is combined with its previous usage. Each scheduler cycle, the scheduler adds the usage increment between this cycle and the previous cycle to its sum for the entity. Each entity's usage is decayed, or cut in half periodically, at the interval set in the half_life parameter in PBS_HOME/sched_priv/sched_config. This interval defaults to 24 hours.

This means that an entity with a lot of current or recent usage will have low priority for starting jobs, but if the entity cuts resource usage, its priority will go back up after a few decay cycles.

A static resource will not change its usage from one cycle to the next. If you use a static resource such as ncpus, the amount being tracked will not change during the lifetime of the job; it will only be added once when the job starts.

Note that if a job ends between two scheduling cycles, its resource usage for the time between previous scheduling cycle and the end of the job will not be recorded. The scheduler's default cycle interval is 10 minutes. The scheduling cycle can be adjusted via the qmgr command. Use

    Qmgr: set server scheduler_iteration=<new value>
4.8.18.6 Finding the Most Deserving Entity

The most deserving entity is found by starting at the root of the tree, comparing its immediate children, finding the most deserving, then looking among that vertex's children for the most deserving child. This continues until a leaf is found. In a set of siblings, the most deserving vertex will be the vertex with the lowest ratio of resource usage divided by tree percentage.

4.8.18.7 Choosing Which Job to Run

The job to be run next will be selected from the set of jobs belonging to the most deserving entity. The jobs belonging to the most deserving entity are sorted according to the methods the scheduler normally uses. This means that fairshare effectively becomes the primary sort key. If the most deserving job cannot run, then the next most is selected to run, and so forth. All of the most deserving entity's jobs would be examined first, then those of the next most deserving entity, et cetera.

At each scheduling cycle, the scheduler attempts to run as many jobs as possible. It selects the most deserving job, runs it if it can, then recalculates to find the next most deserving job, runs it if it can, and so on.

When the scheduler starts a job, all of the job's requested usage is added to the sum for the owner of the job for one scheduling cycle. The following cycle, the job's usage is set to the actual usage used between the first and second cycles. This prevents one entity from having all its jobs started and using up all of the resource in one scheduling cycle.
4.8.18.8 Files and Parameters Used in Fairshare

The following parameters from `PBS_HOME/sched_priv/sched_config` are used in fairshare:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>fair_share</td>
<td>[True/False] Enable or disable fairshare</td>
</tr>
<tr>
<td>fairshare_usage_res</td>
<td>Resource whose usage is to be tracked; default is cput</td>
</tr>
<tr>
<td>half_life</td>
<td>Decay time period; default is 24 hours</td>
</tr>
<tr>
<td>unknown_shares</td>
<td>Number of shares for unknown vertex; default 10, 0 if commented out</td>
</tr>
<tr>
<td>fairshare_entity</td>
<td>The kind of entity which is having fairshare applied to it. Leaves in the tree are this kind of entity. Default: euser.</td>
</tr>
<tr>
<td>fairshare_enforce_no_shares</td>
<td>If an entity has no shares, this controls whether it can run jobs. T: an entity with no shares cannot run jobs. F: an entity with no shares can only run jobs when no other jobs are available to run.</td>
</tr>
<tr>
<td>by_queue</td>
<td>If on, queues cannot be designated as fairshare entities, and fairshare will work queue by queue instead of on all jobs at once.</td>
</tr>
</tbody>
</table>

PBS_HOME/sched_priv/resource_group
Contains the description of the fairshare tree.

PBS_HOME/sched_priv/usage
Contains the usage database.

qmgr
Used to set scheduler cycle frequency; default is 10 minutes.

Qmgr: set server scheduler_iteration=<new value>

job attributes
Used to track resource usage:

resources_used.<resource>
Default is cput.

4.8.18.9 Fairshare for Complex or within Queues

You can use fairshare to compare all jobs in the complex, or within each queue. Fairshare with a queue means that the scheduler examines the jobs in a queue, and compares them to each other, to determine which job to start next.

To use fairshare for the entire complex, set the by_queue and round_robin scheduler configuration parameters to False.

To use fairshare within queues, set the by_queue scheduler parameter to True, and round_robin to False. If you want to examine queues in a particular order, prioritize the queues by setting each queue’s priority attribute, and set the sort_queues scheduler parameter to True.

The scheduler configuration parameter by_queue in the file PBS_HOME/sched_priv/sched_config is set to True by default.

If by_queue is True, queues cannot be designated as fairshare entities.

4.8.18.10 Fairshare and Strict Ordering

Fairshare dynamically reorders the jobs with every scheduling cycle. Strict ordering is a rule that says we always run the next-most-deserving job. If there were no new jobs submitted, strict ordering could give you a snapshot
of how the jobs would run for the next \( n \) days. Hence fairshare appears to break that. However, looked at from a dynamic standpoint, fairshare is another element in the strict order.

### 4.8.18.11 Fairshare and Entity Shares (Strict Priority)

If you enable entity shares (strict priority), you use the same fairshare tree that you would use for fairshare. Fairshare and entity shares (strict priority) are incompatible, so in order to use entity shares, you disable fairshare by setting `fair_share` to `False`. For how to configure entity shares, see section 4.8.14 “Sorting Jobs by Entity Shares (Was Strict Priority)” on page 180.

### 4.8.18.12 Viewing and Managing Fairshare Data

The `pbsfs` command provides a command-line tool for viewing and managing some fairshare data. You can display the tree in tree form or in list form. You can print all information about an entity, or set an entity's usage to a new value. You can force an immediate decay of all the usage values in the tree. You can compare two fairshare entities. You can also remove all entities from the `unknown` department. This makes the tree easier to read. The tree can become unwieldy because entities not listed in the file `PBS_HOME/sched_priv/resource_group` all land in the `unknown` group.

The fairshare usage data is written to the file `PBS_HOME/sched_priv/usage`, whenever the scheduler has new usage data. The usage data is always up-to-date.

For more information on using the `pbsfs` command, see “pbsfs” on page 120 of the PBS Professional Reference Guide.

### 4.8.18.13 Using Fairshare in Job Execution Priority

Jobs are sorted as specified by the formula in `job_sort_formula`, if it exists, or by fairshare, if it is enabled and there is no formula, or if neither of those is used, by `job_sort_key`. The job sorting formula can use the value of `fair_share_perc`, the percentage of the fairshare tree for this job’s entity. See section 4.8.16 “Calculating Job Execution Priority” on page 187.
4.8.18.14 Using Fairshare in Job Preemption Priority

You can use the *fairshare* preemption level in determining job preemption priority. This level applies to jobs whose owners are over their fairshare allotment. See section 4.8.33 “Using Preemption” on page 258.

4.8.18.15 Fairshare and Requeued Jobs

When a job is requeued, it normally retains its original place in its execution queue with its former priority. The job is usually the next job to be considered during scheduling, unless the relative priorities of the jobs in the queue have changed. This can happen when the job sorting formula assigns higher priority to another job, another higher-priority job is submitted after the requeued job started, this job’s owner has gone over their fairshare limit, etc.

4.8.18.16 Fairshare Caveats

- If the job sorting formula is defined, it overrides fairshare.
- Do not use fairshare with *help_starving_jobs*.
- Do not use fairshare with *strict_ordering*, or *strict_ordering* and *backfill*. The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group will be chosen as the filler job. The usage from these small jobs will lower the priority of the most deserving job. See section 4.8.46.4 “Strict Ordering Caveats” on page 311.
- Do not use fairshare when using the *fair_share_perc* option to *job_sort_key*.
- Do not use static resources such as *ncpus* as the resource to track. The scheduler adds the incremental change in the tracked resource at each scheduling cycle, and a static resource will not change.
- The most deserving entity can change with every scheduling cycle, if each time a job is run, it changes usage sufficiently.
- Fairshare dynamically reorders the jobs with every scheduling cycle. Strict ordering is a rule that says we always run the next-most-deserving job. If there were no new jobs submitted, strict ordering could give you a snapshot of how the jobs would run for the next $n$ days. Hence
fairshare appears to break that. However, looked at from a dynamic standpoint, fairshare is another element in the strict order.

### 4.8.19 FIFO Scheduling

With FIFO scheduling, PBS runs jobs in the order in which they are submitted. You can use FIFO order for all of the jobs in your complex, or you can go queue by queue, so that the jobs within each queue are considered in FIFO order.

#### 4.8.19.1 Configuring Basic FIFO Scheduling

To configure basic FIFO scheduling, whether across the complex or queue by queue, set the following scheduler parameters to these values:

- `round_robin`: False ALL
- `job_sort_key`: (commented out)
- `fair_share`: False ALL
- `help_starving_jobs`: False ALL
- `backfill`: False ALL
- `job_sort_formula`: (unset)

#### 4.8.19.2 FIFO for Entire Complex

To configure FIFO across your entire complex, do one of the following:

- Use only one execution queue
- Set the `by_queue` scheduler parameter to `False`

#### 4.8.19.3 Queue by Queue FIFO

To configure FIFO for each queue separately, first decide how you want queues to be selected. You can set the order in which PBS chooses queues from which to run jobs, or you can allow the queues to be selected in an undefined way.

- To allow queues to be selected in an undefined way, set the `by_queue`
scheduler parameter to `True`.

- To set the order in which queues are selected, do the following:
  - Specify a priority for each queue
  - Set the `sort_queues` scheduler parameter to `True`
  - Set the `by_queue` scheduler parameter to `True`

### 4.8.19.4 FIFO with Strict Ordering

If your jobs must run exactly in submission order, you can use strict ordering with FIFO scheduling. If you use strict ordering with FIFO scheduling, this means that when the job that is supposed to run next cannot run, no jobs can run. This can result in less throughput than you could otherwise achieve. To avoid that problem, you can use backfilling. See the following section.

To use strict ordering with FIFO scheduling, use the following scheduler parameter settings in `PBS_HOME/sched_priv/sched_config`:

```
strict_ordering: True    ALL
round_robin: False       ALL
job_sort_key: (commented out)
fair_share: False        ALL
help_starving_jobs: False ALL
backfill: False          ALL
job_sort_formula: (unset)
```
4.8.19.5 FIFO with Strict Ordering and Backfilling

If you want to run your jobs in submission order, except for backfilling around top jobs that are stuck, use the following:

```
strict_ordering: True ALL
round_robin: False ALL
job_sort_key: (commented out)
fair_share False ALL
help_starving_jobs False ALL
backfill: True ALL
job_sort_formula: (unset)
```

4.8.20 Using a Formula for Computing Job Execution Priority

You can choose to use a formula by which to sort jobs at the finest-granularity level. The formula can only direct how jobs are sorted at the finest level of granularity. However, that is where most of the sorting work is done.

When the scheduler sorts jobs according to the formula, it computes a priority for each job. The priority computed for each job is the value produced by the formula. Jobs with a higher value get higher priority. See section 4.8.16.3 “Sorting Jobs Within Classes” on page 188 for how the formula is used in setting job execution priority.

This formula will override both job_sort_key and fair_share for sorting jobs. If the job_sort_formula server attribute contains a formula, the scheduler will use it. If not, and fairshare is enabled, the scheduler computes job priorities according to fairshare. If neither the formula nor fairshare is defined, the scheduler uses job_sort_key.

Only one formula is used to prioritize all jobs. At each scheduling cycle, the formula is applied to all jobs, regardless of when they were submitted. If you change the formula, the new formula is applied to all jobs.
For example, if you submit some jobs, change the formula, then submit more jobs, the new formula is used for all of the jobs, during the next scheduling cycle.

You may find that the formula is most useful when you use it with custom resources inherited by or allocated to jobs. For example, you may want to route all jobs from a particular project to a queue where they inherit a specific value for a custom resource. Other jobs may end up at a different queue, where they inherit a different value, or they may inherit no value. You can then use this custom resource in the formula as a way to manage job priority. See section 12.2 “Allocating Resources to Jobs” on page 842, and section 4.8.8 “Using Custom and Default Resources” on page 155.

It may be helpful if these custom resources are invisible and unrequestable by users. See section 4.8.20.9 “Examples of Using Resource Permissions in Job Sorting Formula” on page 211.

4.8.20.1 Using the Formula

Once you set `job_sort_formula` via `qmgr`, it takes effect with the following scheduling cycle.

Variables are evaluated at the start of the scheduling cycle.

If an error is encountered while evaluating the formula, the formula evaluates to zero for that job, and the following message is logged at event class 0x0100:

“1234.mars;Formula evaluation for job had an error. Zero value will be used”

4.8.20.2 Configuring the Job Sorting Formula

You specify the formula in the server’s `job_sort_formula` attribute. To set the `job_sort_formula` attribute, use the `qmgr` command. When specifying the formula, be sure to follow the requirements for entering an attribute value via `qmgr`: strings containing whitespace, commas, or other special characters must be enclosed in single or double quotes. See “Attribute Values” on page 186 of the PBS Professional Reference Guide. Format:

`Qmgr: s s job_sort_formula = "<formula>"`
4.8.20.3 Requirements for Creating Formula

The job sorting formula must be created at the server host.

Under UNIX/Linux, root privilege is required in order to operate on the job_sort_formula server attribute.

Under Windows, this must be done from the installation account. For domained environments, the installation account must be a local account that is a member of the local Administrators group on the local computer. For standalone environments, the installation account must be a local account that is a member of the local Administrators group on the local computer.

4.8.20.4 Format of Formula

The formula can be made up of any number of expressions, where expressions contain terms which are added, subtracted, multiplied, or divided. You can use parentheses, exponents, and unary + and - operators. All operators use standard mathematical precedence.

The formula can be any length.

The range for the formula is defined by the IEEE floating point standard for a double.

4.8.20.5 Units in Formula

The variables you can use in the formula have different units. Make sure that some terms do not overpower others, by normalizing them where necessary. Resources like ncpus are integers, size resources like mem are in kb, so 1gb is 1048576kb, and time-based resources are in seconds (e.g. walltime). Therefore, if you want a formula that combines memory and ncpus, you’ll have to account for the factor of 1024 difference in the units.

The following are the units for the supported built-in resources:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Units</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time resources</td>
<td><em>Integer number of seconds</em></td>
<td>300</td>
</tr>
</tbody>
</table>
Example 1: If you use ‘1 * ncpus + 1 * mem’, where mem=2mb, ncpus will have almost no effect on the formula result. However, if you use ‘1024 * ncpus + 1 * mem’, the scaled mem won’t overpower ncpus.

Example 2: You are using gb of mem:

\[
\text{Qmgr: s s job_sort_formula=}’1048576 * ncpus + 2 * mem’
\]

Example 3: If you want to add days of walltime to queue priority, you might want to multiply the time by 0.0000115, equivalent to dividing by the number of seconds in a day:

\[
\text{Qmgr: s s job_sort_formula = ‘.0000115*walltime + queue_priority’}
\]

### 4.8.20.6 Formula Coefficients

The formula operates only on resources in the job’s Resource_List attribute. These are the numeric job-level resources, and may have been explicitly requested, inherited, or summed from consumable host-level resources. See section 5.9.2 “Resources Requested by Job” on page 338.

This means that all variables and coefficients in the formula must be resources that were either requested by the job or were inherited from defaults at the server or queue. These variables and coefficients can be custom numeric resources inherited by the job from the server or queue, or they are long integers or floats.

You may need to create custom resources at the server or queue level to be used for formula coefficients. See section 4.8.8 “Using Custom and Default Resources” on page 155.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Units</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>(kb)</td>
<td>1gb =&gt; 1048576kb</td>
</tr>
<tr>
<td>ncpus</td>
<td>(Integer)</td>
<td>8</td>
</tr>
</tbody>
</table>
The following table lists the terms that can be used in the formula:

**Table 4-11: Terms in Job Sorting Formula**

<table>
<thead>
<tr>
<th>Terms</th>
<th>Allowable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constants</strong></td>
<td>NUM or NUM.NUM</td>
</tr>
<tr>
<td>Attribute values</td>
<td></td>
</tr>
<tr>
<td>queue_priority</td>
<td>Value of priority attribute for queue in which job resides</td>
</tr>
<tr>
<td>job_priority</td>
<td>Value of the job’s priority attribute</td>
</tr>
<tr>
<td>fair_share_perc</td>
<td>Percentage of fairshare tree for this job’s entity</td>
</tr>
<tr>
<td>eligible_time</td>
<td>Amount of wait time job has accrued while waiting for resources</td>
</tr>
<tr>
<td><strong>Resources</strong></td>
<td></td>
</tr>
<tr>
<td>ncpus</td>
<td></td>
</tr>
<tr>
<td>mem</td>
<td></td>
</tr>
<tr>
<td>walltime</td>
<td></td>
</tr>
<tr>
<td>cput</td>
<td></td>
</tr>
<tr>
<td><strong>Custom numeric job-wide resources</strong></td>
<td>Uses the amount requested, not the amount used. Must be of type long, float, or size. See section 5.14.2.7 “Custom Resource Values” on page 367.</td>
</tr>
</tbody>
</table>

### 4.8.20.7 Modifying Coefficients For a Specific Job

Formula coefficients can be altered for each job by using the `qalter` command to change the value of that resource for that job. If a formula coefficient is a constant, it cannot be altered per-job.

### 4.8.20.8 Examples of Using the Job Sorting Formula

Examples of formulas:
Chapter 4  Scheduling

Example 1: 10 * ncpus + 0.01 * walltime + A * mem

Here, “A” is a custom resource.

Example 2: ncpus + 0.0001 * mem

Example 3: To change the formula on a job-by-job basis, alter the value of a resource in the job’s Resource_List.<resource>. So if the formula is A * queue_priority + B * job_priority + C * ncpus + D * walltime, where A-D are custom numeric resources. These resources can have a default value via resources_default.A ... resources_default.D. You can change the value of a job’s resource through qalter.

Example 4: ncpus * mem

Example 5: Set via qmgr:

```
qmgr -c 'set server job_sort_formula=5*ncpus+0.05*walltime'
```

Following this, the output from qmgr -c 'print server' will look like

```
set server job_sort_formula="5*ncpus+0.05*walltime"
```

Example 6:

```
Qmgr: s s job_sort_formula=ncpus
```

Example 7:

```
Qmgr: s s job_sort_formula='queue_priority + ncpus'
```

Example 8:

```
Qmgr: s s job_sort_formula='5*job_priority + 10*queue_priority'
```
Example 9: Sort jobs using the value of $ncpus \times walltime$:

Formula expression: “$ncpus \times walltime$”

Submit these jobs:

Job 1: $ncpus=2$ walltime=01:00:00 $\Rightarrow$ $2*60s = 120$

Job 2: $ncpus=1$ walltime=03:00:00 $\Rightarrow$ $1*180s = 180$

Job 3: $ncpus=5$ walltime=01:00:00 $\Rightarrow$ $5*60s = 300$

The scheduler logs the following:

Job ;1.host1;Formula Evaluation = 120
Job ;2.host1;Formula Evaluation = 180
Job; 3.host1;Formula Evaluation = 300

The jobs are sorted in the following order:

Job 3
Job 2
Job 1

4.8.20.9 Examples of Using Resource Permissions in Job Sorting Formula

See section 5.14.2.5 “Resource Permission Flags” on page 362 for information on using resource permissions.

Example 1: You may want to create per-job coefficients in your job sorting formula which are set by system defaults and which cannot be viewed, requested or modified by the user. To do this, you create custom resources for the formula coefficients, and make them invisible to users. In this example, A, B, C and D are the coefficients. You then use them in your formula:

$$A \times (Queue \text{ Prior}ity) + B \times (Job \text{ Class \text{ Prior}ity}) + C \times (CPUs) + D \times (Queue \text{ Wait \text{ Time}})$$

Example 2: You may need to change the priority of a specific job, for example, have one job or a set of jobs run next. In this case, you can define a custom resource for a special job priority. If you do not want users to be able to change this priority, set the resource permission flag
for the resource to \( r \). If you do not want users to be able to see the priority, set its resource permission flag to \( i \). For the job or jobs that you wish to give top priority, use `qalter` to set the special resource to a value much larger than any formula outcome.

Example 3: To use a special priority:

\[
\text{sched\_priority} = \text{W\_prio} \times \text{wait\_secs} + \text{P\_prio} \times \text{priority} + \ldots + \text{special\_priority}
\]

Here, special\_priority is very large.

### 4.8.20.10 Caveats and Error Messages

- It is invalid to set both `job\_sort\_formula` and `job\_sort\_key` at the same time. If they are both set, `job\_sort\_key` is ignored and the following error message is logged:
  
  "Job sorting formula and job\_sort\_key are incompatible. The job sorting formula will be used."

- If the formula overflows or underflows the sorting behavior is undefined.

- If you set the formula to an invalid formula, `qmgr` will reject it, with one of the following error messages:
  
  "Invalid Formula Format"
  "Formula contains invalid keyword"
  "Formula contains a resource of an invalid type"

- If an error is encountered while evaluating the formula, the formula evaluates to zero for that job, and the following message is logged at event class 0x0100:
  
  "1234.mars;Formula evaluation for job had an error. Zero value will be used"

- The job sorting formula must be set via `qmgr` at the server host.

- When a job is moved to a new server or queue, it will inherit new default resources from that server or queue. If it is moved to a new server, it will be prioritized according to the formula on that server, if one exists.

- If the job is moved to another server through peer scheduling and the pulling server uses queue priority in its job sorting formula, the queue
priority used in the formula will be that of the queue to which the job is
moved.

- If you are using FIFO scheduling, the job_sort_formula server
  attribute must be unset.

- If you are using eligible time in the formula, and eligible_time_enable
  is False, each job’s eligible time evaluates to zero in the formula.

- If a job is requeued, and you are using the formula, the job may lose its
  place, because various factors may affect the job’s priority. For exam-
  ple, a higher-priority job may be submitted between the time the job is
  requeued and the time it would have run, or another job’s priority may
  be increased due to changes in which jobs are running or waiting.

- If the formula is configured, it is in force during both primetime and
  non-primetime.

- If the job sorting formula is defined, it overrides fairshare.

### 4.8.20.11 Logging

For each job, the evaluated formula answer is logged at the highest log
event class (0x0400):

“Formula Evaluation = <answer>”

### 4.8.21 Gating Jobs at Server or Queue

You can set resource limits at the server and queues so that jobs must con-
form to the limits in order to be admitted. This way, you can reject jobs
that request more of a resource than the complex or a queue can supply.

You can also force jobs into specific queues where they will inherit the
desired values for unrequested or custom resources. You can then use these
resources to manage jobs, for example by using the resources in the job
sorting formula or to route jobs to particular vnodes.

You can either force users to submit their jobs to specific queues, or you
can have users submit jobs to routing queues, and then route the jobs to the
desired queues.

For information on using resources for gating, see section 5.13 “Using
Resources to Restrict Server, Queue Access” on page 351.
For a description of which resources can be used for gating, see section 2.2.6.4.2 “Resources Used for Routing and Admittance” on page 29.

For how queue resource limits are applied to jobs, see section 2.2.6.4.1 “How Queue Limits Are Applied” on page 28.

For how routing queues work, see section 2.2.6 “Routing Queues” on page 26.

For how to route jobs to particular vnodes, see section 4.8.2 “Associating Vnodes with Queues” on page 141.

For how to use resources in the job sorting formula, see section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

### 4.8.21.1 Gating Caveats

For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. See section 5.9.3.7 “Using Gating Values As Defaults” on page 341.

### 4.8.22 Managing Application Licenses

PBS does not check application licenses out from the license server. PBS has no direct control over application licenses. However, you can have the scheduler use a dynamic resource to track application license use. This way, the scheduler knows how many application licenses are available, and how many have been checked out. For how to configure dynamic resources to represent application licenses, see section 5.14.7 “Supplying Application Licenses” on page 386.

Unfortunately, some jobs or applications don’t check out all of the application licenses they use until they have been running for some time. For example, job J1, which requests licenses, starts running, but doesn’t check them out for a few minutes. Next, the scheduler considers job J2, which also requests licenses. The scheduler runs its query for the number of available licenses, and the query returns with a sufficient number of licenses to run J2, so the scheduler starts J2. Shortly afterward, J1 checks out licenses, leaving too few to run J2.
It might appear that you could track the number of application licenses being used with a static integer PBS resource, and force jobs requesting application licenses to request this resource as well, but there is a drawback: if a job that has requested this resource is suspended, its static resources are released, but its application licenses are not. In this case you could end up with a deceptively high number for available licenses.

You can limit the number of jobs that request application licenses, if you know how many jobs can run at one time:

- Create a custom server-level consumable integer resource to represent these jobs. See section 5.14.4 “Configuring Server-level Resources” on page 370.
- Use `qmgr` to set `resources_available.<job limit>` at the server to the number of jobs that can run at one time.
- Force all jobs requesting the application to request one of these. See section 12.2 “Allocating Resources to Jobs” on page 842.

### 4.8.23 Limits on Per-job Resource Usage

You can specify how much of each resource any job is allowed to request, at the server and queue level. The server and queues each have per-job limit attributes. The `resources_min.<resource>` and `resources_max.<resource>` server and queue attributes are limits on what each individual job may request.

See section 5.15.3 “Placing Resource Limits on Jobs” on page 436, and section 5.13 “Using Resources to Restrict Server, Queue Access” on page 351.

### 4.8.24 Limits on User and Group Jobs

You can manage the number of jobs being run by users or groups, at the server or queue level. For example, you can limit the number of jobs enqueued in queue QueueA by any one group to 30, and by any single user to 5.

See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.
4.8.25 Limits on User and Group Resource Usage

You can manage the total amount of each resource that is used by users or groups, at the server or queue level. For example, you can manage how much memory is being used by jobs in queue QueueA.

See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.

4.8.26 Limits on Jobs at Vnodes

You can set limits on the number of jobs that can be run at each vnode by users, by groups, or overall. See section 5.15.2 “Limiting Number of Jobs at Vnode” on page 435.

4.8.27 Using Load Balancing

PBS can track the load on each execution host, running new jobs on the host according to the load on the host. You can specify that PBS does this for all machines in the complex. This is somewhat different behavior from that used for managing the load on vnodes; when managing load levels on vnodes, the scheduler only pays attention to the state of the vnode, and does not calculate whether a job would put the vnode over its load limit. Managing load levels on vnodes does not require load balancing to be turned on. See section 10.4.5 “Managing Load Levels on Vnodes” on page 768.

You use the load_balancing scheduler parameter to control whether PBS tracks the load on each host.

The load_balancing parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.
4.8.27.1 How Load Average is Computed

When load balancing is on, the scheduler queries each MOM once each scheduling cycle for the MOM’s load. MOM checks the load average on her host every 10 seconds.

The load used by MOM is the following:

- On UNIX/Linux, it is the raw one-minute averaged “loadave” returned by the operating system
- On Windows, it is based on the processor queue length

When a new load is added to a vnode, the load average increases slowly over time, so that more jobs than you want may be started at first. Eventually, the load average matches the actual load. If this is above the limit, PBS won’t start any more jobs on that vnode. As jobs terminate, the load average slowly moves down, and it takes time before the vnode is chosen for new jobs.

Consult your OS documentation to determine load values that make sense.

MOM sets the load only on the natural vnode, so it is the same for all vnodes on a multi-vnode machine.

4.8.27.2 How PBS Uses Load Information

When choosing vnodes for a job, the scheduler considers the load on the vnode in addition to whether the vnode can supply the resources specified in the job’s Resource_List attribute.

PBS estimates that a 1-CPU job will produce one CPU’s worth of load. This means that if you have a 2-CPU machine whose load is zero, PBS will put two 1-CPU jobs, or one 2-CPU job, on that machine.

When using load balancing, if a vnode has gone above $max_load, PBS does not run new jobs on the vnode until the load drops below $ideal_load.

MOM sets the vnode’s state according to its load. When a vnode’s load goes above $max_load, MOM marks the vnode busy. When the load drops below $ideal_load, MOM marks the vnode free. When a vnode’s state changes, for example from free to busy, MOM informs the server.
When using load balancing, PBS does not run new jobs on vnodes under the following conditions:

- Vnodes that are marked *busy*
- Vnodes whose resources, such as *ncpus*, are already fully allocated
- Vnodes that are above $\text{max}_{}\text{load}$
- Vnodes where running the job would cause the load to go above $\text{max}_{}\text{load}$

### 4.8.27.3 When to Use Load Balancing

When using load balancing (meaning the `load_balancing` scheduler parameter is *True*), the only changes to behavior are the following:

- The scheduler won’t place a job on a vnode whose load is above $\text{max}_{}\text{load}$
- The scheduler won’t place a job on a vnode where that job would put the load above $\text{max}_{}\text{load}$

Load balancing is useful when you want to oversubscribe CPUs, managing job placement by load instead. This can help when you want to run lots of jobs where each job will need only some CPU time, and the average load on the machine will be reasonable.

### 4.8.27.4 Suspending Jobs on Overloaded Vnodes

You can specify that MOM should suspend jobs when the load goes above $\text{max}_{}\text{load}$, by adding the `suspend` argument to the $\text{max}_{}\text{load}$ parameter. See section “$\text{max}_{}\text{load} <\text{load}> [\text{suspend}]$” on page 221. In this case, MOM suspends all jobs on the vnode until the load drops below $\text{ideal}_{}\text{load}$, then resumes them. This option is useful only when the source of the load is not strictly PBS jobs. This option is not recommended when the load is due solely to PBS jobs, because it can lead to the vnode cycling back and forth between becoming overloaded, being marked busy, suspending all jobs, being marked free, then starting all jobs, becoming overloaded, and so on.
4.8.27.5 Configuring Load Balancing

If you want to oversubscribe CPUs, set the value of `ncpus` on the vnode to the desired higher value.

We recommend setting the value of `$max_load` to a slightly higher value than the desired load, for example `.25 + ncpus`. Otherwise, the scheduler will not schedule jobs onto the last CPU, because it thinks a 1-CPU job will raise the load by 1, and the machine probably registers a load above zero.

To configure load balancing, perform the following steps:

1. Turn on load balancing by setting the `load_balancing` scheduler parameter to `True`:
   ```
   load_balancing: True ALL
   ```

2. Choose whether you want load balancing during primetime, non-primetime, or all. If you want separate behavior for primetime and non-primetime, specify each separately. The default is both. Example of separate behavior:
   ```
   load_balancing True prime
   load_balancing False non_prime
   ```

3. Set the ideal and maximum desired load for each execution host, by specifying values for `$ideal_load` and `$max_load` in each execution host’s MOM configuration file:
   ```
   $ideal_load <value at which to start new jobs>
   $max_load <value at which to cease starting jobs>
   ```

4. Set each host’s `resources_available.ncpus` to the maximum number of CPUs you wish to allocate on that host. Follow the recommendations in section 3.5.2 “Choosing Configuration Method” on page 58.

4.8.27.6 Load Balancing Caveats and Recommendations

- When setting `ncpus` and `$max_load`, consider the ratio between the two. PBS won’t allocate more than the value of `resources_available.ncpus`, so you can use this value to keep the load average from getting too high.
- Make sure that load balancing does not interfere with communications.
Please read section 10.4.5 “Managing Load Levels on Vnodes” on page 768.

- Load balancing is incompatible with sorting vnodes on a key (node_sort_key) when sorting on a resource using the “unused” or “assigned” parameters. Load balancing will be disabled. See section 4.8.47 “Sorting Vnodes on a Key” on page 312.

- You can use load balancing with SMP cluster distribution, but smp_cluster_dist will behave as if it is set to pack. See section 4.8.41 “SMP Cluster Distribution” on page 300.

- We recommend setting the value of $max_load to a slightly higher value than the desired load, for example .25 + ncpus. Otherwise, the scheduler will not schedule jobs onto the last CPU, because it thinks a 1-CPU job will raise the load by 1, and the machine probably registers a load above zero.

- If you are using cycle harvesting via load balancing, make sure your load balancing settings do not interfere with cycle harvesting. Be careful with the settings for $ideal_load and $max_load. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MOM to report being busy, and that PBS does not start any new jobs while the user is working. Please read section 4.8.9.5 “Cycle Harvesting Based on Load Average” on page 164.

- Using load balancing with multi-vnoded machines is not supported. MOM sets the load average only on the natural vnode, so all vnodes on a multi-vnoded machine are given the same value regardless of their actual load.

- It is not recommended to specify that MOM should suspend jobs when the load goes above $max_load. See section 4.8.27.4 “Suspending Jobs on Overloaded Vnodes” on page 218.

- Load level tracking is not supported for the HPCBP MOM.

- If you configure both placement sets and load balancing, the net effect is that vnodes that are over their load limit will be removed from consideration.
4.8.27.7 Parameters Affecting Load Balancing

$ideal_load <load>
MOM parameter. Defines the load below which the vnode is not considered to be busy. Used with the $max_load parameter. Not supported in the HPCBP MOM.

Example:
$ideal_load 1.8

Format: Float
No default

$max_load <load> [suspend]
MOM parameter. Defines the load above which the vnode is considered to be busy. Used with the $ideal_load parameter.

If the optional suspend argument is specified, PBS suspends jobs running on the vnode when the load average exceeds $max_load, regardless of the source of the load (PBS and/or logged-in users).

Not supported in the HPCBP MOM.

Example:
$max_load 3.5

Format: Float
Default: number of CPUs

load_balancing <T|F> [time slot specification]
Scheduler parameter. When set to True, the scheduler takes into account the load average on vnodes as well as the resources listed in the resources: line in sched_config. See “load_balancing” on page 334 of the PBS Professional Reference Guide.

Format: Boolean
Default: False all
4.8.28 Matching Jobs to Resources

The scheduler places each job where the resources requested by the job are available. The scheduler handles built-in and custom resources the same way. For a complete description of PBS resources, see Chapter 5, "PBS Resources", on page 317.

4.8.28.1 Scheduling on Consumable Resources

The scheduler constrains the use of a resource to the value that is set for that resource in resources_available.<resource>. For a consumable resource, the scheduler won’t place more demand on the resource than is available. For example, if a vnode has resources_available.ncpus set to 4, the scheduler will place jobs requesting up to a total of 4 CPUs on that vnode, but no more.

The scheduler computes how much of a resource is available by subtracting the total of resources_assigned.<resource> for all running jobs from resources_available.<resource>.

4.8.28.2 Scheduling on Non-Consumable Resources

For non-consumable resources such as arch or host, the scheduler matches the value requested by a job with the value at one or more vnodes. Matching a job this way does not change whether or not other jobs can be matched as well; non-consumable resources are not used up by jobs, and therefore have no limits.

4.8.28.3 Scheduling on Dynamic Resources

At each scheduling cycle, the scheduler queries each dynamic resource. If a dynamic resource is not under the control of PBS, jobs requesting it may run in an unpredictable fashion.
4.8.28.4 Unrequestable or Invisible Resources

You can define custom resources that are invisible to and unrequestable by users, or simply unrequestable by users. The scheduler treats these resources the same as visible, requestable resources. See section 5.14.2.5 “Resource Permission Flags” on page 362.

4.8.28.5 Enforcing Scheduling on Resources

The scheduler chooses which resources to schedule on according to the following rules:

- The scheduler always schedules jobs based on the availability of the following vnode-level resources:
  - vnode
  - host
  - Any Boolean resource

- The scheduler will schedule jobs based on the availability of other resources only if those resources are listed in the "resources:" line in PBS_HOME/sched_priv/sched_config. Some resources are automatically added to this line. You can add resources to this line. The following resources are automatically added to this line:
  - aoe
  - arch
  - host
  - mem
  - ncpus
  - netwins
  - vnode
4.8.28.6 Matching Unset Resources

When job resource requests are being matched with available resources, unset resources are treated as follows:

- A numerical resource that is unset on a host is treated as if it were zero
- An unset resource on the server or queue is treated as if it were infinite
- An unset string cannot be matched
- An unset Boolean resource is treated as if it were set to False.
- The resources ompthreads, mpiprocs, and nodes are ignored for unset resource matching.

The following table shows how a resource request will or won’t match an unset resource at the host level.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Unset Resource</th>
<th>Matching Request Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>float</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>long</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>size</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>string</td>
<td>Never matches</td>
<td></td>
</tr>
<tr>
<td>string array</td>
<td>Never matches</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>0, 0:0, 0:0.0, 0:0:0</td>
<td>0, 0:0, 0:0.0, 0:0:0</td>
</tr>
</tbody>
</table>

4.8.28.6.1 When Dynamic Resource Script Fails

If a server dynamic resource script fails, the scheduler uses the value of resources_available.<resource>. If this was never set, it is treated as an unset resource, described above.

If a host-level dynamic resource script fails, the scheduler treats the resource as if its value is zero.
4.8.28.6.2 Backward Compatibility of Unset Resources

To preserve backward compatibility, you can set the server’s resource_unset_infinite attribute with a list of host-level resources that will behave as if they are infinite when they are unset. See “resource_unset_infinite” on page 342 of the PBS Professional Reference Guide for information on resource_unset_infinite.

4.8.28.7 Resource Scheduling Caveats

- Do not set values for resources such as walltime at the server or a queue, because the scheduler will not allocate more than the specified value. This means that if you set resources_available.walltime at the server to 10:00:00, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.
- Jobs may be placed on different vnodes from those where they would have run in earlier versions of PBS. This is because a job’s resource request will no longer match the same resources on the server, queues and vnodes.
- Beware of application license race conditions. If two jobs require the same application license, the first job may be started, but may not get around to using the license before the second job is started and uses the license. The first job must then wait until the license is available, taking up resources. The scheduler cannot avoid this problem.

4.8.29 Node Grouping

The term “node grouping” has been superseded by the term “placement sets”. Vnodes were originally grouped according to the value of one resource, so for example all vnodes with a value of linux for arch were grouped together, and all vnodes with a value of solaris for arch were in a separate group. We use placement sets now because this means grouping vnodes according to the value of one or more resources. See section 4.8.32 “Placement Sets” on page 237.
4.8.29.1 Configuring Old-style Node Grouping

Configuring old-style node grouping means that you configure the simplest possible placement sets. In order to have the same behavior as in the old node grouping, group on a single resource. If this resource is a string array, it should only have one value on each vnode. This way, each vnode will be in only one node group.

You enable node grouping by setting the server’s node_group_enable attribute to True.

You can configure one set of vnode groups for the entire complex by setting the server’s node_group_key attribute to a resource name.

You can configure node grouping separately for each queue by setting that queue’s node_group_key attribute to a resource name.

4.8.30 Overrides

You can use various overrides to change how one or more jobs run.

4.8.30.1 Run a Job Manually

You can tell PBS to run a job now, and you can optionally specify where to run it. You run a job manually using the qrun command. The -H option to the qrun command makes an important difference:

qrun

When preemption is enabled, the scheduler preempts other jobs in order to run this job. Running a job via qrun gives the job higher preemption priority than any other class of job, except for reservation jobs.

When preemption is not enabled, the scheduler runs the job only if enough resources are available.

qrun -H

PBS runs the job regardless of scheduling policy and available resources.

4.8.30.1.1 qrun Caveats

Be careful when using qrun \(-H\) on jobs or vnodes involved in reservations.

4.8.30.2 Hold a Job Manually

You can use the qhold command to place a hold on a job. The effect of placing a hold depends on whether the job is running and whether you have checkpointing configured:

- If the job is queued, the job will not run.
- If the job is running and checkpoint-abort is configured, the job is checkpointed, requeued, and held.
- If the job is running and checkpoint-abort is not configured, the only change is that the job’s Hold_Types attribute is set to User Hold. If the job is subsequently requeued, it will not run until the hold is released.

You can release the hold using the qrls command.

For information on checkpointing jobs, see section 10.3 “Checkpoint and Restart” on page 735.

See “qhold” on page 175 of the PBS Professional Reference Guide and “qrls” on page 208 of the PBS Professional Reference Guide.

4.8.30.3 Suspend a Job Manually

You can use the qsig \(-s\) suspend command to suspend a job so that it won’t run. If you suspend a job, and then release it using the qsig \(-s\) resume command, the job remains in the suspended state until the required resources are available.

You can resume the job immediately by doing the following:

1. Resume the job:
   \[qsig -s resume <job ID>\]
2. Run the job manually:
   \[qrun <job ID>\]
Chapter 4  Scheduling

See “qsig” on page 223 of the PBS Professional Reference Guide.

4.8.30.4 Set Special Resource Value Used in Formula

You can change the value of a resource used in the job sorting formula. For example, to give a particular job a higher priority by changing the value of a custom resource called “higher”:

- Create a custom resource that is invisible to job submitters in resourcedef:
  
  ```
  higher type=float flag=i
  ```

- Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

- The formula expression includes “higher”:
  
  ```
  Qmgr: s s job_sort_formula = "higher"
  ```

- Set the default for this resource at the server:

  ```
  Qmgr: set server resources_default.higher = 1
  ```

- These jobs are submitted:

  Job 1
  Job 2
  Job 3

- Change Job 2 so that its value for “higher” is 5:

  ```
  qalter -l higher = 5 job2
  ```

- The scheduler logs the following:

  ```
  Job ;1.host1;Formula Evaluation = 1
  Job ;2.host1;Formula Evaluation = 5
  Job; 3.host1;Formula Evaluation = 1
  ```

- Jobs are sorted in this order:

  Job 2
  Job 1
  Job 3
4.8.30.5 Change Formula On the Fly

You can change the job sorting formula on the fly, so that the next scheduler iteration uses your new formula. This will change how job priorities are computed, and can rearrange the order in which jobs are run. See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

4.8.30.6 Using Dedicated Time

You can set up blocks of dedicated time, where the only jobs eligible to be started or running are the ones in dedicated time queues. You can use dedicated time for upgrades. See section 4.8.10 “Dedicated Time” on page 171, and section 2.2.5.2.1 “Dedicated Time Queues” on page 25.

4.8.30.7 Using crontab Jobs or the Windows Task Scheduler

You can use crontab jobs or the Windows Task Scheduler to change PBS settings according to the needs of your time slots. See section 4.8.7 “cron Jobs, or the Windows Task Scheduler” on page 155.

4.8.30.8 Using Hooks

You can use hooks to examine jobs and alter their characteristics. See “Hooks” on page 461.

4.8.31 Peer Scheduling

Peer scheduling allows separate PBS complexes to automatically run jobs from each other’s queues. This means that you can dynamically balance the workload across multiple, separate PBS complexes. These cooperating PBS complexes are referred to as “Peers”.

4.8.31.1 How Peer Scheduling Works

In peer scheduling, a PBS server pulls jobs from one or more peer servers and runs them locally. When Complex A pulls a job from Complex B, Complex A is the “pulling” complex and Complex B is the “furnishing”
complex. When the pulling scheduler determines that another complex’s job can immediately run locally, it moves the job to the specified queue on the pulling server and immediately run the job. The job is run as if it had been submitted to the pulling complex.

You can set up peer scheduling so that A pulls from B and C, and so that B also pulls from A and C.

A job is pulled only when it can run immediately.

The pulling complex must have all of the resources required by the job, including custom resources.

When a job is pulled from one complex to another, the pulling complex applies its policy to the job. The job’s execution priority is determined by the policy of the pulling complex. You can set special priority for pulled jobs; see section 4.8.31.4.2 “Setting Priority for Pulled Jobs” on page 234.

4.8.31.2 Prerequisites for Peer Scheduling

- You must create the pulling and furnishing queues before peer scheduling can be configured. See section 2.2.3 “Creating Queues” on page 22 on how to create queues.

- When configuring peer scheduling, it is strongly recommended to use the same version of PBS Professional at all peer locations.

- Make sure that custom resources are consistent across peer locations. Jobs requesting custom resources at one location will not be able to run at another unless the same resources are available.

- Under Windows, if single_signon_password_enable is set to True among all peer servers, users must have their password cached on each server. See section 9.13.1.1 “Per-user/per-server Passwords” on page 699.
4.8.31.3 Configuring Peer Scheduling

The following sections give details on how to configure peer scheduling. Here is a brief outline:

- Define a flat user namespace on all complexes
- Map pulling queues to furnishing queues
- If necessary, specify port
- Grant manager access to each pulling server
- If possible, make user-to-group mappings be consistent across complexes
- If any of the peering sites is using failover, configure peering to work with failover

4.8.31.3.1 Defining a Flat User Namespace

Peer scheduling requires a flat user namespace in all complexes involved. This means that user “joe” on the remote peer system(s) must be the same as user “joe” on the local system. Your site must have the same mapping of user to UID across all hosts, and a one-to-one mapping of UIDs to user names. It means that PBS does not need to check whether X@hostA is the same as X@hostB; it can just assume that this is true. Set flatuid to *True*:

```
Qmgr: set server flatuid = True
```

For more on flatuid, see section 9.3.13 “Flatuid and Access” on page 681.

4.8.31.3.2 Mapping Pulling Queues to Furnishing Queues

You configure peer scheduling by mapping a furnishing peer’s queue to a pulling peer’s queue. You can map each pulling queue to more than one furnishing queue, or more than one pulling queue to each furnishing queue.

The pulling and furnishing queues must be *execution* queues, not route queues. However, the queues can be either ordinary queues that the complex uses for normal work, or special queues set up just for peer scheduling.
You map pulling queues to furnishing queues by setting the peer_queue scheduler configuration option in PBS_HOME/sched_priv/sched_config. The format is:

```
peer_queue: "<pulling queue> <furnishing queue>@<furnishing server>.domain"
```

For example, Complex A’s queue “workq” is to pull from two queues: Complex B’s queue “workq” and Complex C’s queue “slowq”. Complex B’s server is ServerB and Complex C’s server is ServerC. You would add this to Complex A’s PBS_HOME/sched_priv/sched_config:

```
peer_queue: "workq workq@ServerB.domain.com"
peer_queue: "workq slowq@ServerC.domain.com"
```

Or if you wish to direct Complex B’s jobs to queue Q1 on Complex A, and Complex C’s jobs to Q2 on Complex A:

```
peer_queue: "Q1 workq@ServerB.domain.com"
peer_queue: "Q2 fastq@ServerC.domain.com"
```

In one complex, you can create up to 50 mappings between queues. This means that you can have up to 50 lines in PBS_HOME/sched_priv/sched_config beginning with “peer_queue”.

### 4.8.31.3.3 Specifying Ports

The default port for the server to listen on is 15001, and the scheduler uses any privileged port (1023 and lower). If the furnishing server is not using the default port, you must specify the port when you specify the queue. For example, if ServerB is using port 16001, and you wish to pull jobs from workq at ServerB to workq at ServerA, add this to PBS_HOME/sched_priv/sched_config at ServerA:

```
peer_queue: "workq workq@ServerB.domain.com:16001"
```

The scheduler and server communicate via TCP.

### 4.8.31.3.4 Granting Manager Access to Pulling Servers

Each furnishing server must grant manager access to each pulling server. If you wish jobs to move in both directions, where Complex A will both pull from and furnish jobs to Complex B, ServerA and ServerB must grant manager access to each other.
On the furnishing complex:

For UNIX:

```
Qmgr: set server managers +=
    root@pullingServer.domain.com
```

For Windows:

```
Qmgr: set server managers += <name of PBS service account>@*
```

4.8.31.3.5 Making User-to-group Mappings Consistent Across Complexes

If possible, ensure that for each user in a peer complex, that user is in the same group in all participating complexes. So if user “joe” is in groupX on Complex A, user “joe” should be in groupX on Complex B. This means that a job’s `egroup` attribute will be the same on both complexes, and any group limit enforcement can be properly applied.

There is a condition when using peer scheduling in which group hard limits may not be applied correctly. This can occur when a job’s effective group, which is its `egroup` attribute, i.e. the job’s owner’s group, is different on the furnishing and pulling systems. When the job is moved over to the pulling complex, it can evade group limit enforcement if the group under which it will run on the pulling system has not reached its hard limit. The reverse is also true; if the group under which it will run on the pulling system has already reached its hard limit, the job won’t be pulled to run, although it should.

This situation can also occur if the user explicitly specifies a group via `qsub -W group_list`.

It is recommended to advise users to *not* use the `qsub` options “-u user_list” or “-W group_list=groups” in conjunction with peer scheduling.
4.8.31.3.6 Configuring Peer Scheduling with Failover

If you are configuring peer scheduling so that Complex A will pull from Complex B where Complex B is configured for failover, you must configure Complex A to pull from both of Complex B’s servers. For these instructions, see section 10.2.6.2 “Configuring Failover to Work With Peer Scheduling” on page 731.

4.8.31.4 Peer Scheduling Advice

4.8.31.4.1 Selective Peer Scheduling

You can choose the kinds of jobs that can be selected for peer scheduling to a different complex. You can do the following:

- Set resource limits at the furnishing queue via the `resources_min` and `resources_max` queue attributes. See section 2.2.6.4 “Using Resources to Route Jobs Between Queues” on page 28.
- Route jobs into the furnishing queue via a hook. See section 6.4.1.1 “Routing Jobs Using Hooks” on page 465.
- Route jobs into the furnishing queue via a routing queue. See section 2.2.6 “Routing Queues” on page 26.

4.8.31.4.2 Setting Priority for Pulled Jobs

You can set a special priority for pulled jobs by creating a queue that is used only as a pulling queue, and setting the pulling queue’s priority to the desired level. You can then use the queue’s priority when setting job execution priority. See section 4.2.5.3.4 “Using Queue Priority when Computing Job Priority” on page 79.

For example, if you give the pulling queue the lowest priority, the pulling complex will pull a job only when there are no higher-priority jobs that can run.

You can also have pulled jobs land in a special queue where they inherit a custom resource that is used in the job sorting formula.
4.8.31.5 How Peer Scheduling Affects Jobs

4.8.31.5.1 How Peer Scheduling Affects Inherited Resources

If the job is moved from one server to another via peer scheduling, any default resources in the job’s resource list inherited from the furnishing queue or server are removed. This includes any select specification and place directive that may have been generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

When the pulling scheduler runs the job the first time, the job is run as if the job still had all of the resources it had at the furnishing complex. If the job is requeued and restarted at the pulling complex, the job picks up new default resources from the pulling complex, and is scheduled according to the newly-inherited resources from the pulling complex.

4.8.31.5.2 How Peer Scheduling Affects Policy Applied to Job

After a job is pulled from one complex to another, the scheduling policy of the pulling complex is applied to the job.

For example, if you use queue priority in the formula and the job is moved to another server through peer scheduling, the queue priority used in the formula will be that of the queue to which the job is moved.

When a job is pulled from one complex to another, hooks are applied at the new complex as if the job had been submitted locally. For example, if the pulling complex has a queuejob hook, that hook runs when a job is pulled.

4.8.31.5.3 How Peer Scheduling Affects Job’s Eligible Time

The job’s eligible_time is preserved when a job is moved due to peer scheduling.
4.8.31.5.4 Viewing Jobs That Have Been Moved to Another Server

If you are connected to ServerA and a job submitted to ServerA has been moved from ServerA to ServerB through peer scheduling, in order to display it via `qstat`, give the job ID as an argument to `qstat`. If you only give the `qstat` command, the job will not appear to exist. For example, the job `123.ServerA` is moved to ServerB. In this case, use

```
qstat 123
```

or

```
qstat 123.ServerA
```

To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

4.8.31.5.5 Peer Scheduling and Hooks

When a job is pulled from one complex to another, the following happens:

- Hooks are applied at the new complex as if the job had been submitted locally
- Any `movejob` hooks at the furnishing server are run

4.8.31.6 Peer Scheduling Caveats

- Each complex can peer with at most 50 other complexes.
- When using peer scheduling, group hard limits may not be applied correctly. This can occur when the job owner’s group is different on the furnishing and pulling systems. For help in avoiding this problem, see section 4.8.31.3.5 “Making User-to-group Mappings Consistent Across Complexes” on page 233.
- You cannot peer schedule between a Windows complex and a UNIX/Linux complex.
- When the pulling scheduler runs the job the first time, the job is run as if the job still had all of the resources it had at the furnishing complex. If the job is requeued and restarted at the pulling complex, the job picks up new default resources from the pulling complex, and is scheduled according to the newly-inherited resources from the pulling complex.
4.8.32 Placement Sets

Placement sets are the sets of vnodes within which PBS will try to place a job. PBS tries to group vnodes into the most useful sets, according to how well connected the vnodes are, or the values of resources available at the vnodes. Placement sets are used to improve task placement (optimizing to provide a “good fit”) by exposing information on system configuration and topology. The scheduler tries to put a job in the smallest appropriate placement set.

4.8.32.1 Definitions

Task placement
The process of choosing a set of vnodes to allocate to a job that will satisfy both the job's resource request (select and place specifications) and the configured scheduling policy.

Placement Set
A set of vnodes. Placement sets are defined by the values of vnode-level string array resources. A placement set is all of the vnodes that have the same value for a specified defining resource substring. For example, if the defining resource is a vnode-level string array named “switch”, which can have values “S1”, “S2”, or “S3”: the set of vnodes which have a substring matching “switch=S2” is a placement set.

Placement sets can be specified at the server or queue level.

Placement Set Series
A set of sets of vnodes.

A placement set series is all of the placement sets that are defined by specifying one string array resource. Each placement set in the series is the set of vnodes that share one value for the resource. There is one placement set for each value of the resource. If the resource takes on N values at the vnodes, then there are N sets in the series. For example, if the defining resource is a string array named “switch”, which can have values “S1”, “S2”, or “S3”, there are three sets in the series. The first is defined by the value “S1”, where all the vnodes in that set have the value “S1” for the
resource switch. The second set is defined by “S2”, and the third by “S3”.

Each of the resources named in node_group_key specifies a placement series. For example, if the server’s node_group_key attribute contains “router,switch”, then the server has two placement set series.

**Placement Pool**

All of the placement sets that are defined; the server can have a placement pool, and each queue can have its own placement pool. If a queue has no placement pool, the scheduler uses the server’s placement pool.

A placement pool is the set of placement set series that are defined by one or more string array resources named in node_group_key.

For example, if the server’s node_group_key attribute contains “router,switch”, and router can take the values “R1” and “R2” and switch can take the values “S1”, “S2”, and “S3”, then there are five placement sets, in two placement series, in the server’s placement pool.

**Static Fit**

A job statically fits into a placement set if the job could fit into the placement set if the set were empty. It might not fit right now with the currently available resources.

**Dynamic Fit**

A job dynamically fits into a placement set if it will fit with the currently available resources (i.e. the job can fit right now).

**4.8.32.2 Requirements for Placement Sets**

• Placement sets are enabled by setting the server’s node_group_enable attribute to True

• Server-level placement sets are defined by setting the server’s node_group_key attribute to a list of vnode-level string array resources.

• Queue-level placement sets are defined by setting a queue’s node_group_key attribute to a list of vnode-level string array resources.
resources.

- At least one vnode-level string array resource must exist on vnodes and be set to values that can be used to partition the vnodes.

### 4.8.32.3 Description of Placement Sets

#### 4.8.32.3.1 What Defines a Placement Set, Series, or Pool

Placement sets are defined by the values of vnode-level string array resources. You define placement sets by specifying the names of these resources in the `node_group_key` attribute for the server and/or queues. Each value of each resource defines a different placement set. A placement set is all of the vnodes that have the same value for a specified defining resource. For example, if the defining resource is a vnode-level string array named “switch”, which has the values “S1”, “S2”, and “S3”, the set of vnodes where switch has the value “S2” is a placement set. If some vnodes have more than one substring, and one of those substrings is the same in each vnode, those vnodes make up a placement set. For example, if the resource is “router”, and vnode V0 has `resources_available.router` set to “r1i0,r1”, and vnode V1 has `resources_available.router` set to “r1i1,r1”, V0 and V1 are in the placement set defined by `resources_available.router = “r1”`. If the resource has \( N \) distinct values across the vnodes, including the value zero and being unset, there are \( N \) placement sets defined by that resource.

Each placement set can have a different number of vnodes; the number of vnodes is determined only by how many vnodes share that resource value.

Each placement set series is defined by the values of a single resource across all the vnodes. For example, if there are three switches, S1, S2 and S3, and there are vnodes with `resources_available.switch` that take on one or more of these three values, then there will be three placement sets in the series.

Whenever you define any placement sets, you are defining a placement pool. Placement pools can be defined for the server and for each queue. You define a server-level placement pool by setting the server’s `node_group_key` to a list of one or more vnode-level string array resources. You define a queue-level placement pool by similarly setting the queue’s `node_group_key`. 

---

Scheduling

Chapter 4

---

PBS Professional 11.1 Administrator’s Guide 239
4.8.32.3.2 Vnode Participation in Placement Sets, Series, and Pools

Each vnode can be in multiple placement sets, placement set series, and placement pools.

A vnode can be in multiple placement sets in the same placement set series. For example, if the resource is called “router”, and a vnode’s router resource is set to “R1, R2”, then the vnode will be in the placement set defined by router = R1 and the set defined by router = R2.

A vnode is in a placement series whenever the resource that defines the series is defined on the vnode. For example, if placement sets are defined by the values of the router and the switch resources, and a vnode has value R1 for router, and S1 for switch, then the vnode is in both placement series, because it is in the set that shares the R1 value for router, and the set that shares the S1 value for switch. Each of those sets is one of a different series.

The server has its own placement pool if the server’s node_group_key attribute is set to at least one vnode-level string array resource. Similarly, each queue can have its own placement pool. A vnode can be in any placement pool that specifies a resource that is defined on the vnode.

4.8.32.3.3 Multihost Placement Sets

Placement sets, series, and pools can span hosts. Placement sets can be made up of vnodes from anywhere, regardless of whether the vnode is from a multi-vnode host.

To set up a multihost placement set, choose a string array resource for the purpose, and list it in the desired node_group_key attribute. For example, create a string_array resource called “span” in the PBS_HOME/server_priv/resourcedef file:

```
span type=string_array flag=h
```

Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367. Add the resource “span” to node_group_key on the server or queue. Use qmgr to give it the same value on all the desired vnodes. You can write a script that sets the same value on each vnode that you want in your placement set.
4.8.32.3.4 Machines with Multiple Vnodes

Machines with multiple vnodes such as the SGI Altix are represented as a generic set of vnodes. Placement sets are used to allocate resources on a single machine to improve performance and satisfy scheduling policy and other constraints. Jobs are placed on vnodes using placement set information.

PBS automatically generates the placement information for vnodes on the following machines:

- Shared-memory Altix running supported versions of ProPack
- Shared-memory Altix running the SGI Performance Suite
- SGI ICE
- SGI UV 1000

On an SGI UV 100, placement information is not automatically generated.

4.8.32.3.5 Placement Sets Defined by Unset Resources

For each defining resource, vnodes where that resource is unset are grouped into their own placement set. For example, if you have ten vnodes, on which there is a string resource COLOR, where two have COLOR set to “red”, two are set to “blue”, two are set to “green” and the rest are unset, there will be four placement sets defined by the resource COLOR. This is because the fourth placement set consists of the four vnodes where COLOR is unset. This placement set will also be the largest.

Every resource listed in node_group_key could potentially define such a placement set.

4.8.32.3.6 Placement Sets and Node Grouping

Node grouping is the same as one placement set series, where the placement sets are defined by a single resource. Node grouping has been superseded by placement sets.

In order to have the same behavior as in the old node grouping, group on a single resource. If this resource is a string array, it should only have one value on each vnode. This way, each vnode will only be in one node group.
4.8.32.4 How Placement Sets Are Used

You use placement sets to partition vnodes according to the value of one or more resources. Placement sets allow you to group vnodes into useful sets.

You can run multi-vnode jobs in one placement set. For example, it makes the most sense to run a multi-vnode job on vnodes that are all connected to the same high-speed switch.

PBS will attempt to place each job in the smallest possible set that is appropriate for the job.

4.8.32.4.1 Order of Placement Pool Selection

The scheduler chooses one placement pool from which to select a placement set. If the job cannot run in that placement pool, the scheduler ignores placement sets for the job.

Queue placement pools override the server’s placement pool. If a queue has a placement pool, jobs from that queue are placed using the queue’s placement pool. If a queue has no placement pool (the queue’s node_group_key is not defined), jobs are placed using the server’s placement pool, if it exists.

A per-job placement set is defined by the -l place statement in the job’s resource request. Since the job can only request one value for the resource, it can only request one specific placement set. A job’s place=group resource request overrides the sets defined by the queue’s or server’s node_group_key.

The scheduler chooses the most specific placement pool available, following this order of precedence:

1. A per-job placement set (job’s place=group= request)
2. A placement set from the placement pool for the job’s queue
3. A placement set from the complex-wide placement pool
4. All vnodes in the complex
4.8.32.4.2 Order of Placement Set Consideration Within Pool

The scheduler looks in the selected placement pool and chooses the smallest possible placement set that is appropriate for the job. The scheduler examines the placement sets in the pool and orders them, from smallest to largest, according to the following rules:

1. Static total \( \text{ncpus} \) of all vnodes in set
2. Static total \( \text{mem} \) of all vnodes in set
3. Dynamic free \( \text{ncpus} \) of all vnodes in set
4. Dynamic free \( \text{mem} \) of all vnodes in set

If a job can fit statically within any of the placement sets in the placement pool, then the scheduler places a job in the first placement set in which it fits dynamically. This ordering ensures the scheduler will use the smallest possible placement set in which the job will dynamically fit. If there are multiple placement sets where the job fits statically, but some are being used, the scheduler uses the first placement set where the job can run now. If the job fits statically into at least one placement set, but these placement sets are all busy, the scheduler waits until a placement set can fit the job dynamically.

If a job cannot statically fit into any placement set in the selected placement pool, the scheduler ignores defined placement sets and uses all available vnodes as its placement set, unless the \texttt{do\_not\_span\_psets} scheduler attribute is \texttt{True}, in which case the job will not run.

For example, we have the following placement sets, and a job that requests 8 CPUs:

- Set1 \( \text{ncpus} = 4 \)
- Set2 \( \text{ncpus} = 12 \); this placement set is full
- Set3 \( \text{ncpus} = 16 \); this placement set is not being used

The scheduler looks at Set1; Set1 is statically too small, and the scheduler moves to the next placement set. Set2 is statically large enough, but the job does not fit dynamically. The scheduler looks at Set3; Set3 is large enough, and the job fits dynamically. The scheduler runs the job in Set3.

If the job requests 24 CPUs, the scheduler attempts to run the job in the set consisting of all vnodes.
4.8.32.4.3 Order of Vnode Selection Within Set

The scheduler orders the vnodes within the selected placement set using the following rules:

- If `node_sort_key` is set, vnodes are sorted by `node_sort_key`. See section 4.8.47 “Sorting Vnodes on a Key” on page 312.
- If `node_sort_key` is not set, the order in which the vnodes are returned by `pbs_statnode()`. This is the default order the vnodes appear in the output of the `pbsnodes -a` command.

The scheduler places the job on the vnodes according to their ordering above.

4.8.32.5 Summary of Placement Set Requirements

The steps to configure placement sets are given in the next section. The requirements are summarized here for convenience:

- A `PBS_HOME/server_priv/resourcedef` file containing definitions of the resources of interest
  - You must restart the server after new definition
- Vnodes defining a value for each resource to be used for placement sets (e.g., rack)
  - If defined via vnode definition, you must HUP the MOMs involved
- The server’s or queue’s `node_group_key` attribute must be set to the resources to be used for placement sets. For example, if we have custom resources named “rack”, “socket”, “board”, and “boardpair”, which are to be used for placement sets:

  ```
  Qmgr: set server node_group_key = “rack,socket,board,boardpair”
  - No signals needed, takes effect immediately
  ```
- Placement sets must be enabled at the server by setting the server’s `node_group_enable` attribute to `True`. For example:

  ```
  Qmgr: set server node_group_enable=True
  - No signals needed, takes effect immediately
  ```
Adding a resource to the scheduler's `resources:` line is required only if the resource is to be specifically requested by jobs. It is not required for `-lplace=group=<resource>`.
4.8.32.6 How to Configure Placement Sets

The following steps show how to satisfy the requirements for placement sets:

1. If the vnodes that you will use in placement sets are not defined, define them. See section 3.1.5 “Creating Vnodes” on page 43.

2. If the vnode-level string array resources that you will use to define placement sets do not exist, create them. See section 5.14.5 “Configuring Host-level Custom Resources” on page 373.

3. Restart the server; see section 5.14.3.1 “Restarting the Server” on page 367.

4. If values for the vnode-level string array resources that you will use to define placement sets are not set at the vnodes you wish to use, set the values. See section 3.5 “How to Configure MOMs and Vnodes” on page 56.

5. If you use vnode definition files to set values for vnode-level string array resources, HUP the MOMs involved.

6. To create queue placement pools, set the node_group_key attribute to the name(s) of one or more vnode-level string array resources. Do this for each queue for which you want a separate pool. For example:

   **Qmgr:** set queue workq node_group_key = <router,switch>

7. To create a server placement pool, set the node_group_key server attribute to the name(s) of one or more vnode-level string array resources. For example:

   **Qmgr:** set server node_group_key = <router,switch>

   For example, to create a server-level placement pool for the resources host, L2 and L3:

   **Qmgr:** set server node_group_key = "host,L2,L3"

8. Set the server’s node_group_enable attribute to True

   **Qmgr:** set server node_group_enable = True

9. For ease of reviewing placement set information, you can add the name of each resource used in each vnode’s pnames attribute:
Qmgr: active node <vnode name>,<vnode name>,...
Qmgr: set node pnames += <resource name>

or

Qmgr: set node pnames = <resource list>

For example:

Qmgr: set node pnames =
   "board,boardpair,iruquadrant,iruhalf,irdu,rack"

We recommend using the natural vnode for any placement set information that is invariant for a given host.

Resources used only for defining placement sets, and not for allocation to jobs, do not need to be listed in the resources: line in PBS_HOME/sched_priv/sched_config. So for example if you create a resource just for defining placement sets, and jobs will not be requesting this resource, you do not need to list it in the resources: line.
4.8.32.7 Examples of Creating Placement Sets

4.8.32.7.1 Cluster with Four Switches

This cluster is arranged as shown with vnodes 1-4 on Switch1, vnodes 5-10 on Switch2, and vnodes 11-24 on Switch3. Switch1 and Switch2 are on Switch4.

To make the placement sets group the vnodes as they are grouped on the switches:

Create a custom resource called `switch`. The `-h` flag makes the resource requestable:

```
switch type=string_array flag=h
```
On vnodes[1-4] set:
   resources_available.switch="switch1,switch4"

On vnodes[5-10] set:
   resources_available.switch="switch2,switch4"

On vnodes[11-24] set:
   resources_available.switch="switch3"

On the server set:
   node_group_enable=True
   node_group_key=switch

So you have 4 placement sets:
   The placement set "switch1" has 4 vnodes
   The placement set "switch2" has 6 vnodes
   The placement set "switch3" has 14 vnodes
   The placement set "switch4" has 10 vnodes

PBS will try to place a job in the smallest available placement set. Does the job fit into the smallest set (switch1)? If not, does it fit into the next smallest set (switch2)? This continues until it finds one where the job will fit.

4.8.32.7.2 Examples of Configuring Placement Sets on an Altix

For information on how to configure vnodes on a cpusetted machine in order to define new placement sets on an Altix, use the instructions in section 3.5.2.3 “Configuring Machines with cpusets” on page 59.
In this example, we define a new placement set using the new resource “NewRes”. We create a file called SetDefs that contains the changes we want.

1. Add the new resource to the server’s resourcedef file:
   ```
   NewRes type=string_array flag=h
   ```

2. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

3. Add NewRes to the server's node_group_key
   ```
   Qmgr: set server node_group_key+="NewRes"
   ```

4. Add NewRes to the value of the pnames attribute for the natural vnode. This makes the name of the resource you used easily available. Add a line like this to SetDefs:
   ```
   altix3: resources_available.pnames = "...,NewRes"
   ```

5. For each vnode, V, that's a member of a new placement set you're defining, add a line of the form:
   ```
   V: resources_available.NewRes = "<value1[,...]>"
   ```
   All the vnodes in the new set should have lines of that form, with the same resource value, in the new configuration file.
   Here the value of the resource is “P” and/or “Q”.
   We’ll put vnodes A, B and C into one placement set, and vnodes B, C and D into another.
   ```
   A: resources_available.NewRes2 = P
   B: resources_available.NewRes2 = "P,Q"
   C: resources_available.NewRes2 = "P,Q"
   D: resources_available.NewRes2 = Q
   ```
   For each new placement set you define, use a different value for the resource.

6. Add SetDefs and tell MOM to read it, to make a Version 2 MOM configuration file NewConfig.
   ```
   pbs_mom -s insert NewConfig SetDefs
   ```

7. Stop and restart the MOM. See section 6.1 “Starting and Stopping
4.8.32.7.3 Example of Altix Placement Pool

In this example, we have vnodes connected to four cbricks and two L2 connectors. Since these come from the MOM, they are automatically added to the server’s `resourcedef` file.

Enable placement sets:

```
Qmgr: s s node_group_enable=True
```

Define the pool you want:

```
Qmgr: s s node_group_key="cbrick, L2"
```

When you use the following:

```
pbsnodes -av | egrep '([^ ]*) | cbrick
```

or

```
pbsnodes -av | egrep '([^ ]*) | L2
```

and the vnodes look like this:

```
vnode1
  resources_available.cbrick=cbrick1
  resources_available.L2=A

vnode2
  resources_available.cbrick=cbrick1
  resources_available.L2=B

vnode3
  resources_available.cbrick=cbrick2
  resources_available.L2=A

vnode4
  resources_available.cbrick=cbrick2
  resources_available.L2=B

vnode5
  resources_available.cbrick=cbrick3
  resources_available.L2=A

vnode6
  resources_available.cbrick=cbrick3
  resources_available.L2=B

vnode7
  resources_available.cbrick=cbrick4
  resources_available.L2=A

vnode8
  resources_available.cbrick=cbrick4
  resources_available.L2=B
```
There are six resulting placement sets.

- cbrick=cbrick1: {vnode1, vnode2}
- cbrick=cbrick2: {vnode3, vnode4}
- cbrick=cbrick3: {vnode5, vnode6}
- cbrick=cbrick4: {vnode7, vnode8}
- L2=A: {vnode1, vnode3, vnode5, vnode7}
- L2=B: {vnode2, vnode4, vnode6, vnode8}

### 4.8.32.7.4 Example of Placement Sets Using Colors

A placement pool is defined by two resources: colorset1 and colorset2, by using "node_group_key=colorset1,colorset2".

If a vnode has the following values set:

```
resources_available.colorset1=blue, red
resources_available.colorset2=green
```

The placement pool contains at least three placement sets. These are:

- \{resources_available.colorset1=blue\}
- \{resources_available.colorset1=red\}
- \{resources_available.colorset2=green\}

This means the vnode is in all three placement sets. The same result would be given by using one resource and setting it to all three values, e.g. colorset=blue, red, green.

**Example:** We have five vnodes v1 – v5:

```
v1 color=red host=mars
v2 color=red host=mars
v3 color=red host=venus
v4 color=blue host=mars
v5 color=blue host=mars
```

The placement sets are defined by

```
node_group_key=color
```
The resulting node groups would be: \{v1, v2, v3\}, \{v4, v5\}

4.8.32.7.5 Simple Switch Placement Set Example

Say you have a cluster with two high-performance switches each with half the vnodes connected to it. Now you want to set up placement sets so that jobs will be scheduled only onto the same switch.

First, create a new resource called “switch”. See section 5.14.2 “Defining New Custom Resources” on page 357.

Next, we need to enable placement sets and specify the resource to use:

```
Qmgr: set server node_group_enable=True
Qmgr: set server node_group_key=switch
```

Now, set the value for switch on each vnode:

```
Qmgr: active node vnode1,vnode2,vnode3
Qmgr: set node resources_available.switch=A
Qmgr: active node vnode4,vnode5,vnode6
Qmgr: set node resources_available.switch=B
```

Now there are two placement sets:

- switch=A: \{vnode1, vnode2, vnode3\}
- switch=B: \{vnode4, vnode5, vnode6\}

4.8.32.8 Placement Sets and Reservations

When PBS chooses a placement set for a reservation, it makes the same choices as it would for a regular job. It fits the reservation into the smallest possible placement set. See section 4.8.32.4.2 “Order of Placement Set Consideration Within Pool” on page 243.

When a reservation is created, it is created within a placement set, if possible. If no placement set will satisfy the reservation, placement sets are ignored. The vnodes allocated to a reservation are used as one single placement set for jobs in the reservation; they are not subdivided into smaller placement sets. A job within a reservation runs within the single placement set made up of the vnodes allocated to the reservation.
4.8.32.9 Placement Sets and Load Balancing

If you configure both placement sets and load balancing, the net effect is that vnodes that are over their load limit will be removed from consideration.

4.8.32.10 Viewing Placement Set Information

You can find information about placement sets in the following places:

- The server’s node_group_enable attribute shows whether placement sets are enabled
- The server’s node_group_key attribute contains the names of resources used for that queue’s placement pool
- Each queue’s node_group_key attribute contains the names of resources used for that queue’s placement pool
- Each vnode’s pnames attribute can contain the names of resources used for placement sets, if properly set
- PBS-generated MOM configuration files contain names and values of resources

4.8.32.11 Placement Set Caveats and Advice

- When you create a Version 2 configuration file for a pre-existing vnode, make sure it specifies all of the information about the vnode, such as resources and attribute settings. The creation of the configuration file overrides previous settings, and if the new file contains no specification for a resource or attribute, that resource or attribute becomes unset.
- If there is a vnode-level platform-specific resource set on the vnodes on the Altix, then node_group_key should probably include this resource, because this will enable PBS to run jobs in more logical sets of vnodes.
- If the user specifies a job-specific placement set, for example -lplace=group=switch, but the job cannot statically fit into any switch placement set, then the job will still run, but not in a switch placement set.
- The pnames vnode attribute is for displaying to the administrator the
resources used for placement sets. This attribute is not used by PBS.

4.8.32.11.1 Non-backward-compatible Change in Node Grouping

Given the following example configuration:

- vnode1: switch=A
- vnode2: switch=A
- vnode3: switch=B
- vnode4: switch=B
- vnode5: switch unset

Qmgr: s s node_group_key=switch

There is no change in the behavior of jobs submitted with qsub -l ncpus=1

- version 7.1: The job can run on any node: node1, ..., node5
- version 8.0: The job can run on any node: node1, ..., node5

Example of 8.0 and later behavior: jobs submitted with qsub -lnodes=1

- version 7.1: The job can only run on nodes: node1, node2, node3, node4. It will never use node5
- version 8.0: The job can run on any node: node1, ..., node5

Overall, the change for version 8.0 was to include every vnode in placement sets (when enabled). In particular, if a resource is used in node_group_key, PBS will treat every vnode as having a value for that resource, hence every vnode will appear in at least one placement set for every resource. For vnodes where a string resource is "unset", PBS will behave as if the value is “".

4.8.32.12 Attributes and Parameters Affecting Placement Sets

do_not_span_psets

Scheduler attribute. Specifies whether or not the scheduler requires the job to fit within one of the existing placement
sets. When `do_not_span_psets` is set to `True`, the scheduler will require the job to fit within a single existing placement set. The scheduler checks all placement sets, whether or not they are currently in use. If the job fits in a currently-used placement set, the job must wait for the placement set to be available. If the job cannot fit within a single placement set, it will not run.

When this attribute is set to `False`, the scheduler will first attempt to place the job in a single placement set, but if it cannot, it will allow the job to span placement sets, running on whichever vnodes can satisfy the job’s resource request.

Format: `Boolean`

Default value: `False` (This matches behavior of PBS 10.4 and earlier)

Example: To require jobs to fit within one placement set:

```
Qmgr: set sched do_not_span_psets=True
```

### node_group_enable

Server attribute. Specifies whether placement sets are enabled.

Format: `Boolean`

Default: `False`

### node_group_key

Server and queues have this attribute. Specifies resources to use for placement set definition. Queue’s attribute overrides server’s attribute.

Format: `string_array`

Default: Unset
4.8.32.13 Errors and Logging

If `do_not_span_psets` is set to `True`, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:
  "Not Running: can't fit in the largest placement set, and can't span psets"
- The following message is printed to the scheduler’s log:
  "Can't fit in the largest placement set, and can't span placement sets"

4.8.33 Using Preemption

PBS provides the ability to preempt currently running jobs in order to run higher-priority work. This is called preemption or preemptive scheduling. Preemption follows a set of preemption priorities that you define. Jobs that have high preemption priority preempt those with low preemption priority. Preemption priority is mostly independent of execution priority.

Preemption is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.8.33.1 Terminology

**Preempt**

Stop one or more running jobs in order to start a higher-priority job

**Preemption level**

Job characteristic that determines preemption priority. Levels can be things like being in an express queue, starving, having an owner who is over a soft limit, being a normal job, or having an owner who is over a fairshare allotment

**Preemption method**

The method by which a job is preempted. This can be checkpointing, suspension, or requeueing
Preemption priority
How important this job is compared to other jobs, when considering whether to preempt

4.8.33.2 Preemption Parameters

The scheduler parameters that control preemption are defined in PBS_HOME/sched_priv/sched_config. They are listed here:

preemptive_sched
Enables job preemption.
Format: String
Default: True all

preempt_order
Defines the order of preemption methods which the scheduler will use on jobs.
Format: String, as quoted list
Default: “SCR”

preempt_prio
Specifies the ordering of priority of different preemption levels.
Format: String, as quoted list
Default: “express_queue, normal_jobs”

preempt_queue_prio
Specifies the minimum queue priority required for a queue to be classified as an express queue.
Format: Integer
Default: 150

preempt_sort
Whether jobs most eligible for preemption will be sorted according to their start times. Allowable values: “min_time_since_start”, or no preempt_sort setting. If set to “min_time_since_start”, first job preempted will be that with most recent start time. If not set, preempted job will be that with longest running time. Must be commented
out in order to be unset; default scheduler configuration file has this parameter set to \textit{min\_time\_since\_start}.

Format: \textit{String}

Default: \textit{min\_time\_since\_start}

### 4.8.33.3 How Preemption Works

If preemption is enabled, the scheduler uses preemption as part of its normal pattern of examining each job and figuring out whether or not it can run now. If a job with high preemption priority cannot run immediately, the scheduler looks for jobs with lower preemption priority. The scheduler preempts these jobs and uses their resources for the higher-priority job. The scheduler tracks resources used by lower-priority jobs, looking for enough resources to run the higher-priority job. If the scheduler cannot find enough work to preempt in order to run a given job, it will not preempt any work.

A job running in a reservation cannot be preempted.

A job’s preemption priority is determined by its preemption level.

### 4.8.33.4 Preemption Levels

A preemption level is a class of jobs, where all the jobs in the class share a characteristic. PBS provides built-in preemption levels, listed in the table below.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Preemption Level} & \textbf{Description} \\
\hline
deadline & Jobs in express queues. See section 4.8.33.4.2 “The Express Queues Preemption Level” on page 264 \\
\hline
starving_jobs & A job that is starving. See section 4.8.33.4.4 “The Starving Job Preemption Level” on page 265 \\
\hline
\end{tabular}
\end{table}
You can specify the relative priority of each preemption level, by listing the levels in the desired order in the `preempt_prio` scheduler parameter. Placing a level earlier in the list, meaning to the left, gives it higher priority. For example, if your list is "express_queue", "normal_jobs", "server_softlimits", you are giving the highest priority to jobs in express queues, and the lowest priority to jobs that are over their server soft limits. You can list levels in any order, but be careful not to work at cross-purposes with your execution priority. See section 4.8.16 “Calculating Job Execution Priority” on page 187.

The default value for `preempt_prio` is the following:

```
preempt_prio: "express_queue, normal_jobs"
```

If you do not list a preemption level in the `preempt_prio` scheduler parameter, the jobs in that level are treated like normal jobs. For example, if you do not list `server_softlimits`, then jobs that are over their server soft limits are treated like jobs in the `normal_jobs` level.

You can create new levels, which are combinations of the built-in levels. For example, you can define a level which is `express_queue + server_softlimits`. This level contains jobs that are in express queues and are over their server soft limits. You would probably want to place this

---

### Table 4-13: Built-in Preemption Levels

<table>
<thead>
<tr>
<th>Preemption Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fairshare</td>
<td>When the entity owning a job exceeds its fair-share limit. See section 4.8.33.4.3 “The Fairshare Preemption Level” on page 265</td>
</tr>
<tr>
<td>queue_softlimits</td>
<td>Jobs which are over their queue soft limits. See section 4.8.33.1 “The Soft Limits Preemption Level” on page 262</td>
</tr>
<tr>
<td>server_softlimits</td>
<td>Jobs which are over their server soft limits. See section 4.8.33.1 “The Soft Limits Preemption Level” on page 262</td>
</tr>
<tr>
<td>normal_jobs</td>
<td>The preemption level into which a job falls if it does not fit into any other specified level. See section 4.8.33.4.5 “The Normal Jobs Preemption Level” on page 265</td>
</tr>
</tbody>
</table>

---
level just to the right of the `express_queue` level, meaning that these jobs could be preempted by jobs that are in express queues but are not over their server soft limits.

You can give two or more levels the same priority. To do this, put a plus sign ("+") between them, and do not list either level separately in `preempt_prio`. You are creating a new level that includes all the built-in levels that should have the same priority. For example, to list express queue jobs as highest in priority, then fairshare and starving jobs at the next highest priority, then normal jobs last, create a new level that contains the `fairshare` and `starving_jobs` levels:

```plaintext
preempt_prio: "express_queue, fairshare+starving_jobs, normal_jobs"
```

You must list `normal_jobs` in the `preempt_prio` scheduler parameter.

### 4.8.33.4.1 The Soft Limits Preemption Level

You can set a limit, called a `hard limit`, on the number of jobs that can be run or the amount of a resource that can be consumed by a person, a group, or by everyone, and this limit can be applied at the server and at each queue. If you set such a limit, that is the greatest number of jobs that will be run, or the largest amount of the resource that will be consumed.

You can also set a `soft limit` on the number of jobs that can be run or the amount of a resource that can be consumed. This soft limit should be lower than the hard limit, and should mark the point where usage changes from being normal to being “extra, but acceptable”. Usage in this “extra, but acceptable” range can be treated by PBS as being lower priority than the normal usage. PBS can preempt jobs that are over their soft limits. The difference between the soft limit and the hard limit provides a way for users or groups to use resources as long as no higher-priority work is waiting.

Example 1: Using group soft limits

One group of users, group A, has submitted enough jobs that the group is over their soft limit. A second group, group B, submits a job and are under their soft limit. If preemption is enabled, jobs from group A are preempted until the job from group B can run.
Example 2: Using soft limits on number of running jobs

Given the following:

- You have three users, UserA, UserB, and UserC
- Each has a soft limit of 3 running jobs
- UserA runs 3 jobs
- UserB runs 4 jobs
- UserC submits a job to an express queue

This means:

- UserB has 1 job over the soft limit, so UserB’s jobs are eligible for preemption by UserC’s job

Example 3: Using soft limits on amount of resource being used

Given the following:

- Queue soft limit for ncpus is 8
- UserA’s jobs use 6 CPUs
- UserB’s jobs use 10 CPUs

This means:

- UserB is over their soft limit for CPU usage
- UserB’s jobs are eligible for preemption

To use soft limits in preemption levels, you must define soft limits. Soft limits are specified by setting server and queue limit attributes. The attributes that control soft limits are:

```
max_run_soft
Sets the soft limit on the number of jobs that can be running

max_run_res_soft.<resource>
Sets the soft limit on the amount of a resource that can be consumed by running jobs
```

Soft limits are enforced only when they are used as a preemption level.

To use soft limits as preemption levels, add their keywords to the preempt_prio parameter in the scheduler’s configuration file:

- To create a preemption level for those over their soft limits at the server
level, add “server_softlimits” to the preempt_prio parameter.

- To create a preemption level for those over their soft limits at the queue level, add “queue_softlimits” to the preempt_prio parameter.

- To create a preemption level for those over their soft limits at both the queue and server, add “server_softlimits+queue_softlimits” to the preempt_prio parameter.

The jobs of a user or group are over their soft limit only as long as the number of running jobs or the amount of resources used by running jobs is over the soft limit. If some of these jobs are preempted or finish running, and the soft limit is no longer exceeded, the jobs of that user or group are no longer over their soft limit, and no longer in that preemption level. For example, if the soft limit is 3 running jobs, and UserA runs 4 jobs, as soon as one job is preempted and only 3 of UserA’s jobs are running, UserA’s jobs are no longer over their soft limit.

For a complete description of the use of these attributes, see section 5.15.1.3 “Hard and Soft Limits” on page 414.

4.8.33.4.2 The Express Queues Preemption Level

The express_queue preemption level applies to jobs residing in express queues. An express queue is an execution queue with priority at or above the value set in the preempt_queue_prio scheduler parameter. The default value for this parameter is 150.

Express queues do not require the by_queue or sort_queues scheduler parameters to be True.

If you will use the express_queue preemption level, you probably want to configure at least one express queue, along with some method of moving jobs into it. See section 2.2 “Queues” on page 19.

If you have more than one express queue, and they have different priorities, you are effectively creating separate sub-levels for express queues. Jobs in a higher-priority express queue have greater preemption priority than jobs in lower-priority express queues.

See “preempt_queue_prio” on page 339 of the PBS Professional Reference Guide.
4.8.33.4.3 The Fairshare Preemption Level

The fairshare preemption level applies to jobs owned by entities who are over their fairshare allotment. For example, if each of five users has 20 percent of the fairshare tree, and UserA is using 25 percent of the resources being tracked for fairshare, UserA’s jobs become eligible for preemption at the fairshare preemption level.

To use the fairshare preemption level, you must enable fairshare. See section 4.8.18 “Using Fairshare” on page 191.

4.8.33.4.4 The Starving Job Preemption Level

The starving_jobs preemption level applies to jobs that are starving. Starving jobs are jobs that have been waiting at least a specified amount of time to run.

To use the starving_jobs preemption level, you must enable starving:

- Set the PBS_HOME/sched_priv/sched_config help_starving_jobs parameter to True
- Set the amount of time that a job must wait before it is starving in the max_starve scheduler parameter


4.8.33.4.5 The Normal Jobs Preemption Level

One special class, normal_jobs, is the default class for any job not otherwise specified. If a job does not fall into any of the specified levels, it is placed in normal_jobs.

Example 1: Starving jobs have the highest priority, then normal jobs, then jobs whose entities are over their fairshare limit:

preempt_prio: “starving_jobs, normal_jobs, fairshare”

Example 2: Starving jobs whose entities are also over their fairshare limit are lower priority than normal jobs:

preempt_prio: “normal_jobs, starving_jobs+fairshare”
4.8.33.5 Selecting Preemption Level

PBS places each job in the most exact preemption level, or the highest preemption level that fits the job.

Example 1: We have a job that is starving and over its server soft limits. The job is placed in the “starving_jobs” level:

\[\text{preempt_prio: \text{“starving_jobs, normal_jobs, server_softlimits”}}\]

Example 2: We have a job that is starving and over its server soft limits. The job is placed in the “starving_jobs+server_softlimits” level:

\[\text{preempt_prio: \text{“starving_jobs, starving_jobs+server_softlimits, normal_jobs, server_softlimits”}}\]

4.8.33.6 Sorting Within Preemption Level

If there is more than one job within the preemption level chosen for preemption, PBS chooses jobs within that level according to their start time. By default, PBS preempts the job which started running most recently. This behavior can be changed using the scheduler parameter \text{preempt_sort}. To direct PBS to preempt the longest-running jobs, comment out the line containing the \text{preempt_sort} parameter in \text{PBS_HOME/sched_priv/sched_config}.

For example, if we have two jobs where job A started running at 10:00 a.m. and job B started running at 10:30 a.m:

- The default behavior preempts job B
- Job A is preempted when \text{preempt_sort} is commented out

The allowable value for the \text{preempt_sort} parameter is “\text{min_time_since_start}”.

The default value for the \text{preempt_sort} parameter is “\text{min_time_since_start}”. Must be commented out in order to be unset; default scheduler configuration file has this parameter set to \text{min_time_since_start}. 
4.8.33.7 Preemption Methods

The scheduler can preempt a job in one of the following ways:

- Suspend the job
- Checkpoint the job
- Requeue the job

The scheduler tries to preempt a job using the methods listed in the order you specify. This means that if you specify that the order is “checkpoint, suspend, requeue”, the scheduler first tries to checkpoint the job, and if it cannot, it tries to suspend the job, and if it cannot do that, it tries to requeue the job.

You can specify the order of these attempts in the `preempt_order` scheduler parameter in `PBS_HOME/sched_priv/sched_config`.

The `preempt_order` parameter defines the order of preemption methods which the scheduler uses on jobs. This order can change depending on the percentage of time remaining on the job. The ordering can be any combination of `S`, `C` and `R` (for suspend, checkpoint, and requeue).

The contents is an ordering, for example “SCR” optionally followed by a percentage of time remaining and another ordering.

The format is a quoted list (“”).

Example 1: PBS should first attempt to use suspension to preempt a job, and if that is unsuccessful, then requeue the job:
```plaintext
preempt_order: “SR”
```

Example 2: If the job has between 100-81% of requested time remaining, first try to suspend the job, then try checkpoint, then requeue. If the job has between 80-51% of requested time remaining, then attempt suspend then checkpoint; and between 50% and 0% time remaining just attempt to suspend the job:
```plaintext
preempt_order: “SCR 80 SC 50 S”
```

The default value for `preempt_order` is “`SCR`”.
4.8.33.7.1 Preemption Via Checkpoint

When a job is preempted via checkpointing, MOM runs the checkpoint_abort script, and PBS kills and requeues the job. When the scheduler elects to run the job again, the scheduler runs the restart script to restart the job from where it was checkpointed.

To preempt via checkpointing, you must define both of the following:

- The checkpointing action in the MOM’s checkpoint_abort $action parameter that is to take place when the job is preempted
- The restarting action in the MOM’s restart $action parameter that is to take place when the job is restarted

To do this, you must supply checkpointing and restarting scripts or equivalents, and then configure the MOM’s checkpoint_abort and restart $action parameters. Do not use the $action checkpoint MOM parameter; it is used when the job should keep running.

See section 10.3 “Checkpoint and Restart” on page 735.

4.8.33.7.2 Preemption Via Suspension

Jobs are normally suspended via the SIGSTOP signal and resumed via the SIGCONT signal. An alternate suspend or resume signal can be configured in MOM’s $suspendsig configuration parameter. See “pbs_mom” on page 66 of the PBS Professional Reference Guide.

4.8.33.7.3 Suspended Jobs and PBS Licenses

When a job is suspended, its PBS usage licenses are returned to the license pool, subject to the constraints of the server’s pbs_license_min and pbs_license_linger_time attributes. The scheduler checks to make sure that licenses are available before resuming any job. If the required licenses are not available, the scheduler will log a message and add a comment to the job. See section 4.7.2 “Licensing and Job States” on page 120 in the PBS Professional Installation & Upgrade Guide.
4.8.33.7.4 Suspended Jobs and Resources

Suspended jobs will hold onto some memory and disk space. Suspended jobs may hold application licenses if the application releases them only when it exits. See section 5.9.5.2.1 “Suspension/resumption Resource Caveats” on page 347.

4.8.33.7.5 Preemption Via Requeue

When a job is preempted and requeued, the job stops execution and is requeued. A requeued job’s eligible time is preserved. The amount of time allowed to requeue a job is controlled by the job_requeue_timeout server attribute. See “Server Attributes” on page 365 of the PBS Professional Reference Guide.

A job that is not eligible to be requeued, meaning a job that was submitted with “-r n”, will not be selected to be preempted via requeue.

4.8.33.8 Enabling Preemption

Preemptive scheduling is enabled by setting parameters in the scheduler’s configuration file PBS_HOME/sched_priv/sched_config.
To enable preemption, you must do the following:

1. Specify the preemption levels to be used by setting `preempt_prio` to desired preemption levels (the default is “express_queue, normal_jobs”)

   The `preempt_prio` parameter must contain an entry for `normal_jobs`.

2. Optional: specify preemption order by setting `preempt_order`

3. Optional: specify whether longest- or shortest-running jobs should be preempted first by setting `preempt_sort`

4. If you will use the `fairshare` preemption level, configure fairshare. See section 4.8.18 “Using Fairshare” on page 191.

5. If you will use the `starving_jobs` preemption level, configure starving. See section 4.8.33.4.4 “The Starving Job Preemption Level” on page 265.

6. If you will use the `server_softlimits` and/or `queue_softlimits` preemption levels, configure server and/or queue soft limits. See section 4.8.33.4.1 “The Soft Limits Preemption Level” on page 262.

7. Enable preemption by setting `preemptive_sched` to `True`. It is `True` by default.

8. Choose whether to use preemption during primetime, non-primetime, or all of the time. The default is ALL. If you want separate behavior for primetime and non-primetime, specify each separately. For example:

   ```
   preemptive_sched True prime
   preemptive_sched False non_prime
   ```

9. If you are sorting jobs on a key using `job_sort_key`, make `preempt_priority` the primary sort key:

   ```
   job_sort_key: “preempt_priority HIGH”
   ```
4.8.33.9 Preemption Example

Below is an example of (part of) the scheduler’s configuration file, showing an example configuration for preemptive scheduling.

```plaintext
# turn on preemptive scheduling
#
preemptive_sched: TRUE ALL
#
# set the queue priority level for express queues
#
preempt_queue_prio: 150
#
# specify the priority of jobs as: express queue
# (highest) then starving jobs, then normal jobs,
# followed by jobs who are starving but the user/group
# is over a soft limit, followed by users/groups over
# their soft limit but not starving
#
preempt_prio: "express_queue, starving_jobs,
  normal_jobs, starving_jobs+server_softlimits,
  server_softlimits"
#
# specify when to use each preemption method.
# If the first method fails, try the next
# method. If a job has between 100-81% time
# remaining, try to suspend, then checkpoint
# then requeue. From 80-51% suspend and then
# checkpoint, but don’t requeue.
# If between 50-0% time remaining, then just
# suspend it.
#
preempt_order: "SCR 80 SC 50 S"
```
4.8.33.10 Preemption Caveats and Recommendations

- When using any of the fairshare, soft limits, express queue, or starving jobs preemption levels, be sure to enable the corresponding PBS feature. For example, when using preemption with the fairshare preemption level, be sure to turn fairshare on. Otherwise, you will be using stale fairshare data to preempt jobs.

- It’s important to be careful about the order of the preemption levels and the sizes of the limits at queue and server. For example, if you make users who are over their server soft limits have higher priority than users who are over their queue soft limits, and you set the soft limit higher at the server than at the queue, you can end up with users who have more jobs running preemption users who have fewer jobs running.

In this example, a user with more jobs preempts a user with fewer jobs.

Given the following:
- preempt_prio line contains “server_softlimits, queue_softlimits”
- Server soft limit is 5
- Queue soft limit is 3
- User1 has 6 jobs running
- User2 has 4 jobs running

This means:
- User1 has higher priority, because User1 is over the server soft limit
- User1’s jobs can preempt User2’s jobs

To avoid this scenario, you could set the preempt_prio line to contain “server_softlimits, queue_softlimits, server_softlimits+queue_softlimits”. In this case, User1 would have lower priority, because User1 is over both soft limits.

- Preemption priority is independent of execution priority. You can list preemption levels in any order in preempt_prio, but be careful not to work at cross-purposes with your execution priority. Be sure that you are not preempting jobs that have higher execution priority. See section
4.8.16 “Calculating Job Execution Priority” on page 187.

- If you are sorting jobs using job_sort_key, we recommend that when you change preempt_prio from the default, you add preempt_priority as the primary key for the job_sort_key scheduler parameter. This is especially important when using soft limits for preemption. Jobs that are over their soft limits should have lower priority, both in execution and preemption. For example, if you’re using soft limit preemption levels, but not assigning job execution priority according to soft limits, you can have job A started and preempted in the same scheduler cycle, because job A has higher execution priority, but is over its soft limits, and has lower preemption priority. To sort jobs on preemption priority:

  job_sort_key: "preempt_priority HIGH"

  See section 4.8.42 “Sorting Jobs on a Key” on page 302.

- Using preemption with strict ordering and backfilling may change which job is being backfilled around.

- When a job is suspended via checkpoint or requeue, it loses its queue wait time. This does not apply to preemption via suspension.

- If a high-priority job has been selected to preempt lower-priority jobs, but is rejected by a runjob hook, the scheduler undoes the preemption of the low-priority jobs. Suspended jobs are resumed, and checkpointed jobs are restarted.

- A job that has requested an AOE will not preempt another job, regardless of whether the job’s requested AOE matches an instantiated AOE. Running jobs are not preempted by jobs requesting AOE.

- If a job is checkpointed by the scheduler because it was preempted, the scheduler briefly applies a hold, but releases the hold immediately after checkpointing the job, and runs the restart script when the job is scheduled to run.

- When jobs are preempted via requeueing, the requeue can fail if the job being preempted takes longer than the allowed timeout. See section 10.4.3 “Setting Job Requeue Timeout” on page 767.

- A job that is suspended has higher execution priority than a queued job. Therefore, a job that is preempted via suspension has higher priority than a job preempted via checkpoint or requeue, because those jobs are
in the queued state.

- When you issue “qrun <job ID>”, without the -H option, the selected job has preemption priority between Reservation and Express.

## 4.8.34 Using Primetime and Holidays

Often is it useful to run different scheduling policies for specific intervals during the day or work week. PBS provides a way to specify two types of interval, called *primetime* and *non-primetime*.

Between them, primetime and non-primetime cover all time. There is no time slot that is neither primetime nor non-primetime. This includes dedicated time. Primetime and/or non-primetime overlap dedicated time.

You can use non-primetime for such tasks as running jobs on desktop clusters at night.

### 4.8.34.1 How Primetime and Holidays Work

The scheduler looks in the PBS_HOME/sched_priv/holidays file for definitions of primetime, non-primetime, and holidays.

Many PBS scheduling parameters can be specified separately for primetime, non-primetime, or all of the time. This means that you can use, for example, fairshare during primetime and no fairshare during non-primetime. These parameters have a time slot default of all, meaning that if enabled, they are in force all of the time.

The scheduler applies the parameters defined for primetime during the primetime time slots, and applies parameters defined for non-primetime during the non-primetime time slots. Any scheduler parameters defined for all time are run whether it is primetime or not.

Any holidays listed in the holidays file are treated as non-primetime. To have a holiday treated like a normal workday or weekend, do not list it in the holidays file.

There are default behaviors for primetime and non-primetime, but you can set up the behavior you want for each type. The names “primetime” and “non-primetime” are meant to be informative, but they are arbitrary. The
default for primetime is from 6:00 AM to 5:30 PM on weekdays, meaning that weekends and nights are non-primetime by default. U.S. Federal holidays are provided in the holidays file.

You can define primetime and non-primetime queues. Jobs in these queues can run only during the designated time. Queues that are not defined specifically as primetime or non-primetime queues are called “anytime queues”.

### 4.8.34.2 Configuring Primetime and Non-primetime

In order to use primetime and non-primetime, you must have a *holidays* file with the current year in it.

You can specify primetime and non-primetime time slots by specifying them in the `PBS_HOME/sched_priv/holidays` file.

The format of the primetime and non-primetime section of the *holidays* file is the following:

```
YEAR YYYY
<day> <prime> <nonprime>
<day> <prime> <nonprime>
```

If there is no `YEAR` line in the holidays file, primetime is in force at all times. If there is more than one `YEAR` line, the last one is used.

In `YEAR YYYY`, `YYYY` is the current year.

*Day* can be *weekday, monday, tuesday, wednesday, thursday, friday, saturday,* or *sunday.*

Each line must have all three fields.

Any line that begins with a “*” or a “#” is a comment.

Weekday names must be lowercase.

The ordering of elements in this file is important. The ordering of `<day>` lines in the holidays file controls how primetime is determined. A later line takes precedence over an earlier line.
For example:

```
weekday  0630  1730
friday   0715  1600
```

means the same as

```
monday   0630  1730
tuesday  0630  1730
wednesday 0630  1730
thursday 0630  1730
friday   0715  1600
```

However, if a specific day is followed by “weekday”,

```
friday   0700  1600
weekday  0630  1730
```

the “weekday” line takes precedence, so Friday will have the same prime-time as the other weekdays.

Times can be expressed as one of the following:

- \texttt{HHMM} with no colons(:)
- The word “all”
- The word “none”

### 4.8.34.3 Configuring Holidays

You can specify primetime and non-primetime time slots by specifying them in the \texttt{PBS\_HOME/sched\_priv/holidays} file.

You must specify the year, otherwise primetime is in force at all times, and PBS will not recognize any holidays. Specify the year here, where YYYY is the current year:

```
YEAR YYYY
```

Holidays are specified in lines of this form:

```
<day of year> <month day-of-month> <holiday name>
```

PBS uses the \texttt{<day of year>} field and ignores the \texttt{<date>} string.
Day of year is the julian day of the year between 1 and 365 (e.g. “1”).

Month day-of-month is the calendar date, for example “Jan 1”.

Holiday name is the name of the holiday, for example “New Year’s Day”.

### 4.8.34.4 Example of holidays File

<table>
<thead>
<tr>
<th>YEAR</th>
<th>2007</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Prime Non-Prime</td>
</tr>
<tr>
<td>* Day</td>
<td>Start Start</td>
</tr>
<tr>
<td>*</td>
<td></td>
</tr>
<tr>
<td>weekday</td>
<td>0600 1730</td>
</tr>
<tr>
<td>saturday</td>
<td>none all</td>
</tr>
<tr>
<td>sunday</td>
<td>none all</td>
</tr>
</tbody>
</table>

* Day of Calendar Company Holiday

<table>
<thead>
<tr>
<th>Year</th>
<th>Date</th>
<th>Holiday</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan 1</td>
<td>New Year's Day</td>
</tr>
<tr>
<td>15</td>
<td>Jan 15</td>
<td>Dr. M.L. King Day</td>
</tr>
<tr>
<td>50</td>
<td>Feb 19</td>
<td>President's Day</td>
</tr>
<tr>
<td>148</td>
<td>May 28</td>
<td>Memorial Day</td>
</tr>
<tr>
<td>185</td>
<td>Jul 4</td>
<td>Independence Day</td>
</tr>
<tr>
<td>246</td>
<td>Sep 3</td>
<td>Labor Day</td>
</tr>
<tr>
<td>281</td>
<td>Oct 8</td>
<td>Columbus Day</td>
</tr>
<tr>
<td>316</td>
<td>Nov 12</td>
<td>Veteran's Day</td>
</tr>
<tr>
<td>326</td>
<td>Nov 22</td>
<td>Thanksgiving</td>
</tr>
<tr>
<td>359</td>
<td>Dec 25</td>
<td>Christmas Day</td>
</tr>
</tbody>
</table>

### 4.8.34.5 Reference Copies of holidays File

Reference copies of the holidays file are provided in PBS_EXEC/etc/holiday.<year>.
The current year’s holidays file has a reference copy in PBS_EXEC/etc/pbs_holidays, and a copy used by PBS in PBS_HOME/sched_priv/holidays.

To use a particular year’s file as the holidays file, copy it to PBS_HOME/sched_priv/holidays -- note the “s” on the end of the filename.

4.8.34.6 Defining Primetime and Non-primetime Queues

Jobs in a primetime queue can start only during primetime. Jobs in a non-primetime queue can start only during non-primetime. Jobs in an anytime queue can start at any time.

You define a primetime queue by naming it using the primetime prefix. The prefix is defined in the primetime_prefix scheduler parameter. The default is “p_”. For example, you could name a primetime queue “p_queueA”, using the default.

Similarly, you define a non-primetime queue by prefixing the name. The prefix is defined in the nonprimetime_prefix scheduler parameter, and defaults to “np_”.

4.8.34.7 Controlling Whether Jobs Cross Primetime Boundaries

You can control whether jobs are allowed to start running in one time slot and finish in another, for example when job A starts during primetime and finishes a few minutes into non-primetime. When a job runs past the boundary, it delays the start of a job that is constrained to run only in the later time slot. For example, if job B can run only during non-primetime, it may have to wait while job A uses up non-primetime before it can start.

You can control this behavior for all queues, or you can exempt anytime queues, controlling only primetime and non-primetime queues. You can also specify how much time past the boundary a job is allowed to run.
To prevent the scheduler from starting any jobs which would run past a primetime/non-primetime boundary, set the `backfill_prime` scheduler parameter to `True`. You can specify this separately for primetime and non-primetime. If you specify it for one type of time slot, it prevents those jobs from crossing the next boundary. For example, if you set the following:

```
backfill_prime True prime
```

jobs in primetime slots are not allowed to cross into non-primetime slots.

If you set the following:

```
backfill_prime True non_prime
```

jobs in non-primetime slots are not allowed to cross into primetime slots.

To exempt jobs in anytime queues from the control of `backfill_prime`, set the `prime_exempt_anytime_queues` scheduler parameter to `True`. This means that jobs in an anytime queue are not prevented from running across a primetime/nonprimetime or non-primetime/primetime boundary.

To allow jobs to spill over a certain amount of time past primetime/non-primetime boundaries, but no more, specify this amount of time in the `prime_spill` scheduler parameter. You can specify separate behavior for primetime and non-primetime jobs. For example, to allow primetime jobs to spill by 20 minutes, but only allow non-primetime jobs to spill by 1 minute:

```
prime_spill 00:20:00 prime
prime_spill 00:01:00 non_prime
```

The `prime_spill` scheduler parameter applies only when `backfill_prime` is `True`.

### 4.8.34.8 Logging

The scheduler logs a message at the beginning of each scheduling cycle indicating whether it is primetime or not, and when this period of primetime or non-primetime will end. The message is at log event class 0x0100. The message is of this form:

```
“It is primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS”
```
or

“It is non-primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS”

### 4.8.34.9 Scheduling Parameters Affecting Primetime

**backfill_prime**

The scheduler will not run jobs which would overlap the boundary between primetime and non-primetime.

Format: *Boolean*

Default: *False all*

**nonprimetime_prefix**

Queue names which start with this prefix will be treated as non-primetime queues. Jobs within these queues will only run during non-primetime.

Format: *String*

Default: *np_*

**primetime_prefix**

Queue names starting with this prefix are treated as prime-time queues. Jobs will only run in these queues during primetime.

Format: *String*

Default: *p_*

**prime_exempt_anytime_queues**

Determines whether anytime queues are controlled by backfill_prime.

If set to *True*, jobs in an anytime queue will not be prevented from running across a primetime/non-primetime or non-primetime/primetime boundary.

If set to *False*, the jobs in an anytime queue may not cross this boundary, except for the amount specified by their prime_spill setting.

Format: *Boolean*

Default: *False*
prime_spill
Specifies the amount of time a job can spill over from non-primetime into primetime or from primetime into non-primetime. This option can be separately specified for prime- and non-primetime. This option is only meaningful if backfill_prime is True.

Format: Duration
Default: 00:00:00

4.8.34.10 Primetime and Holiday Caveats

• In order to use primetime and non-primetime, you must have a holidays file with the current year in it. If there is no holidays file with a year in it, primetime is in force all of the time.

• You cannot combine holidays files.

• If you use the formula, it is in force all of the time.

4.8.35 Provisioning

PBS provides automatic provisioning of an OS or application, on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

You can configure vnodes so that PBS will automatically install the OS or application that jobs need in order to run on those vnodes. For example, you can configure a vnode that is usually running RHEL to run SLES instead whenever the Physics group runs a job requiring SLES. If a job requires an application that is not usually installed, PBS can install the application in order for the job to run.

You can use provisioning for booting multi-boot systems into the desired OS, downloading an OS to and rebooting a diskless system, downloading an OS to and rebooting from disk, instantiating a virtual machine, etc. You can also use provisioning to run a configuration script or install an application.

For a complete description of how provisioning works and how to configure it, see “Provisioning” on page 599.
4.8.36 Queue Priority

Queues and queue priority play several different roles in scheduling, so this section contains pointers to other sections.

Each queue can have a different priority. A higher value for priority means the queue has greater priority. By default, queues are sorted from highest to lowest priority. Jobs in the highest priority queue will be considered for execution before jobs from the next highest priority queue. If queues don’t have different priority, queue order is undefined. Each queue’s priority is specified in its priority attribute. By default, the queue priority attribute is unset. See “Queue Attributes” on page 407 of the PBS Professional Reference Guide.

4.8.36.1 Configuring Queue Priority

You can specify the priority of each queue by setting a value for its priority attribute:

```
Qmgr: set queue <queue name> priority = <value>
```

4.8.36.2 Using Queue Priority

You can configure the scheduler so that job execution or preemption priority is partly or entirely determined by the priority of the queue in which the job resides. Queue priority can be used for the following purposes:

- Queue priority can be used as a term in the job sorting formula. See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205
- Queue priority can be used to specify order in which queues are examined when scheduling jobs. If you want jobs to be examined queue by queue, in order of queue priority, you must also sort the queues and specify a different priority for each queue. A queue with a higher value is examined before a queue with a lower value. See section 4.2.5.3.1 “Using Queue Order to Affect Order of Consideration” on page 77
- You can set up execution priority levels that include jobs in express queues. For information on configuring job priorities in the scheduler,
see section 4.8 “Setting Job Execution Priority” on page 112.

- You can set up preemption levels that include jobs in express queues. For information on preemption, see section 4.12.32 “Using Preemption” on page 202.

A queue is an express queue if its priority is greater than or equal to the value that defines an express queue. For more about using express queues, see section 4.8.17 “Express Queues” on page 190.

4.8.36.3 Queue Priority Caveats

- If you use queue priority in the formula and the job is moved to another server through peer scheduling, the queue priority used in the formula will be that of the new queue to which the job is moved.

4.8.37 Advance and Standing Reservations

PBS provides a way to reserve specific resources for a defined time period. You can make a one-time reservation, or you can make a series of reservations, where each one is for the same resources, but for a different time period.

Reservations are useful for accomplishing the following:

- To get a time slot on a specific host
- To run a job in a specific time slot, meaning at or by a specific time
- To be sure a job will run
- To have a high-priority job run soon

4.8.37.1 Definitions

**Advance reservation**

A reservation for a set of resources for a specified time. The reservation is available only to the creator of the reservation and any users or groups specified by the creator.

**Standing reservation**

An advance reservation which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every
Wednesday and Thursday from 5pm to 8pm, for the next three months.

**Occurrence of a standing reservation**

An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

- While a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occurrence. You can only specify *when* the job is eligible to run. See “qsub” on page 245 of the PBS Professional Reference Guide.

- When an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each occurrence may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the `resv_nodes` attribute reported by `pbs_rstat`.

Also called an *instance* of a standing reservation.

**Soonest occurrence of a standing reservation**

The occurrence which is currently active, or if none is active, then it is the next occurrence.

**Degraded reservation**

An advance reservation for which one or more associated vnodes are unavailable.

A standing reservation for which one or more vnodes associated with the soonest occurrence are unavailable.
4.8.37.2  How Reservations Work

4.8.37.2.1  Creating Reservations

Any PBS user can create both advance and standing reservations using the `pbs_rsub` command. PBS either confirms that the reservation can be made, or rejects the request. Once the reservation is confirmed, PBS creates a queue for the reservation’s jobs. Jobs are then submitted to this queue.

When a reservation is confirmed, it means that the reservation will not conflict with currently running jobs, other confirmed reservations, or dedicated time, and that the requested resources are available for the reservation. A reservation request that fails these tests is rejected. All occurrences of a standing reservation must be acceptable in order for the standing reservation to be confirmed.

The `pbs_rsub` command returns a reservation ID, which is the reservation name. For an advance reservation, this reservation ID has the format:

```
R<unique integer>.<server name>
```

For a standing reservation, this reservation ID refers to the entire series, and has the format:

```
S<unique integer>.<server name>
```

The user specifies the resources for a reservation using the same syntax as for a job.

See section 7.8, "Advance and Standing Reservation of Resources", on page 181 of the PBS Professional User’s Guide, for detailed information on creation and use of reservations.

The time for which a reservation is requested is in the time zone at the submission host.

4.8.37.2.2  Reservations and Placement Sets

When PBS chooses a placement set for a reservation, it makes the same choices as it would for a regular job. It fits the reservation into the smallest possible placement set. See section 4.8.32.4.2 “Order of Placement Set Consideration Within Pool” on page 243.
When a reservation is created, it is created within a placement set, if possible. If no placement set will satisfy the reservation, placement sets are ignored. The vnodes allocated to a reservation are used as one single placement set for jobs in the reservation; they are not subdivided into smaller placement sets. A job within a reservation runs within the single placement set made up of the vnodes allocated to the reservation.

### 4.8.37.2.3 Requesting Resources for Reservations

Reservations request resources using the same mechanism that jobs use. If a resource is unrequestable, users cannot request it for a reservation. If a resource is invisible, users cannot view it or request it for a reservation.

### 4.8.37.2.4 Reservations and Provisioning

Users can create reservations that request AOEIs. Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE. See section 8.4.3 “Provisioning And Reservations” on page 606.

The vnodes allocated to a reservation that requests an AOE are put in the `resv-exclusive` state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

For information on restrictions applying to reservations used with provisioning, see section 8.7.2.3 “Vnode Reservation Restrictions” on page 628.

For how to avoid problems with provisioning and reservations, see section 8.10.1 “Using Provisioning Wisely” on page 645.

### 4.8.37.2.5 Reservation Priority

A job running in a reservation cannot be preempted.

A job running in a reservation has the highest execution priority.

### 4.8.37.3 Querying Reservations

To query a reservation, use the `pbs_rstat` command. See section 7.8.6, "Viewing the Status of a Reservation", on page 189 of the PBS Professional User’s Guide. To delete an advance reservation, use the `pbs_rdel` command, not the `qmgr` command.
Controlling Access to Reservations

You can specify which users and groups can and cannot submit jobs to reservations. Use the `qmgr` command to set the reservation queue’s `acl_users` and/or `acl_groups` attributes. See section 9.3 “Using Access Control” on page 659.

Reservation Fault Tolerance

PBS automatically keeps track of the vnodes assigned to reservations, and tries to find replacement vnodes for those that become unavailable. See section 10.5 “Reservation Fault Tolerance” on page 774.

Advance and Standing Reservations and Licensing

Reservation jobs won’t run if PBS runs out of licenses. Set the server’s `pbs_license_min` attribute to the total number of CPUs, including virtual CPUs, in the PBS complex. See section 4.7.4 “Licensing and Reservations” on page 121 in the PBS Professional Installation & Upgrade Guide and section 4.4.3 “Setting Server Licensing Attributes” on page 108 in the PBS Professional Installation & Upgrade Guide.

Logging Reservation Information

The start and end of each occurrence of a standing reservation is logged as if each occurrence were a single advance reservation.

Reservation-related messages are logged at level 0x0200 (512).

Accounting

Resources requested for a reservation are recorded in the reservation’s `Resource_List` attribute, and reported in the accounting log B record for the reservation. The accounting log B record is written at the beginning of a reservation.
4.8.37.9 Attributes Affecting Reservations

Most of the attributes controlling a reservation are set when the reservation is created by the user. However, some server and vnode attributes also control the behavior of reservations.

The server attributes that affect reservations are listed here, and described in “Server Attributes” on page 365 of the PBS Professional Reference Guide.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl_resv_host_enable</td>
<td>Controls whether or not the server uses the acl_resv_hosts access control lists.</td>
</tr>
<tr>
<td>acl_resv_hosts</td>
<td>List of hosts from which reservations may and may not be created at this server.</td>
</tr>
<tr>
<td>acl_resv_group_enable</td>
<td>Controls whether or not the server uses the acl_resv_groups access control lists.</td>
</tr>
<tr>
<td>acl_resv_groups</td>
<td>List of groups who may and may not create reservations at this server.</td>
</tr>
<tr>
<td>acl_resv_user_enable</td>
<td>Controls whether or not the server uses the acl_resv_users access control lists.</td>
</tr>
<tr>
<td>acl_resv_users</td>
<td>List of users who may and may not create reservations at this server.</td>
</tr>
<tr>
<td>resv_enable</td>
<td>Controls whether or not reservations can be created at this server.</td>
</tr>
</tbody>
</table>
The vnode attributes that affect reservations are listed here. See “Vnode Attributes” on page 420 of the PBS Professional Reference Guide for more information.

### Table 4-15: Vnode Attributes Affecting Reservations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Associates the vnode with an execution queue. If this attribute is set, this vnode cannot be used for reservations.</td>
</tr>
<tr>
<td>reserve_retry_cutoff</td>
<td>Cutoff time for reconfirmation retries before a degraded occurrence or advance reservation. After this cutoff, PBS will not try to reconfirm the occurrence or reservation.</td>
</tr>
<tr>
<td>reserve_retry_init</td>
<td>Length of time to wait between when a reservation becomes degraded and when PBS tries to reconfirm the reservation. Default: 2 hours</td>
</tr>
<tr>
<td>resv_enable</td>
<td>Controls whether the vnode can be used for reservations. Default is True, but set to False for a vnode used for cycle harvesting.</td>
</tr>
</tbody>
</table>

#### 4.8.37.10 Reservation Advice and Caveats

- Do not delete a reservation’s queue.
- Do not start a reservation’s queue (do not set the reservation’s started attribute to True). Jobs will run prematurely.
- Do not use `qmgr` to set attribute values for a reservation queue.
- Reservations are incompatible with cycle harvesting. Do not allow reservations on machines used for cycle harvesting. The user may begin using the machine, which will suspend any PBS jobs, possibly preventing them from finishing before the reservation runs out. Set each cycle harvesting vnode’s resv_enable attribute to False, to prevent the vnode from being used for reservations.
- You can write hooks that execute, modifying a reservation’s attributes, when a reservation is created. See “Hooks” on page 461.
- Allow enough time in reservations. If a job is submitted to a reservation...
with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.

- The `xpbs` GUI cannot be used for creation, querying, or deletion of reservations.
- Hosts or vnodes that have been configured to accept jobs only from a specific queue (vnode-queue restrictions) cannot be used for advance reservations. Hosts or vnodes that are being used for cycle harvesting should not be used for reservations.
- Hosts with `$max_load` and `$ideal_load` configured should not be used for reservations. Set the `resv_enable` vnode attribute on these hosts to `False`.
- For troubleshooting problems with reservations, see section 14.7.4 “Job in Reservation Fails to Run” on page 925.
- Be careful when using `qrun -H` on jobs or vnodes involved in reservations. Make sure that you don’t oversubscribe reserved resources.
- In order to create reservations, the submission host must have its timezone set to a value that is understood by the PBS server. See section 14.5.18 “Unrecognized Timezone Variable” on page 917.

### 4.8.38 Round Robin Queue Selection

PBS can select jobs from queues by examining the queues in round-robin fashion. When using the round-robin method, the scheduler considers the first queue, tries to run a job from that queue, then considers the next queue, tries to run a job from that queue, then considers the next queue, and so on, in a circular fashion.

You can specify whether PBS uses the round-robin method to select jobs, by setting the value of the `round_robin` scheduler attribute. When this attribute is `True`, PBS uses the round-robin method. See “`round_robin`” on page 342 of the PBS Professional Reference Guide.

The `round_robin` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.
The order in which queues are selected is determined by each queue’s priority, and whether you specify that queues should be sorted. You can set each queue’s priority; see section 2.2.5.3 “Prioritizing Execution Queues” on page 26. You can specify that queues should be sorted by setting the sort_queues scheduler parameter to True. See section 4.8.44 “Sorting Queues into Priority Order” on page 306. If queue priorities are not set, they are undefined. If you do not sort the queues, their order is undefined.

You can use the round-robin method as a resource allocation tool. For example, if you need to run the same number of jobs from each group, you can put each group’s jobs in a different queue, and then use round-robin to run jobs, one from each queue.

The round-robin method is also used in PBS for some features that are not controlled by the round_robin scheduler attribute. They are the following:

- Routing queues try destinations in round-robin fashion, in the order listed
- The SMP cluster distribution parameter, smp_cluster_dist, can use a round-robin method to place jobs

### 4.8.38.1 Round-robin Caveats

- Each scheduling cycle starts with the highest-priority queue. Therefore, when using round-robin, this queue gets preferential treatment.
- When set to True, the round_robin parameter overrides the by_queue parameter.
- If round robin and strict ordering are True, and the top job cannot run, whether because of resources or rejection by MOM, no job runs. However, if round robin is True and strict ordering is False, and the top job in the current queue cannot run, the top job in the next queue is consid-
erated instead. For example, we have 3 queues, each with 3 jobs:

Q1: J1 J2 J3
Q2: J4 J5 J6
Q3: J7 J8 J9

If `round_robin` and `strict_ordering` are `True`, and J1 cannot run, no job runs.

If `round_robin` is `True` and `strict_ordering` is `False`, and J1 cannot run, job order is J4, J7, J2, J5, J8, J3, etc.

• With `round_robin` and `strict_ordering` set to `True`, a job continually rejected by a runjob hook may prevent other jobs from being run. A well-written hook would put the job on hold or requeue the job at some later time to allow other jobs in the same queue to be run.

### 4.8.39 Routing Jobs

Before reading this section, please read about the mechanics of configuring and using routing queues, in section 2.2.6 “Routing Queues” on page 26.

In this section, we use the term “routing” to mean the general process of moving a job somewhere, whether it is from one queue to another, from one complex to another, or from a queue to particular vnodes.

Routing jobs can involve collecting jobs so they don’t stray into the wrong queues, moving those jobs to the correct queues, and filtering which jobs are allowed into queues.

You may need to collect jobs into a routing queue, before moving them to the correct destination queue. If you use a routing queue, you can force users to submit jobs to the routing queue only, you can grab jobs as they are submitted and put them in the routing queue, and you can set a routing queue as the default. The mechanisms to collect jobs are described below, and listed here:

• Setting default queue; see section 4.8.39.1.1 “Default Queue as Mechanism to Collect Jobs” on page 294
• Grabbing jobs upon submission; see section 4.8.39.1.2 “Grabbing Jobs
Upon Submission” on page 294

- Disallowing direct submission to execution queues; see section 4.8.39.1.3 “Disallowing Direct Submission as Mechanism to Collect Jobs” on page 294

- Disallowing submission using access controls; see section 4.8.39.3.2 “Access Controls as Filtering Mechanism” on page 297

You can use any of several mechanisms for moving jobs. Each is described in subsections below. The mechanisms for moving jobs are the following:

- Routing Queues; see section 4.8.39.2.1 “Routing Queues as Mechanism to Move Jobs” on page 295

- Hooks; see section 4.8.39.2.2 “Hooks as Mechanism to Move Jobs” on page 295

- Peer scheduling; see section 4.8.39.2.3 “Peer Scheduling as Mechanism to Move Jobs” on page 296

- The `qmove` command; see section 4.8.39.2.4 “The qmove Command as Mechanism to Move Jobs” on page 296

You can use filtering methods to control which jobs are allowed into destination queues. We describe filtering methods in subsections below. The filtering mechanisms are the following:

- Resource limits; jobs are filtered by resource request. See section 4.8.39.3.1 “Resource Limits as Filtering Mechanism” on page 296

- Access control limits; jobs are filtered by owner. See section 4.8.39.3.2 “Access Controls as Filtering Mechanism” on page 297

You can use a combination of moving a job and “tagging” it, that is, including a special custom resource in the job’s resource request, to route the job. If you set the resource using a hook, you can route the job either to a queue or to vnodes. If you make the job inherit the resource from a queue, you can route it only to vnodes. You can set resource limits for the special custom resource at the receiving queue, allowing in only jobs with the special resource. You can set the special custom resource at vnodes, so that the job must run there. Mechanisms for tagging jobs are listed here:

- Using a hook; see section 4.8.39.4.1 “Using Hooks to Tag Jobs” on page 297

- Associating vnodes with queues; see section 4.8.2.2 “Associating
4.8.39.1 Mechanisms for Collecting Jobs

4.8.39.1.1 Default Queue as Mechanism to Collect Jobs

You can make it easy on your users by having their jobs land in your routing queue by default. You probably don’t want frustrated users trying to submit jobs without specifying a queue, only to have the jobs be rejected if you have set access controls on, or only allowed routing to, the default queue. The server’s default_queue attribute specifies the name of the default queue. To make things easy, make the default queue be the routing queue:

```
Qmgr: set server default_queue = <queue name>
```

4.8.39.1.2 Grabbing Jobs Upon Submission

You can allow users to submit jobs to any queue, and then scoop up the newly-submitted jobs and put them in the desired queue. To do this, you write a hook. There is a hook of this kind the example “Redirecting newly-submitted jobs” on page 554.

4.8.39.1.3 Disallowing Direct Submission as Mechanism to Collect Jobs

If you are using a routing queue, you can disallow job submission to all other queues. This forces users to submit jobs to the routing queue. You should probably make the routing queue be the default queue in this case, to avoid irritating users. Whether or not a queue allows direct job submission is controlled by its from_route_only attribute. To disallow job submission to a queue:

```
Qmgr: set queue <queue name> from_route_only = True
```
4.8.39.2 Mechanisms for Moving Jobs

4.8.39.2.1 Routing Queues as Mechanism to Move Jobs

Routing queues are a mechanism supplied by PBS that automatically move jobs from a routing queue to another queue. You can direct which destination queues accept a job using these filters at each destination queue:

- Resource limits: you can set up execution queues designed for specific kinds of jobs, and then route each kind of job separately. For example, you can create two execution queues, and one routing queue, and route all jobs requesting large amounts of memory to one of the execution queues, and the rest of the jobs to the other queue. See section 2.2.6.4 “Using Resources to Route Jobs Between Queues” on page 28.

- Access control limits: you can set up destination queues that are designed for specific groups of users. Each queue accepts jobs only from a designated set of users or groups. For example, if you have three departments, Math, Physics, and Chemistry, the queue belonging to Math accepts only users from the Math department. See section 2.2.6.5 “Using Access Control to Route Jobs” on page 33.

When routing a job between complexes, the job’s owner must be able to submit a job to the destination complex.

For how to configure and use routing queues, see section 2.2.6 “Routing Queues” on page 26.

4.8.39.2.2 Hooks as Mechanism to Move Jobs

You can use a submission hook to move jobs into queues such as dedicated time queues, queues with special priority, or reservation queues. You write the hook so that it identifies the jobs that should go into a particular queue, and then moves them there. This is a snippet, where you would replace <destination queue> with the queue name.

```python
import pbs
e = pbs.event()
e.job.queue = pbs.server().queue("<destination queue>")
```

For complete information on hooks, see Chapter 6, "Hooks", on page 461.
4.8.39.2.3 Peer Scheduling as Mechanism to Move Jobs

To send jobs from one complex to another, you use peer scheduling. In peer scheduling, the complex that supplies the jobs (the “furnishing” complex) contains at least one special queue (the “furnishing queue”), whose jobs can be pulled over to another complex, to be run at the other complex. The complex that pulls jobs contains a special queue (the “pulling queue”), where those pulled jobs land.

You can use any of the job routing methods, such as routing queues, tagging, or hooks, to control which jobs land in the furnishing queue.

You can use any of the job filtering methods, such as resource limits or access controls, to control which jobs land in the furnishing queue.

You can use job submission hooks on the jobs that land in the pulling queue.

See section 4.8.31 “Peer Scheduling” on page 229.

4.8.39.2.4 The qmove Command as Mechanism to Move Jobs

You can use the qmove command, either manually or via a cron job or the Windows Task Scheduler, to move jobs into the desired queues. See “qmove” on page 201 of the PBS Professional Reference Guide.

4.8.39.3 Mechanisms for Filtering Jobs

4.8.39.3.1 Resource Limits as Filtering Mechanism

You can filter whether each job is accepted at the server or a queue based on the job’s resource request. For example, you can control which jobs are allowed to be submitted to the server, by limiting the amount of memory a job is allowed to request. You can do the same at execution queues. These limits apply regardless of the routing mechanism being used, and apply to jobs being submitted directly to the queue. See section 5.13 “Using Resources to Restrict Server, Queue Access” on page 351.
4.8.39.3.2 Access Controls as Filtering Mechanism

You can filter jobs whether each job is accepted at the server or a queue based on the job’s owner, or the job owner’s group. At each queue and at the server, you can create a different list of the users who can submit jobs and the users who cannot submit jobs. You can do the same for groups.

For example, you can set up a routing queue and several execution queues, where each execution queue has access controls allowing only certain users and groups. When PBS routes the jobs from the routing queue, it will route them into the execution queues that accept owners of the jobs. See section 2.2.6.5 “Using Access Control to Route Jobs” on page 33.

4.8.39.4 Mechanisms for Tagging Jobs

4.8.39.4.1 Using Hooks to Tag Jobs

You can use a hook to force certain jobs to run on particular hardware, by having the hook set the value of a host-level custom resource in a job’s resource request. The hook sets this resource to match the value at the selected vnodes, so that the job must run on one or more of those vnodes. Note that the value at other vnodes should be different, otherwise the job could end up on vnodes you don’t want.

- Define a host-level custom resource; see section 5.14.5 “Configuring Host-level Custom Resources” on page 373.
- Set this resource to a special value on the special vnodes only. See section 5.7.1 “Setting Values for Global Static Resources” on page 333.
- Create a hook that filters jobs by size or other characteristic, and sets the value of the custom resource to the special value, in the job’s resource request. See Chapter 6, "Hooks", on page 461.

If you must use a routing queue, and you need to route on host-level resources (resources in the job’s select specification), you can use a hook to tag jobs so that they are routed correctly. The hook reads the job’s host-level resource request, and sets the job’s server-level resource request accordingly. This server-level resource is used for routing:

- Create a custom server-level resource that you use exclusively for routing; set it to appropriate values on the destination queues; see section
5.14.4 “Configuring Server-level Resources” on page 370

- Create a submit hook to extract the host-level resource value and use it to populate the custom resource that you use exclusively for routing; see Chapter 6, "Hooks", on page 461

4.8.39.4.2 Using the qalter Command to Tag Jobs

You can change a job’s resource request using the qalter command. This way you can override normal behavior. See “qalter” on page 152 of the PBS Professional Reference Guide.

4.8.40 Shared vs. Exclusive Use of Resources by Jobs

When PBS places a job, it can do so on hardware that is either already in use or has no jobs running on it. PBS can make the choice at the vnode level or at the host level. How this choice is made is controlled by a combination of the value of each vnode’s sharing attribute and the placement requested by a job.

You can set each vnode’s sharing attribute so that the vnode or host is always shared, always exclusive, or so that it honors the job’s placement request. The value of a vnode’s sharing attribute takes precedence over a job’s placement request.

Each vnode can be allocated exclusively to one job (each job gets its own vnodes), or its resources can be shared among jobs (PBS puts as many jobs as possible on a vnode). If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes job-exclusive. No other job can use the vnode.

Hosts can also be allocated exclusively to one job, or shared among jobs.

For a complete description of the sharing attribute, and a table showing the interaction between the value of the sharing attribute and the job’s placement request, see “sharing” on page 426 of the PBS Professional Reference Guide.
4.8.40.0.1 Sharing on a Shared-memory Altix

On a shared-memory Altix, the scheduler will share memory from a chunk even if all the CPUs are used by other jobs. It will first try to put a chunk entirely on one vnode. If it can, it will run it there. If not, it will break the chunk up across any vnode it can get resources from, even for small amounts of unused memory.

4.8.40.1 Setting the sharing Vnode Attribute

When setting the sharing vnode attribute, follow the rules in section 3.5.2 “Choosing Configuration Method” on page 58.

4.8.40.2 Viewing Sharing Information

You can use the qmgr or pbsnodes commands to view sharing information. See “qmgr” on page 178 of the PBS Professional Reference Guide and “pbsnodes” on page 123 of the PBS Professional Reference Guide.

4.8.40.3 Sharing Caveats

- On the Cray, the sharing attribute is set to force_exclhost by default. Do not change this setting.
- The term “sharing” is also used to describe the case where MOM manages a resource that is shared among her vnodes, for example a license shared by the vnodes of a multi-vnode machine.
- The term “sharing” is also used to mean oversubscribing CPUs, where more than one job is run on one CPU; the jobs are “sharing” a CPU. See section 10.4.5 “Managing Load Levels on Vnodes” on page 768.
- If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either default_exclhost or force_exclhost, all vnodes on that host must have the same value for the sharing attribute. When the MOM starts or restarts, if any vnode on a host is set to either default_exclhost or force_exclhost, and another vnode is set to a different value, the MOM will exit and log the following error message at event class
0x0001:

It is erroneous to mix sharing= <sharing val> for vnode <name> with sharing= <force_exclhost|default_exclhost> which is set for other vnodes on host <host>

- For vnodes with sharing=default_shared, jobs can share a vnode, so that unused memory on partially-allocated vnodes is allocated to a job. The exec_vnode attribute will show this allocation.

4.8.41 SMP Cluster Distribution

PBS provides a method for distributing single-chunk jobs to a cluster of single-vnode machines according to a simple set of rules. The method is called SMP cluster distribution. It takes into account the resources specified on the resources: line in PBS_HOME/sched_priv/sched_config. The SMP cluster distribution method allows you to choose one of three job distribution systems:

Table 4-16: SMP Cluster Distribution Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>pack</td>
<td>Pack all jobs onto one vnode, until that vnode is full, then move to the next vnode</td>
</tr>
<tr>
<td>round_robin</td>
<td>Place one job on each vnode in turn, before cycling back to the first vnode</td>
</tr>
<tr>
<td>lowest_load</td>
<td>Place the job on the host with the lowest load average</td>
</tr>
</tbody>
</table>

4.8.41.1 How to Use SMP Cluster Distribution

To use SMP cluster distribution, do the following:

- Set the smp_cluster_dist scheduler parameter to the desired value. For example, to enable SMP cluster distribution using the round robin algorithm during primetime, and the pack algorithm during non-prime-
time, set the following in the scheduler’s configuration file:

```
smp_cluster_dist: round_robin prime
smp_cluster_dist: pack non_prime
```

- Set `resources_available.<resource>` to the desired limit on each vnode. You do not need to set any of the resources that are automatically set by PBS. For a list of these, see section 5.6.1.1 “Default Behavior of Vnode Resources” on page 330.

- Specify the resources to use during scheduling, in `PBS_HOME/sched_priv/sched_config`:
  
  ```
  resources: “ncpus, mem, arch, host, ...”
  ```

The `smp_cluster_dist` parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

### 4.8.41.2 How To Disable SMP Cluster Distribution

To ensure that SMP cluster distribution does not interfere with your scheduling policy, leave the `smp_cluster_dist` parameter set to its default value:

```
smp_cluster_dist pack all
```

### 4.8.41.3 SMP Cluster Distribution Caveats and Advice

- This feature was intended for early implementations of complexes, and probably is not useful for you.

- If you use this feature, you are committed to using it for the entire complex; you cannot designate some machines where it will be used and others where it will not be used.

- If `smp_cluster_dist` with either `round_robin` or `lowest_load` is used with `node_sort_key` set to `unused` or `assigned`, `smp_cluster_dist` is set to `pack`.

- The `avoid_provision` provisioning policy is incompatible with the `smp_cluster_dist` scheduler configuration parameter. If a job requests an AOE, the `avoid_provision` policy overrides the behavior of `smp_cluster_dist`.

- This feature is applied only to single-chunk jobs that specify an arrange-
ment of pack. Multi-chunk jobs are ignored.

- This feature is useful only for single-vnode machines. On a multi-
  vnoded machine, this feature distributes jobs across vnodes, but those
  jobs can end up all stuck on a single host.

- The choice of `smp_cluster_dist` with `round_robin` can be replaced by
  sorting vnodes according to unused CPUs, which does a better job:

  `node_sort_key: “ncpus HIGH unused”`

### 4.8.42 Sorting Jobs on a Key

PBS allows you to sort jobs on a key that you specify. This can be used
when setting both execution and preemption priority. Sorting jobs comes
into play after jobs have been divided into classes, because each class may
contain more than one job. You can sort on one or more of several different
keys, and for each key, you can sort either from low to high or from high to
low.

You configure sorting jobs on a key by setting values for the `job_sort_key`
scheduler parameter.

You can create an invisible, unrequestable custom resource, and use a hook
to set the value of this resource for each job. The hook would modify the
job’s resource request to include the new resource, and set the value to
whatever the hook computes. Then you can sort jobs according to the
value of this resource.

The `job_sort_key` parameter is a primetime option, meaning that you can
configure it separately for primetime and non-primetime, or you can spec-
ify it for all of the time.

#### 4.8.42.1 `job_sort_key` Syntax

`job_sort_key: “<sort key> HIGH | LOW <primetime option>”`
You can use the following keys for sorting jobs:

### Table 4-17: Keys for Sorting Jobs

<table>
<thead>
<tr>
<th>Sort Key</th>
<th>Allowed Order</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;PBS resource&gt;</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>fair_share_perc</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>job_priority</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>preempt_priority</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>sort_priority</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
</tbody>
</table>

You can sort on up to 20 keys.

The argument to the job_sort_key parameter is a quoted string. The default for job_sort_key is that it is not in force.

See “job_sort_key” on page 332 of the PBS Professional Reference Guide.

#### 4.8.42.2 Configuring Sorting Jobs on a Key

You can specify more than one sort key, where you want a primary sort key, a secondary sort key, etc.

If you specify more than one entry for job_sort_key, the first entry is the primary sort key, the second entry is the secondary sort key, which is used to sort equal-valued entries from the first sort, and so on.

Each entry is specified one to a line.
To sort jobs on a key, set the `job_sort_key` scheduler parameter:

- Set the desired key
- Specify whether high or low results should come first
- Specify the primetime behavior

The scheduler’s configuration file is read on startup and HUP.

### 4.8.42.3 Examples of Sorting Jobs on Key

**Example 1:** Sort jobs so that those with long walltime come first:

```
job_sort_key: “walltime HIGH”
```

**Example 2:** For example, if you want big jobs to run first, where “big” means more CPUs, and if the CPUs are the same, more memory, sort on the number of CPUs requested, then the amount of memory requested:

```
job_sort_key: “ncpus HIGH” all
job_sort_key: “mem HIGH” all
```

**Example 3:** Sort jobs so that those with lower memory come first:

```
job_sort_key: “mem LOW” prime
```

**Example 4:** Sort jobs so that those with highest preemption priority come first:

```
job_sort_key: “preempt_priority HIGH”
```

**Example 5:** Sort jobs according to the value of an invisible custom resource called `JobOrder`:

```
job_sort_key: “JobOrder LOW” all
```

### 4.8.42.4 Caveats and Advice for Sorting Jobs on Key

- Do not use `fair_share_perc` as the sort key when using fairshare, meaning the `fair_share` scheduler parameter is enabled. If you do this, the scheduler will attempt to sort a set of jobs where each job has the same sort key value. This will not sort the jobs.
- Use the `fair_share_perc` option only when ordering jobs by entity

- If you are sorting jobs using job_sort_key, we recommend that when you change preempt_prio from the default, you add preempt_priority as the primary key for the job_sort_key scheduler parameter. This is especially important when using soft limits for preemption. Jobs that are over their soft limits should have lower priority, both in execution and preemption. For example, if you’re using soft limit preemption levels, but not assigning job execution priority according to soft limits, you can have job A started and preempted in the same scheduler cycle, because job A has higher execution priority, but is over its soft limits, and has lower preemption priority. To sort jobs on preemption priority:
  
  ```bash
  job_sort_key: "preempt_priority HIGH"
  ```

- To run big jobs first, use ncpus as the primary sort key for job_sort_key:
  
  ```bash
  job_sort_key: "ncpus HIGH"
  ```

- The job_sort_key parameter is overridden by the job sorting formula and by fairshare. It is invalid to set both job_sort_formula and job_sort_key at the same time. If they are both set, job_sort_key is ignored and the following error message is logged:
  
  “Job sorting formula and job_sort_key are incompatible. The job sorting formula will be used.”

- The scheduler’s configuration file contains an example line for job_sort_key. This line is commented out, but shows an example of job_sort_key with “cput” as the sorting key.

### 4.8.43 Sorting Jobs by Requested Priority

You can sort jobs according to the priority that was requested for the job. This value is found in the job’s Priority attribute. You can use this value in the following ways:

- The term job_priority represents the value of the job’s priority attribute in the job sorting formula. See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.

- The job_sort_key scheduler parameter can take the term job_priority
as an argument. The term *job_priority* represents the value of the job’s *Priority* attribute. See section 4.8.42 “Sorting Jobs on a Key” on page 302.

You can use a hook to set or change the value of a job’s *Priority* attribute. See section “Hooks” on page 461.

### 4.8.44 Sorting Queues into Priority Order

You can tell PBS to sort all the execution queues in your complex according to their priority, and then to use that ordering when examining queues individually. Using this, queues are ordered with the highest-priority queue first.

If you want queues to be considered in a specific order, you must assign a different priority to each queue, and set the *sort_queues* scheduler parameter to *True*. When the *sort_queues* scheduler parameter is *True*, the scheduler sorts the queues according to their priority.

To do this, give the queue you want considered first the highest priority, then the next queue the next highest priority, and so on. To set a queue’s priority, use the *qmgr* command to assign a value to the *priority* queue attribute.

```
qmgr: set queue <queue name> priority = <value>
```

The *sort_queues* parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time. See “sort_queues” on page 343 of the PBS Professional Reference Guide.

Sorting queues into priority order is useful for the following:

- Examining queues one at a time. See section 4.8.4 “Examining Jobs Queue by Queue” on page 151.
- Selecting jobs from queues in a round-robin fashion. See section 4.8.38 “Round Robin Queue Selection” on page 290.
4.8.44.1 Caveats and Advice when Sorting Queues

- If you do not set queue priorities, their ordering is undefined.
- The sort_queues parameter is useful only when PBS is looking at individual queues, such as when using by_queue or round_robin. If all jobs in the complex are being considered as if they were in one queue, sort_queues has no effect.

4.8.45 Starving Jobs

PBS can keep track of the amount of time a job has been waiting to run, and then mark the job as starving if this time has passed a specified limit. You can use this starving status in calculating both execution and preemption priority.

4.8.45.1 Enabling Starving

You enable tracking whether jobs are starving by setting the help_starving_jobs scheduler parameter to True.

You specify the amount of time required for a job to be considered starving in the max_starve scheduler parameter. The default for this parameter is 24 hours.

The help_starving_jobs parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time. See “help_starving_jobs” on page 331 of the PBS Professional Reference Guide.

4.8.45.2 Time Used for Starving

PBS can use one of the following kinds of time to determine whether a job is starving:

- The job’s eligible wait time, described in section 4.8.13 “Eligible Wait Time for Jobs” on page 173
- The amount of time the job has been queued
You specify which to use in the server’s `eligible_time_enable` attribute. When `eligible_time_enable` is set to `True`, each job’s `eligible_time` value is used as its wait time for starving. If `eligible_time_enable` is set to `False`, the amount of time the job has been queued is used as its wait time for starving. The default for `eligible_time_enable` is `False`.

If the server’s `eligible_time_enable` attribute is set to `False`, the following rules apply:

- The amount of time the job has been queued is used as its wait time for starving.
- Jobs lose their queue wait time whenever they are requeued, as with the `qrerun` command. This includes when they are checkpointed or requeued (but not suspended) during preemption.
- Suspended jobs do not lose their queue wait time. However, when they become suspended, the amount of time since they were submitted is counted towards their queue wait time. For example, if a job was submitted, then remained queued for 1 hour, then ran for 26 hours, then was suspended, if `max_starve` is 24 hours, then the job will become starving.

If the server’s `eligible_time_enable` attribute is set to `True`, the following rules apply:

- The job’s `eligible_time` value is used as its wait time for starving.
- Jobs do not lose their `eligible_time` when they are requeued.
- Jobs do not lose their `eligible_time` when they are suspended.

### 4.8.45.3 Starving and Job Priority

Starving is one of the job classes used by PBS to calculate job execution priority. If you enable starving jobs, PBS will classify starving jobs in the Starving class, which gives them greater than ordinary priority. See section 4.8.16 “Calculating Job Execution Priority” on page 187. Each job’s eligible wait time can also be used in the job sorting formula used to calculate job execution priority. See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.
Starving is one of the job classes that you can use when specifying how preemption should work. You can choose how much preemption priority is given to starving jobs when you set preemption levels. See section 4.8.33 “Using Preemption” on page 258.

4.8.45.4 Parameters and Attributes Affecting Starving

The following table lists the parameters and attributes that affect starving:

<table>
<thead>
<tr>
<th>Parameter or Attribute</th>
<th>Location</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>help_starving_jobs</td>
<td>PBS_HOME/sched_priv/sched_config</td>
<td>Controls whether long-waiting jobs are considered starving. When set to True, jobs can be starving. Default: True all</td>
</tr>
<tr>
<td>max_starve</td>
<td>PBS_HOME/sched_priv/sched_config</td>
<td>Amount of wait time for job to be considered starving. Default: 24 hours.</td>
</tr>
<tr>
<td>eligible_time_enable</td>
<td>Server attribute</td>
<td>Controls whether a job’s wait time is taken from its eligible_time or from its queued time. When set to True, a job’s eligible_time is used as its wait time. Default: False.</td>
</tr>
<tr>
<td>eligible_time</td>
<td>Job attribute</td>
<td>The amount of time a job has been blocked from running due to lack of resources.</td>
</tr>
</tbody>
</table>

4.8.45.5 Starving and Queued or Running Jobs

When a job is running, it keeps the starving status it had when it was started. While a job is running, if it wasn’t starving before, it can’t become starving. However, it keeps its starving status if it became starving while queued.
4.8.45.6 Starving and Subjobs

Subjobs that are queued can become starving. Starving status is applied to individual subjobs in the same way it is applied to jobs. The queued subjobs of a job array can become starving while others are running. If a job array has starving subjobs, then the job array is starving.

4.8.45.7 Starving and Backfilling

Because a starving job can become a top job, but can continue to be unable to run due to a lack of resources, you may find it useful to use backfilling around starving jobs. See section 4.8.3 “Using Backfilling” on page 145.

4.8.45.8 Starving Caveats

Do not enable starving with fairshare, meaning do not set both the fair_share and help_starving_jobs scheduler parameters to True.

4.8.46 Using Strict Ordering

By default, when scheduling jobs, PBS orders jobs according to execution priority, then considers each job, highest-priority first, and runs the next job that can run now. Using strict ordering means that you tell PBS that it must not skip a job when choosing which job to run. If the top job cannot run, no job runs.

Strict ordering does not change how execution priority is calculated.

4.8.46.1 Configuring Strict Ordering

To configure strict ordering, set the strict_ordering scheduler parameter to True.

The strict_ordering parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time. See “strict_ordering” on page 344 of the PBS Professional Reference Guide.
4.8.46.2 How Strict Ordering Works

Strict ordering does not affect how jobs are prioritized. When strict_ordering is True, the scheduler runs jobs in exactly the order of their priority. If jobs are being run queue by queue, that order is honored.

4.8.46.3 Combining Strict Ordering and Backfilling

Strict ordering alone may cause some resources to stand idle while the top job waits for resources to become available. If you want to prevent this, you can use backfilling with strict ordering. Using backfilling, if the top job cannot run, filler jobs can be squeezed in around the job that cannot run. See section 4.8.3 “Using Backfilling” on page 145.

4.8.46.4 Strict Ordering Caveats

- It is inadvisable to use strict ordering and backfilling with fairshare. The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group will be chosen as the filler job. The usage from these filler jobs will lower the priority of the top job.

  For example, if a user has a large job that is the top job, and that job cannot run, smaller jobs owned by that user will chew up the user's usage, and prevent the large job from being likely to ever run. Also, if the small jobs are owned by a user in one area of the fairshare tree, no large jobs owned by anyone else in that section of the fairshare tree are likely to be able to run.

- Using dynamic resources with strict ordering and backfilling may result in unpredictable scheduling. See section 4.8.3.8 “Backfilling Recommendations and Caveats” on page 149.

- Using preemption with strict ordering and backfilling may change which job is being backfilled around.

- With both round robin and strict ordering, a job continually rejected by a runjob hook may prevent other jobs from being run. A well-written hook would put the job on hold or requeue the job at some later time to allow other jobs in the same queue to be run.
4.8.47 Sorting Vnodes on a Key

PBS can sort vnodes according to a key that you specify. This can be used when deciding which vnodes to use for jobs. Sorting vnodes comes into play after a placement set has been selected, or when a job will run on vnodes associated with a queue, or when placement sets are not used, because in those cases there may be more vnodes available than are needed. You can sort vnodes on one or more different keys, and for each key, you can sort from high to low, or the reverse. The default way to sort vnodes is according to the value of the vnode **priority** attribute, from higher to lower.

When you sort vnodes according to the assigned or unused amount of a resource, the vnode list is re-sorted after every job is run. This is because each job may change the usage for that resource.

You configure sorting vnodes on a key by setting values for the **node_sort_key** scheduler parameter.

The **node_sort_key** parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

When vnodes are not sorted on a key, their order is undefined.

4.8.47.1 node_sort_key Syntax

```
node_sort_key: "sort_priority HIGH | LOW" <prime option>
node_sort_key: "<resource> HIGH | LOW" <prime option>
node_sort_key: "<resource> HIGH | LOW total | assigned | unused" <prime option>
```

where

**total**

Use the **resources_available** value

**assigned**

Use the **resources_assigned** value
**unused**

Use the value given by `resources_available - resources_assigned`

Specifying a resource such as `mem` or `ncpus` sorts vnodes by the resource specified.

Specifying the `sort_priority` keyword sorts vnodes on the vnode priority attribute.

The default third argument for a resource is `total`. If the third argument, `total | assigned | unused`, is not specified with a resource, `total` is used. This provides backwards compatibility with previous releases.

The values used for sorting must be numerical.

### 4.8.47.2 Configuring Sorting Vnodes on a Key

You can specify up to 20 sort keys, where you want a primary sort key, a secondary sort key, etc.

If you specify more than one entry for `node_sort_key`, the first entry is the primary sort key, the second entry is the secondary sort key, which is used to sort equal-valued entries from the first sort, and so on.

Each entry is specified one to a line.

To sort jobs on a key, set the `node_sort_key` scheduler parameter:

- Set the desired key
- Specify whether high or low results should come first
- For sorting on a resource, optionally specify total, assigned, or unused
- Specify the primetime behavior

The scheduler’s configuration file is read on startup and HUP.

The argument to the `node_sort_key` parameter is a quoted string. The default for `node_sort_key` is the following:

```
node_sort_key: “sort_priority HIGH” all
```

See “node_sort_key” on page 335 of the PBS Professional Reference Guide.
4.8.47.2.1 Examples of Sorting Vnodes

Example 1: This sorts vnodes by the highest number of unused CPUs:
```
node_sort_key: "ncpus HIGH unused" all
```

Example 2: This sorts vnodes by the highest amount of memory assigned to vnodes, but only during primetime:
```
node_sort_key: "mem HIGH assigned" prime
```

Example 3: This sorts vnodes according to speed. You want to run jobs on the fastest host available. You have 3 machines, where HostA is fast, HostB is medium speed, and HostC is slow.

Set node priorities so that faster machines have higher priority:
```
Qmgr: set node HostA priority = 200
Qmgr: set node HostB priority = 150
Qmgr: set node HostC priority = 100
```

Specify that vnodes are sorted according to priority, with highest priority first:
```
node_sort_key: "sort_priority HIGH" ALL
```

Example 4: The old “nodepack” behavior can be achieved by this:
```
node_sort_key: "ncpus low unused"
```

Example 5: In this example of the interactions between placement sets and node_sort_key, we have 8 vnodes numbered 1-8. The vnode priorities are the same as their numbers. However, in this example, when
unsorted, the vnodes are selected in the order 4, 1, 3, 2, 8, 7, 5, 6. This is to illustrate the change in behavior due to node_sort_key.

We use:

node_sort_key: “sort_priority LOW”

Using node_sort_key, the vnodes are sorted in order, 1 to 8. We have three placement sets:

A: 1, 2, 3, 4 when sorted by node_sort_key; 4, 1, 3, 2 when no node_sort_key is used

B: 5, 6, 7, 8 when sorted by node_sort_key; 8, 7, 5, 6 when no node_sort_key is used

C: 1-8 when sorted, 4, 1, 3, 2, 8, 7, 5, 6 when not sorted.

A 6-vnode job will not fit in either A or B, but will fit in C. Without the use of node_sort_key, it would get vnodes 4, 1, 3, 2, 8, 7. With node_sort_key, it would get vnodes 1 - 6, still in placement set C.

4.8.47.2.2 Caveats for Sorting Vnodes

- Sorting on a resource with node_sort_key and using “unused” or “assigned” cannot be used with load_balancing. If both are used, load balancing will be disabled.

- Sorting on a resource and using “unused” or “assigned” cannot be used with smp_cluster_dist when it is set to anything but “pack”. If both are used, smp_cluster_dist will be set to “pack”.
Chapter 5

PBS Resources

This chapter contains information about resources, including providing resources for user jobs, setting up resources such as application licenses and scratch space, and how resources are used, defined, inherited, and viewed.

The PBS Professional Reference Guide contains resource reference material. For a list of built-in and custom Cray resources, as well as information on using resources, see “Resources” on page 347 of the PBS Professional Reference Guide. For a description of the format of each type of resource, see “Formats” on page 455 of the PBS Professional Reference Guide.

5.1 Introduction

PBS resources represent things such as CPUs, memory, application licenses, switches, scratch space, and time. They can also represent whether or not something is true, for example, whether a machine is dedi-
Chapter 5  
PBS Resources

cated to a particular project. PBS provides a set of built-in resources, and allows you to define additional custom resources. For some systems, PBS creates specific custom resources; see “Custom Cray Resources” on page 357 of the PBS Professional Reference Guide. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS can enforce limits on resource usage by jobs. The administrator can specify which resources are available at the server, each queue, and each vnode.

5.2 Chapter Contents

5.1 Introduction ............................................. 317
5.2 Chapter Contents ........................................ 318
5.3 Terminology ............................................. 320
5.4 Categories of Resources ............................... 324
  5.4.1 Built-in and Custom Resources ...................... 324
  5.4.2 Server, Queue, and Vnode Resources .............. 324
  5.4.3 Consumable and Non-consumable Resources ....... 325
  5.4.4 Static and Dynamic Resources ..................... 326
  5.4.5 Global and Local Resources ......................... 326
  5.4.6 Requested and Default Resources ................... 328
  5.4.7 Shared and Non-shared Vnode Resources .......... 328
  5.4.8 Platform-specific and Generally Available Resources .... 329
5.5 Resource Types ......................................... 329
5.6 Behavior of Resources ................................. 330
  5.6.1 Default Behavior .................................. 330
  5.6.2 How the Scheduler Uses Resources ................. 332
  5.6.3 Resource Names .................................. 332
  5.6.4 Resource Values .................................. 332
5.7 How to Set Resource Values ............................ 332
  5.7.1 Setting Values for Global Static Resources ....... 333
  5.7.2 Setting Values for Local Static Resources ........ 333
  5.7.3 Setting Values for String Arrays ................. 333
  5.7.4 Resource Value Caveats ............................ 334
5.8 Overview of Ways Resources Are Used .............. 335
  5.8.1 Advice on Using Resources ....................... 336
Chapter 5  

PBS Resources

5.17.3  Finding Current Value .................................................. 456
5.17.4  Restrictions on Viewing Resources .............................. 456
5.18   Resource Recommendations and Caveats ................. 456

5.3  Terminology

Advance reservation
A reservation for a specific set of resources for a specified start time and duration in the future. Advance reservations are created by users to reserve resources for jobs. The reservation is available only to the creator of the reservation and any users or groups specified by the creator.

Borrowing vnode
A shared vnode resource is available for use by jobs at more than one vnode, but is managed at just one vnode. A borrowing vnode is a vnode where a shared vnode resource is available, but not managed.

Built-in resource
A resource that is defined in PBS Professional as shipped. Examples of built-in resources are ncpus, which tracks the number of CPUs, and mem, which tracks memory. See section 5.4.1 “Built-in and Custom Resources” on page 324.

Chunk
A set of resources allocated as a unit to a job. Specified inside a selection directive. All parts of a chunk come from the same host. In a typical MPI (Message-Passing Interface) job, there is one chunk per MPI process.

Consumable resource
A consumable resource is a resource that is reduced or taken up by being used. Examples of consumable resources are memory or CPUs. See section 5.4.3 “Consumable and Non-consumable Resources” on page 325.
CPU

Has two meanings, one from a hardware viewpoint, and one from a software viewpoint:

1. A core. The part of a processor that carries out computational tasks. Some systems present virtual cores, for example in hyperthreading.

2. Resource required to execute a program thread. PBS schedules jobs according, in part, to the number of threads, giving each thread a core on which to execute. The resource used by PBS to track CPUs is called “ncpus”. The number of CPUs available for use defaults to the number of cores reported by the OS. When a job requests one CPU, it is requesting one core on which to run.

Custom resource

A resource that is not defined in PBS as shipped. Custom resources are created by the PBS administrator or by PBS for some systems. See section 5.4.1 “Built-in and Custom Resources” on page 324.

Floating license

A unit of license dynamically allocated (checked out) when a user begins using an application on some host (when the job starts), and deallocated (checked in) when a user finishes using the application (when the job ends).

Generic group limit

A limit that applies separately to groups at the server or a queue. This is the limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Generic user limit

A limit that applies separately to users at the server or a queue. This is the limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.
Global resource
A global resource is defined in a `resources_available` attribute, at the server, a queue, or a host. Global resources can be operated on via the `qmgr` command and are visible via the `qstat` and `pbsnodes` commands. See section 5.4.5 “Global and Local Resources” on page 326.

Group limit
Refers to configurable limits on resources and jobs. This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

Indirect resource
A shared vnode resource at vnode(s) where the resource is not defined, but which share the resource.

Individual group limit
Applies separately to groups at the server or a queue. This is the limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Individual user limit
Applies separately to users at the server or a queue. This is the limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.

Limit
A maximum that can be applied in various situations:

- The maximum amount of a resource that can be consumed at any time by running jobs
- The maximum amount of a resource that can be allocated to queued and running jobs,
- The maximum number of jobs that can be running
- The maximum number of jobs that can be queued
Limit for all users
Limit for everyone submitting jobs. In the context of server limits, this is the limit for users submitting jobs to the PBS complex. In the context of queue limits, this is the limit for everyone submitting jobs to the queue. A limit for all users is applied to the total usage by all users at the specified location. A separate limit for all users can be specified at the server and each queue.

Local resource
A local resource is defined in a Version 1 MOM configuration file. Local resources cannot be operated on via the qmgr command and are not visible via the qstat and pbsnodes commands. Local resources can be used by the scheduler. See section 5.4.5 “Global and Local Resources” on page 326.

Managing vnode
The vnode where a shared vnode resource is defined, and which manages the resource.

Memory-only vnode
Represents a node board that has only memory resources (no CPUs), for example, an Altix memory-only blade.

Non-consumable resource
A non-consumable resource is a resource that is not reduced or taken up by being used. Examples of non-consumable resources are Boolean resources and walltime. See section 5.4.3 “Consumable and Non-consumable Resources” on page 325.

Resource
A resource can be something used by a job, such as CPUs, memory, high-speed switches, scratch space, licenses, or time, or it can be an arbitrary item defined for another purpose. PBS has built-in resources, and allows custom-defined resources.

Shared resource
A vnode resource defined and managed at one vnode, but available for use at others.
Chapter 5  PBS Resources

User limit
Refers to configurable limits on resources and jobs. A user’s limit, whether generic or individual.

5.4 Categories of Resources

5.4.1 Built-in and Custom Resources

Built-in resources are the resources that are already defined for you in PBS. PBS supplies built-in resources including number of cpus, CPU time, and memory. For a list of built-in resources, see “Built-in Resources” on page 350 of the PBS Professional Reference Guide. Custom resources are those that you define, or that PBS creates for some systems. For example, if you wanted a resource to represent scratch space, you could define a resource called **Scratch**, and specify a script which queries for the amount of available scratch space. See section 5.14 “Custom Resources” on page 352.

5.4.2 Server, Queue, and Vnode Resources

PBS resources can be available at the server, queues, both the server and queues, or at vnodes. Any of these resources can be static or dynamic, built-in or custom, and consumable or non-consumable. Vnode resources can additionally be global or local.

5.4.2.1 Server Resources

A server resource, also called a server-level resource, is a resource that is available at the server. A server resource is available to be consumed or matched at the server if you set the server’s resources_available.<resource name> attribute to the available or matching value. For example, you can define a custom resource called **FloatingLicenses** and set the server’s resources_available.FloatingLicenses attribute to the number of available floating licenses.

A server resource is a job-wide resource. This means that a job can request this resource for the entire job, but not for individual chunks.
An example of a job-wide resource is shared scratch space, or any custom resource that is defined at the server and queue level.

5.4.2.2 Queue Resources

A queue resource, also called a queue-level resource, is available to be consumed or matched by jobs in the queue if you set the queue’s resources_available.<resource name> attribute to the available or matching value.

A queue resource is a job-wide resource. A job can request a queue resource for the entire job, but not for individual chunks.

An example of a job-wide resource is floating licenses, or any custom resource that is defined at both server and queue level.

5.4.2.3 Resources Defined at Both Server and Queue

Custom resources can be defined to be available either at vnodes, or at both the server and queues. Consumable custom resources that are defined at the server and queue level have their consumption monitored at the server and queue level. In our example, if a job requests one FloatingLicenses, then the value of the resources_assigned.FloatingLicenses attribute is incremented by one at both the server and the queue in which the job resides.

5.4.2.4 Vnode Resources

A vnode resource, also called a vnode-level or host-level resource, is available only at vnodes. A vnode resource is a chunk-level resource, meaning that it can be requested for a job only inside of a chunk.

5.4.3 Consumable and Non-consumable Resources

A consumable resource is one that is reduced by being used. Consumable resources include ncpus, mem and vmem by default, and any custom resource defined with the -n or -f flags.
A non-consumable resource is not reduced through use, meaning that allocation to one job does not affect allocation to other jobs. The scheduler matches jobs to non-consumable resources. Examples of non-consumable resources are walltime, file, cput, pcput, pmem, pvmem, nice, or Boolean resources.

## 5.4.4 Static and Dynamic Resources

Static resources are managed by PBS and have values that are fixed until you change them or until you change the hardware and MOM reports a new value for memory or number of CPUs.

Dynamic resources are not under the control of PBS, meaning that they can change independently of PBS. Dynamic resources are reported via a script; PBS runs a query to discover the available amount. Server dynamic resources use a script that runs at the server’s host. Host-level (MOM) dynamic resources use a script that runs at the execution host.

Static and dynamic resources can be available at the server or host level.

### 5.4.4.1 Dynamic Resource Caveats

Dynamic resource values are displayed in `qstat`, however, the value displayed is the last value retrieved, not the current value. Dynamic resources have no `resources_available.<resource>` representation anywhere in PBS.

## 5.4.5 Global and Local Resources

### 5.4.5.1 Global Static Resources

Global static resources are defined in `resources_available` attributes at the server, queue, or vnode, and are available at the server, queue, or vnode level. Global static resources can be operated on via the `qmgr` command and viewed via the `qstat` command. Values for built-in global static resources are set via the `qmgr` command. The walltime and aoe resources
are examples of global static resources. For custom global static resources, see section 5.14.2.6 “Example of Defining Each Type of Custom Resource” on page 365.

### 5.4.5.2 Global Dynamic Resources

Global dynamic resources are defined in the server’s resourcedef file, and can be used at the server, queue, or vnode level. Global host-level dynamic resources can be viewed via the qstat command. Server dynamic resource values have no resources_available.<resource> representation anywhere in PBS. See section 5.14.4.1 “Dynamic Server-level Resources” on page 370.

The value displayed via qstat for a dynamic resource is the most recently retrieved, not the current value.

### 5.4.5.3 Local Static Resources

It is not recommended to use local static resources. Local static resources are defined in the MOM Version 1 configuration file. These resources cannot be operated on via the qmgr command or viewed via the qstat command. They can be used by the scheduler.

### 5.4.5.4 Local Dynamic Resources

Dynamic local resources are defined in the MOM Version 1 configuration file. These are scripts that run on the execution host where they are defined and return a value. These resources can be used by the scheduler. Host dynamic resource values have no resources_available.<resource> representation anywhere in PBS. See section 5.14.5.1 “Dynamic Host-level Resources” on page 374.

The value displayed via qstat for a dynamic resource is the most recently retrieved, not the current value.
### 5.4.6 Requested and Default Resources

A job’s requested resources are the resources explicitly requested by the job. Default resources are resources that you specify that each job should have if not requested. For example, you can specify that any job that does not request `walltime` gets 12 hours of `walltime`. For jobs that do request `walltime`, the default of 12 hours is not applied.

For information on default resources, see section 5.9.3 “Specifying Job Default Resources” on page 338 and section 5.9.4 “Allocating Default Resources to Jobs” on page 342.

### 5.4.7 Shared and Non-shared Vnode Resources

#### 5.4.7.1 Non-shared Vnode Resources

Most vnode resources are not shared. When a resource is defined at one vnode for use by jobs only at that vnode, the resource is not shared. For example, when `resources_available.ncpus` is set to 4 on a single-vnode machine, and no other vnodes have `resources_available.ncpus` defined as a pointer to this resource, this resource is not shared.

#### 5.4.7.2 Shared Vnode Resources

When more than one vnode needs access to the same actual resource, that resource can be shared among those vnodes. The resource is defined at one vnode, and the other vnodes that supply the resource contain a pointer to that vnode. Any of the vnodes can supply that resource to a job, but only up to the amount where the total being used by jobs is less than or equal to the total available at the vnode where the resource is defined. For example, if you had a 4-vnode machine which had 8GB of memory, and wanted any single vnode to be able to supply up to 8GB to jobs, you would make the memory a shared resource. See section 5.14.5.3 “Shared Host-level Resources” on page 378.
5.4.8 Platform-specific and Generally Available Resources

Most PBS built-in resources are available on, and apply to, all supported platforms. However, PBS provides some resources specifically designed for a given platform. These platform-specific resources are not applicable to any other platform, and cannot be used on platforms other than the one(s) for which they are designed. For example, PBS creates custom resources that represent Cray elements, such as the Cray nid and the Cray label. PBS has several built-in resources whose names begin with mpp; these apply only to the Cray.

5.5 Resource Types

PBS supplies the following types of resources:

- Boolean
- duration
- float
- long
- size
- string
- string_array

See “List of Formats” on page 455 of the PBS Professional Reference Guide for a description of each resource type.
5.6 Behavior of Resources

5.6.1 Default Behavior

PBS automatically collects information about some resources and sets their initial values accordingly. If you explicitly set the value for a resource, that value is carried forth across Server restarts.

5.6.1.1 Default Behavior of Vnode Resources

PBS sets the value for certain resources at each vnode. This means that the value for the vnode’s resources_available.<resource name> attribute is set by PBS. The following table lists the vnode resources that are set automatically by PBS.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Initial Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
<td><em>Value reported by OS</em></td>
<td>Settable. If you unset the value, it remains unset until MOM is restarted.</td>
</tr>
<tr>
<td>host</td>
<td><em>Short form of hostname in Mom vnode attribute</em></td>
<td>Settable. If you unset the value, it remains unset until MOM is restarted.</td>
</tr>
<tr>
<td>mem</td>
<td><em>Amount reported by OS</em></td>
<td>Settable. If you unset the value, it remains unset until MOM is restarted.</td>
</tr>
<tr>
<td>ncpus</td>
<td><em>Number of CPUs reported by OS</em></td>
<td>Settable. If you unset this value, the MOM will reset it to the value reported by the OS.</td>
</tr>
</tbody>
</table>
For example, PBS automatically sets the value of `resources_available.ncpus` at each vnode.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Initial Value</th>
<th>Notes</th>
</tr>
</thead>
</table>
| PBScrayhost             | On CLE 2.2, set to default .  
                          | On CLE 3.0 and higher, set to value of mpp_host for this system              | Do not set.                                |
| PBScraylabel_&lt;label name&gt; | Concatenation of PBScraylabel_ and label name.  
                           | Set to True on all of node's vnodes . | Do not set.                                |
| PBScrayid               | Value of node_id for this compute node                                       | Do not set.                                |
| PBScrayorder            | Value starts at 1 and increments by 1 for each node in inventory              | Do not set                                |
| PBScrayseg              | Segment ordinal of associated NUMA node.                                     | Do not set                                |
| router                  | Name of router, from topology file                                           | Applies to vnodes on certain Altix machines only, such as the 4700.            |
| vnode                   | Name of the vnode                                                            | Vnode name must be specified via the qmgr create node command.                 |
5.6.1.2 Default Behavior of Server and Queue Resources

PBS automatically sets the value for `default_chunk.ncpus` to 1 at the server and queues.

5.6.1.3 Default Behavior of Job Resources

PBS automatically sets the value of the `estimated.start_time` job resource to the estimated start time for each job.

5.6.2 How the Scheduler Uses Resources

How the scheduler uses resources is described in section 4.8.28 “Matching Jobs to Resources” on page 222.

5.6.3 Resource Names

Resource names are case-insensitive. See “Resource Name” on page 460 of the PBS Professional Reference Guide for the format of resource names.

5.6.4 Resource Values

String resource values are case-sensitive. For format information, see “Resource Value” on page 460 of the PBS Professional Reference Guide.

5.7 How to Set Resource Values

Since the value for each dynamic resource is set by PBS to the value returned by a script or command, you will set values for static resources only.

You set values for custom and built-in resources using the same methods.
5.7.1 Setting Values for Global Static Resources

To set the value for a global vnode, queue, or server resource, use the `qmgr` command to set the value of the appropriate `resources_available.<resource>` attribute.

Example 1: Set the value of `floatlicenses` at the server to `10`:
```
Qmgr: set server resources_available.floatlicenses = 10
```

Example 2: Set the value of `RunsMyApp` to `True` at the vnode named `Vnode1`:
```
Qmgr: set node Vnode1
resources_available.RunsMyApp = True
```

5.7.1.1 Restrictions on Setting Values for Global Static Resources

When setting global static vnode resources on multi-vnode machines, follow the rules in section 3.5.2 “Choosing Configuration Method” on page 58.

5.7.2 Setting Values for Local Static Resources

It is not recommended to use local static resources, because these resources cannot be requested, and cannot be viewed using `qstat` or managed using `qmgr`. To set the value of a local vnode resource, edit `PBS_HOME/mom_priv/config` and change the value section of the resource’s line.

5.7.3 Setting Values for String Arrays

A string array that is defined on vnodes can be set to a different set of strings on each vnode.
Example of defining and setting a string array:

- Define a new resource:
  
  ```
  foo_arr type=string_array flag=h
  ```

- Setting via `qmgr`:
  
  ```
  Qmgr: set node n4 resources_available.foo_arr="f1, f3, f5"
  ```

  Vnode n4 has 3 values of `foo_arr`: f1, f3, and f5. We add f7:
  
  ```
  Qmgr: set node n4 resources_available.foo_arr+=f7
  ```

  Vnode n4 now has 4 values of `foo_arr`: f1, f3, f5 and f7.

  - We remove f1:
    
    ```
    Qmgr: set node n4 resources_available.foo_arr-=f1
    ```

  Vnode n4 now has 3 values of `foo_arr`: f3, f5, and f7.

  - Submission:
    
    ```
    qsub -l select=1:ncpus=1:foo_arr=f3
    ```

### 5.7.4 Resource Value Caveats

- It is not recommended to set the value for `resources_available.ncpus`. The exception is when you want to oversubscribe CPUs. See section 10.4.5.1.3 “How To Share CPUs” on page 770.

- Do not attempt to set values for `resources_available.<resource>` for dynamic resources.

- Do not set values for any resources, except those such as shared scratch space or floating licenses, at the server or a queue, because the scheduler will not allocate more than the specified value. For example, if you set `resources_available.walltime` at the server to 10:00:00, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.
5.7.4.1 Resource Value Caveats for Multi-vnode Machines

- When setting global static vnode resources on multi-vnode machines, follow the rules in section 3.5.2 “Choosing Configuration Method” on page 58.
- It is not recommended to change the value of ncpus at vnodes on a multi-vnoded machine.
- On multi-vnode machines, do not set the values for mem, vmem or ncpus on the natural vnode. If any of these resources has been explicitly set to a non-zero value on the natural vnode, set resources_available.ncpus, resources_available.mem and resources_available.vmem to zero on each natural vnode.

5.8 Overview of Ways Resources Are Used

Resources are used in several ways in PBS. The following table lists the ways resources are used, and gives links to the section describing each one:

<table>
<thead>
<tr>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation to and use by jobs</td>
<td>See section 5.9 “Resources Allocated to Jobs and Reservations” on page 336</td>
</tr>
<tr>
<td>Limiting job resource usage</td>
<td>See section 5.15.3 “Placing Resource Limits on Jobs” on page 436</td>
</tr>
<tr>
<td>Restricting access to server and queues</td>
<td>See section 5.13 “Using Resources to Restrict Server, Queue Access” on page 351</td>
</tr>
<tr>
<td>Routing jobs</td>
<td>See section 2.2.6.4 “Using Resources to Route Jobs Between Queues” on page 28</td>
</tr>
</tbody>
</table>
Chapter 5  

PBS Resources

Table 5-2: How Resources Are Used

<table>
<thead>
<tr>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describing topology and placing jobs</td>
<td>See section 5.11 “Using Resources for Topology and Job Placement” on page 350</td>
</tr>
<tr>
<td>Setting job execution priority</td>
<td>See section 5.12 “Using Resources to Prioritize Jobs” on page 350</td>
</tr>
<tr>
<td>Reserving resources ahead of time</td>
<td>See section 4.8.37 “Advance and Standing Reservations” on page 283.</td>
</tr>
<tr>
<td>Tracking and controlling allocation</td>
<td>See section 5.10 “Using Resources to Track and Control Allocation” on page 347</td>
</tr>
<tr>
<td>Determining job preemption priority</td>
<td>See section 4.8.33 “Using Preemption” on page 258</td>
</tr>
</tbody>
</table>

5.8.1    Advice on Using Resources

See “Advice on Using Resources” on page 347 of the PBS Professional Reference Guide for tips on using resources.

5.9    Resources Allocated to Jobs and Reservations

Resources allocated to jobs provide the job with items such as CPUs and memory to be consumed by the job’s processes, as well as qualities such as architecture and host. The resources allocated to a job are those that the job requests and those that are assigned to it through resource defaults that you define.

Jobs use resources at the job-wide and chunk level. Job-wide resources such as `walltime` or `vmem` are applied to and requested by the job as a whole. Chunk-level resources, such as `ncpus`, are applied and requested in individual chunks.
Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. See “Resources” on page 347 of the PBS Professional Reference Guide and section 3.3.1, "Rules for Submitting Jobs", on page 27 of the PBS Professional User’s Guide.

Jobs inherit resource defaults for resources not explicitly requested. See section 5.9.4 “Allocating Default Resources to Jobs” on page 342.

Chunk-level resources are made available at the host (vnode) level by defining them via resources_available.<resource> at the vnode, and are requested using -l select=<resource>=<value>.

Job-wide resources are made available by defining them via resources_available.<resource> at the queue or server. These resources are requested using -l <resource> =<value>.

The scheduler matches requested resources with available resources, according to rules defined by the administrator.

When a job is requesting a string array resource, it can request only one of the values set in the string array resource. The job will only be placed on a vnode where the job’s requested string matches one of the values of the string array resource. For example, if the resource named Colors is set to “red, blue, green” on vnode V1, and “red, blue” on V2:

- A job can request only one of “red”, “blue”, or “green”
- A job requesting Colors=green will only be placed on V1

5.9.1 Allocating Chunks

Chunks cannot be split across hosts. Chunks can be made up of vchunks. If a chunk is broken up over multiple vnodes, all participating vnodes must belong to the same execution host. Each vnode supplies a vchunk. These participating vnodes are supplying the vchunks that make up the chunk. A chunk defines a logical set of resources, for example, those needed for an MPI task. The resources must come from a single host, but if the requested resources exceed that of any one vnode, the physical resources can be taken from multiple vnodes on the same host.
5.9.2 Resources Requested by Job

The job’s Resource_List attribute lists the following resources requested by the job:

- Job-wide resources either explicitly requested by the job or inherited from defaults
- The following built-in chunk-level resources either explicitly requested by the job or inherited from defaults:
  - mpiprocs
  - ncpus
  - netwins
  - mem
  - vmem
- Custom vnode-level (chunk-level) resources that are global and have the n, q, or f flags set, either explicitly requested by the job or inherited from defaults

5.9.3 Specifying Job Default Resources

You can specify which resources are automatically added to job resource requests. When a job does not request a specific resource, the default value for that resource is automatically added to the job’s resource request.

The amount of each resource a job is allowed to use is the amount in its resource request. See section 5.15.3 “Placing Resource Limits on Jobs” on page 436. Therefore you may wish to add default limits on resource usage. This is done by adding default resources to the job’s resource request. For example, if a job does not request walltime, but you do not want jobs not specifying walltime to run for more than 12 hours, you can specify a default of 12 hours for walltime. Jobs that do specify walltime do not inherit this default; they keep their requested amount.

You can use default resources to manage jobs. For example, if you want to keep track of and limit the number of jobs using something such as a disk arm, you can have each job using the disk arm automatically request one
counting resource. Then you can place a limit on the amount of this resource that can be in use at one time. This technique is described in section 5.10 “Using Resources to Track and Control Allocation” on page 347.

Default resources can be defined for the server and for each queue. Default resources defined at the server are applied to all jobs. Default resources at a queue are applied only to the jobs that are in that queue.

Default resources on the server and queue can be job-wide, which is the same as adding -l <resource name> to the job’s resource request, or they can be chunk resources, which is the same as adding :<resource name>=<value> to a chunk.

Job-wide resources are specified via resources_default on the server or queue, and chunk resources are specified via default_chunk on the server or queue. You can also specify default resources to be added to any qsub arguments. In addition, you can specify default placement of jobs.

5.9.3.1 Specifying Job-wide Default Resources at Server

To specify a server-level job-wide default resource, use the qmgr command to set the server’s resources_default attribute:

Qmgr: set server resources_default.<resource>=<value>

For example, to set the default architecture on the server:

Qmgr: set server resources_default.arch=linux

5.9.3.2 Specifying Chunk Default Resources at Server

To specify a server-level chunk default resource, use the qmgr command to set the server’s default_chunk attribute:

Qmgr: set server default_chunk.<resource>=<value>

For example, if you want all chunks that don’t specify ncpus or mem to inherit the values you specify:

Qmgr: set server default_chunk.ncpus=1
Qmgr: set server default_chunk.mem=1gb
5.9.3.3 Specifying Job-wide Default Resources at Queue

To specify a default for a job-wide resource at a queue, use the \texttt{qmgr} command to set the queue’s \texttt{resources\_default} attribute:

\texttt{Qmgr: set queue <queue name> resources\_default.<resource> = <value>}

5.9.3.4 Specifying Chunk Default Resources at Queue

To specify a queue-level chunk default resource, use the \texttt{qmgr} command to set the queue’s \texttt{default\_chunk} attribute:

\texttt{Qmgr: set queue <queue name> default\_chunk.<resource>=<value>}

For example, if you want all chunks that don’t specify \texttt{ncpus} or \texttt{mem} to inherit the values you specify:

\texttt{Qmgr: set queue small default\_chunk.ncpus=1}
\texttt{Qmgr: set queue small default\_chunk.mem=512mb}

5.9.3.5 Specifying Default \texttt{qsub} Arguments

You can set defaults for any \texttt{qsub} arguments not explicitly requested by each job. You do this at the server by using the \texttt{qmgr} command to set the server’s \texttt{default\_qsub\_arguments} attribute:

\texttt{Qmgr: set server default\_qsub\_arguments=<string containing arguments>}

For example, to set the default \texttt{rerunnable} option in each job’s resource request, and the name of the job:

\texttt{Qmgr: set server default\_qsub\_arguments= ”-r y -N MyJob”}

Or to set a default Boolean in each job’s resource request so that jobs don’t run on \texttt{Red} unless they explicitly ask to do so:

\texttt{Qmgr: set server default\_qsub\_arguments=’”-l Red=False”}
5.9.3.6 Specifying Default Job Placement

You can specify job placement defaults at both the server and queue level. You use the `qmgr` command to set the `resources_default.place` attribute at the server or queue:

```
Qmgr: set queue <queue name> resources_default.place=<value>
```

For example, to set the default job placement for a queue:

```
Qmgr: set queue Q1 resources_default.place=free
```

When setting default placement involving a colon, enclose the value in double quotes:

```
Qmgr: set server resources_default.place="<value>"
```

For example, to set default placement at the server to `pack:shared`, do the following:

```
Qmgr: set server resources_default.place= "pack:shared"
```

See section 3.6, "Placing Jobs on Vnodes", on page 50 of the PBS Professional User’s Guide for detailed information about how `-lplace` is used.

5.9.3.7 Using Gating Values As Defaults

For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. If this is set at the queue, the queue value of `resources_max.<resource>` is used. If this is set only at the server, the job inherits the value set at the server. However, for `mpp*` resources, the job does not inherit the gating value. For example, if the job does not request `mppnpn`, and no defaults are set at the server and queue, but `resources_max.mppnpn` is set at the queue, the job does not inherit the queue’s value.

5.9.3.8 Default Resource Caveats

- While users cannot request custom resources that are created with the `r` flag, jobs can inherit these as defaults from the server or queue.
resources_default.<resource> attribute.

- A qsub or pbs_rsub hook does not have resources inherited from the server or queue resources_default or default_chunk as an input argument.

- For mpp* resources, the job does not inherit the gating value. For example, if the job does not request mppnppn, and no defaults are set at the server and queue, but resources_max.mppnppn is set at the queue, the job does not inherit the queue’s value.

### 5.9.4 Allocating Default Resources to Jobs

Jobs inherit default resources, job-wide or per-chunk, with the following order of precedence.

**Table 5-3: Order In Which Default Resources Are Assigned to Jobs**

<table>
<thead>
<tr>
<th>Order of assignment</th>
<th>Default value</th>
<th>Affects Chunks?</th>
<th>Job-wide?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default qsub arguments</td>
<td>If specified</td>
<td>If specified</td>
</tr>
<tr>
<td>2</td>
<td>Queue’s default_chunk</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Server’s default_chunk</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Queue’s resources_default</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Server’s resources_default</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Queue’s resources_max</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Server’s resources_max</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See section 5.9.3 “Specifying Job Default Resources” on page 338 for how to set these defaults.
For each chunk in the job's selection statement, first default qsub arguments are applied, then queue chunk defaults are applied, then server chunk defaults are applied. If the chunk does not contain a resource defined in the defaults, the default is added. The chunk defaults are specified in the default_chunk.<resource name> server or queue attribute.

For example, if the queue in which the job is enqueued has the following defaults defined,

\[
\begin{align*}
\text{default\_chunk.ncpus} &= 1 \\
\text{default\_chunk.mem} &= 2gb
\end{align*}
\]

then a job submitted with this selection statement:

\[
\text{select=2:ncpus=4+1:mem=9gb}
\]

will have this specification after the default_chunk elements are applied:

\[
\text{select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb}
\]

In the above, \text{mem=2gb} and \text{ncpus=1} are inherited from \text{default\_chunk}.

The job-wide resource request is checked against queue resource defaults, then against server resource defaults, then against the queue’s resources_max.<resource>, then against the server’s resources_max.<resource>. If a default or maximum resource is defined which is not specified in the resource request, it is added to the resource request.

### 5.9.4.1 Default Resource Allocation Caveats

Resources assigned from the default_qsub_arguments server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag, whether that resource was requested by the user or came from default_qsub_arguments. Be aware that creating custom resources with permission flags and then using these in the default_qsub_arguments server attribute can cause jobs to be rejected. See section 5.14.2.5 “Resource Permission Flags” on page 362.
5.9.4.2 Moving Jobs Between Queues or Servers Changes Defaults

If the job is moved from the current queue to a new queue or server, any default resources in the job’s Resource_List inherited from the queue or server are removed. This includes a select specification and place directive generated by the rules for conversion from the old syntax. If a job’s resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job’s resource list. If either select or place is missing from the job’s new resource list, it will be automatically generated, using any newly inherited default values.

Jobs may be moved between servers when peer scheduling is in operation. Given the following set of queue and server default values:

- Server
  
  \texttt{resources\_default.ncpus=1}

- Queue QA

  \texttt{resources\_default.ncpus=2}
  \texttt{default\_chunk.mem=2GB}

- Queue QB

  \texttt{default\_chunk.mem=1GB}
  \texttt{no default for ncpus}

The following illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

Example 1: Submission:

\texttt{qsub \_l ncpus=1 \_lmem=4gb}

- In QA:

  \texttt{select=1:ncpus=1:mem=4gb}

  - No defaults need be applied

- In QB:

  \texttt{select=1:ncpus=1:mem=4gb}

  - No defaults need be applied
Example 2: Submission:

```
qsub -l ncpus=1
```

- In QA:
  ```
  select=1:ncpus=1:mem=2gb
  ```
  - Picks up 2GB from queue default chunk and 1 ncpus from qsub
- In QB:
  ```
  select=1:ncpus=1:mem=1gb
  ```
  - Picks up 1GB from queue default_chunk and 1 ncpus from qsub

Example 3: Submission:

```
qsub -lmem=4gb
```

- In QA:
  ```
  select=1:ncpus=2:mem=4gb
  ```
  - Picks up 2 ncpus from queue level job-wide resource default and 4GB mem from qsub
- In QB:
  ```
  select=1:ncpus=1:mem=4gb
  ```
  - Picks up 1 ncpus from server level job-wide default and 4GB mem from qsub

Example 4: Submission:

```
qsub -lnodes=4
```

- In QA:
  ```
  select=4:ncpus=1:mem=2gb
  ```
  - Picks up a queue level default memory chunk of 2GB. (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)
- In QB:
  ```
  select=4:ncpus=1:mem=1gb
  ```
  (In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)
Example 5: Submission:

\[ qsub -l mem=16gb -l nodes=4 \]

- In QA:
  \[ select=4:ncpus=1:mem=4gb \]
  
  (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

- In QB:
  \[ select=4:ncpus=1:mem=4gb \]
  
  (In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the \texttt{ncpus}=1 is not inherited from the server default.)

### 5.9.5 Allocation of Dynamic Resources

#### 5.9.5.1 Dynamic Resource Allocation Caveats

When a job requests a dynamic resource, PBS checks to see how much of the resource is available, but cannot know how much will be used by another job while this job executes. This can lead to a resource shortage. For example, there is 20GB of scratch on a disk, no jobs are running, and a job requests 15GB. This job writes to 5GB during the first part of its execution, then another job requests 10GB. The second job is started by PBS, because there is 15GB available. Now there is a shortage of scratch space.

You can avoid this problem by configuring a static consumable resource to represent scratch space. Set it to the amount of available scratch space. See section 5.14.6.3 “Static Server-level Scratch Space” on page 384 and section 5.14.6.4 “Static Host-level Scratch Space” on page 385.

#### 5.9.5.2 Job Suspension and Resource Usage

When a job is suspended, PBS releases all of the job’s resources, including the licenses used by PBS for the job. This does not include the licenses used by the application, if any.

A job is resumed only when sufficient resources are available. When a person resumes a job, the job is not run until resources are available.
5.9.5.2.1 Suspension/resumption Resource Caveats

Dynamic resources can cause problems with suspension and resumption of jobs.

When a job is suspended, its resources are freed, but the scratch space written to by the job is not available.

A job that uses scratch space may not suspend and resume correctly. This is because if the job writes to scratch, and is then suspended, when PBS queries for available scratch to resume the job, the script may return a value too small for the job’s request. PBS cannot determine whether the job itself is the user of the scratch space; PBS can only determine how much is still unused. If a single suspended job has left less scratch space available than it requests, that job cannot be resumed.

The above is true for any dynamic resource, such as application licenses.

5.10 Using Resources to Track and Control Allocation

You can use resources to track and control usage of things like hardware and licenses. For example, you might want to limit the number of jobs using floating licenses or a particular vnode. There is more than one way to accomplish this.

Example 1: You can set a complex-wide limit on the number of jobs using a type of complex-wide floating license. This example uses a single queue for the entire complex. This method requires job submitters to
request one of a floatlicensecount resource in order to be able to use the license. To set a complex-wide limit, take the following steps:

1. Create a custom static integer license resource that will be tracked at the server and queue:
   
a. In PBS_HOME/server_priv/resourcedef, add the line:
      
      floatlicensecount type=long flag=q
   
b. Add the resource to the resources: line in PBS_HOME/sched_priv/sched_config:
      
      resources: "[...], floatlicensecount"

2. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

3. HUP the scheduler:
   
   kill -HUP <scheduler PID>

4. Set the available resource at the server using qmgr. If you have enough floating licenses for 4 jobs:
   
   Qmgr: set server
   
   resources_available.floatlicensecount = 4

5. Inform job submitters that jobs using they must request one job-wide floatlicensecount resource via the following:
   
   qsub -l floatlicensecount=1
   
   The scheduler will schedule up to 4 jobs at a time using the licenses. You do not need to set the resource at any queue.

Example 2: Here, your job submitters don’t need to request a counting resource. Jobs are routed based on the size of the request for memory, and the counting resource is inherited from a default. In this example, we are limiting the number of jobs from each group that can use a particular vnode that has a lot of memory. This vnode is called MemNode.

Jobs that request 8GB or more of memory are routed into queue BigMem, and inherit a default counting resource called memcount. All
other jobs are routed into queue *SmallMem*. The routing queue is called *RouteQueue*.

1. Create a custom static integer *memcount* resource that will be tracked at the server and queue:

   a. In `PBS_HOME/server_priv/resourcedef`, add the line:
   ```
   memcount   type=long   flag=q
   ```

   b. Add the resource to the resources: line in `PBS_HOME/sched_priv/sched_config`:
   ```
   resources: “[…], memcount”
   ```

2. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

3. HUP the scheduler:
   ```
   kill -HUP <scheduler PID>
   ```

4. Set limits at *BigMem* and *SmallMem* so that they accept the correct jobs:
   ```
   Qmgr: set queue BigMem resources_min.mem = 8gb
   Qmgr: set queue SmallMem resources_max.mem = 8gb
   ```

5. Set the order of the destinations in the routing queue so that *BigMem* is tested first, so that jobs requesting exactly 8GB go into *BigMem*:
   ```
   Qmgr: set queue RouteQueue route_destinations = “BigMem, SmallMem”
   ```

6. Set the available resource at *BigMem* using `qmgr`. If you want a maximum of 6 jobs from *BigMem* to use *MemNode*:
   ```
   Qmgr: set queue BigMem
   resources_available.memcount = 6
   ```

7. Set the default value for the counting resource at *BigMem*, so that jobs inherit the value:
   ```
   Qmgr: set queue BigMem resources_default.memcount = 1
   ```

8. Associate the vnode with large memory with the *BigMem* queue:
   ```
   Qmgr: set node MemNode queue = BigMem
   ```
The scheduler will only schedule up to 6 jobs from BigMem at a time on the vnode with large memory.

5.11 Using Resources for Topology and Job Placement

Using the topology information in the server’s node_group_key attribute, PBS examines the values of resources at vnodes, and uses those values to create placement sets. Jobs are assigned to placement sets according to their resource requests. Users can specify particular placement sets by requesting the resources that define that particular placement set. For example, if the switch named A25 connects the desired set of vnodes, a user can request the following:

-1 switch=A25

See section 4.8.32 “Placement Sets” on page 237.

5.11.1 Restrictions on Using Resources for Job Placement

Only vnode-level resources can be used to direct jobs to particular vnodes.

5.12 Using Resources to Prioritize Jobs

You can define the formula the scheduler uses to compute job execution priorities. Elements in this formula can be inherited default custom resources. These resources must be job-wide numeric resources, or consumable host-level resources. See section 5.9.3 “Specifying Job Default Resources” on page 338 and section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.
You can make jobs inherit numeric resources according to non-numeric qualities, such as the job owner’s group or whether the job requests a Boolean or string resource. You can do this by either of the following methods:

- Use a hook to identify the jobs you want and alter their resource requests to include the custom resources for the formula. See Chapter 6, "Hooks", on page 461
- Use a routing queue and minimum and maximum resource limits to route jobs to queues where they inherit the default custom resources for the formula. See section 2.2.6.4 “Using Resources to Route Jobs Between Queues” on page 28

For details on how job execution priority is calculated, see section 4.8.16 “Calculating Job Execution Priority” on page 187.

For a complete description of how PBS prioritizes jobs, see section 4.2.5 “Job Prioritization” on page 76.

## 5.13 Using Resources to Restrict Server, Queue Access

You can set resource limits at the server and queues so that jobs must conform to the limits in order to be admitted. This way, you can reject jobs that request more of a resource than the complex or a queue can supply. You can also force jobs into specific queues where they will inherit the desired values for unrequested or custom resources. You can then use these resources to manage jobs, for example by using them in the job sorting formula or to route jobs to particular vnodes.

You set a maximum for each resource at the server using the `resources_max.<resource>` server attribute; there is no `resources_min.<resource>` at the server.

You can set a minimum and a maximum for each resource at each queue using the `resources_min.<resource>` and `resources_max.<resource>` queue attributes.
Chapter 5  

PBS Resources

Job resource requests are compared to resource limits the same way, whether at the server or a queue. For a complete description of how jobs are tested against limits, see section 2.2.6.4.1 “How Queue Limits Are Applied” on page 28.

Job resource requests are compared first to queue admittance limits. If there is no queue admittance limit for a particular resource, the job’s resource request is compared to the server’s admittance limit.

5.13.1 Restrictions on Resources Used for Admittance

Time-based resources, such as walltime, cannot be used as admittance limits. For a list of resources that are compared to admittance limits, see section 2.2.6.4.2 “Resources Used for Routing and Admittance” on page 29. For information on using strings, string arrays, and Booleans for admittance controls, see section 2.2.6.4.4 “Using String, String Array, and Boolean Values for Routing” on page 30.

5.14 Custom Resources

You can define, that is, create, new resources within PBS. The primary use of this feature is to add site-specific resources, such as to manage software application licenses. This section describes how to define and use custom resources.

Once new resources are defined, jobs may request these new resources and the scheduler can schedule on the new resources.

Using this feature, it is possible to schedule resources where the number or amount available is outside of PBS's control.

Custom resources can be made invisible to users or unalterable by users via resource permission flags. See section 5.14.2.5 “Resource Permission Flags” on page 362. A user will not be able to print or list custom resources which have been made either invisible or unalterable.
PBS provides certain custom resources that are designed to reflect resources or properties found on specific systems. Do not create custom resources with the names that PBS uses for these resources. See “Custom Cray Resources” on page 357 of the PBS Professional Reference Guide.

5.14.1 How to Use Custom Resources

Custom resources can be static or dynamic, server-level or host-level, and local or global. They can also be shared or not.

5.14.1.1 Choosing the Resource Category

Use dynamic resources for quantities that PBS does not control, such as externally-managed licenses or scratch space. PBS runs a script or program that queries an external source for the amount of the resource available and returns the value via stdout. Use static resources for things PBS does control, such as licenses managed by PBS. PBS tracks these resources internally.

Use server-level resources for things that are not tied to specific hosts, that is, they can be available to any of a set of hosts. An example of this is a floating license. Use host-level resources for things that are tied to specific hosts, like the scratch space on a machine or node-locked licenses.

5.14.1.1.1 Quick Guide to Configuring a Custom Resource

The following table gives a quick guide to configuring each kind of custom resource:

<table>
<thead>
<tr>
<th>Use for Resource</th>
<th>Link to Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>License: Floating, externally-managed</td>
<td>See section 5.14.7.3.1 “Example of Floating, Externally-managed License” on page 387</td>
</tr>
<tr>
<td>License: Floating, externally-managed with features</td>
<td>See section 5.14.7.3.2 “Example of Floating, Externally-managed License with Features” on page 390</td>
</tr>
</tbody>
</table>
## Table 5-4: Examples of Configuring Custom Resources

<table>
<thead>
<tr>
<th>Use for Resource</th>
<th>Link to Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>License: Floating, PBS-managed</td>
<td>See section 5.14.7.3.3 “Example of Floating License Managed by PBS” on page 392</td>
</tr>
<tr>
<td>License: Node-locked, per-host</td>
<td>See section 5.14.7.4.4 “Example of Per-host Node-locked Licensing” on page 395</td>
</tr>
<tr>
<td>License: Node-locked, per-CPU</td>
<td>See section 5.14.7.4.6 “Example of Per-CPU Node-locked Licensing” on page 400</td>
</tr>
<tr>
<td>License: Node-locked, per-use</td>
<td>See section 5.14.7.4.5 “Example of Per-use Node-locked Licensing” on page 397</td>
</tr>
<tr>
<td>GPU: any GPU</td>
<td>See section 5.14.8.3 “Configuring PBS for Basic GPU Scheduling” on page 403</td>
</tr>
<tr>
<td>GPU: specific GPU</td>
<td>See “Configuring PBS for Advanced GPU Scheduling” on page 404</td>
</tr>
<tr>
<td>Scratch space: local to a host</td>
<td>See section 5.14.6.2 “Dynamic Host-level Scratch Space” on page 384 and section 5.14.6.4 “Static Host-level Scratch Space” on page 385</td>
</tr>
<tr>
<td>Generic dynamic server-level</td>
<td>See section 5.14.4.1.1 “Example of Configuring Dynamic Server-level Resource” on page 371</td>
</tr>
<tr>
<td>Generic static server-level</td>
<td>See section 5.14.4.2.1 “Example of Configuring Static Server-level Resource” on page 373</td>
</tr>
<tr>
<td>Generic dynamic host-level</td>
<td>See section 5.14.5.1.1 “Example of Configuring Dynamic Host-level Resource” on page 376</td>
</tr>
<tr>
<td>Generic static host-level</td>
<td>See section 5.14.5.2.1 “Example of Configuring Static Host-level Resource” on page 377</td>
</tr>
<tr>
<td>Generic shared static host-level</td>
<td>See section 5.14.5.3.5 “Configuring Shared Static Resources” on page 380</td>
</tr>
</tbody>
</table>
5.14.1.2 Dynamic Custom Resources

A dynamic resource is one which is not under the control of PBS, meaning it can change independently of PBS. In order to use a dynamic resource, PBS must run a query to discover the available amount of that resource. Dynamic custom resources can be defined at the server or vnodes.

5.14.1.2.1 Dynamic Server-level Custom Resources

A dynamic server-level custom resource is used to track a resource that is available on the server/scheduler host. You use a dynamic server-level resource to track something that is not under the control of PBS, and changes outside of PBS, for example, floating licenses. At each scheduler cycle, the scheduler runs a script at the scheduler/server host to determine the available amount of that resource. Server-level custom resources are used as job-wide resources.

5.14.1.2.2 Dynamic Host-level Custom Resources

A dynamic host-level custom resource is used to track a resource that is available at the execution host or hosts. You use a dynamic host-level resource for a resource that is not under the control of PBS, and changes outside of PBS, for example, scratch space. At each scheduler cycle, the scheduler queries the MOM for the available amount of the resource. The MOM runs a script which returns the current value of the resource. Host-level dynamic resources are used inside chunks.

5.14.1.3 Static Custom Resources

A static resource is one which is under the control of PBS. Any changes to the value are performed by PBS or by the administrator. Static custom resources are defined ahead of time, at the server, queues or vnodes. Static custom resources can be local or global.

5.14.1.3.1 Global Static Custom Resources

Global static custom resources are defined in PBS_HOME/server_priv/resourcedef. Global static custom resource values at vnode, queue and server are set via qmgr, by setting resources_available.<custom resource name> = <value>. These resources are available at the server, queues, or vnodes.
5.14.1.3.2 Local Static Custom Resources

Local static custom resources are defined in `PBS_HOME/mom_priv/config`, and are available only at the host where they are defined. Note that these resources cannot be set via `qmgr` or viewed via `qstat`. It is not recommended to use local static custom resources.

5.14.1.4 Shared Vnode Resources

A shared vnode resource is managed at one vnode, but available to be used by jobs at others. This allows flexible allocation of the resource. See section 5.14.5.3 “Shared Host-level Resources” on page 378 for information on resources shared across vnodes.

5.14.1.5 Using Custom Resources for Application Licenses

The following table lists application licenses and what kind of custom resource to define for them. See section 5.14.7 “Supplying Application Licenses” on page 386 for specific instructions on configuring each type of license and examples of configuring custom resources for application licenses.

<table>
<thead>
<tr>
<th>Floating or Node-locked</th>
<th>Unit Being Licensed</th>
<th>How License is Managed</th>
<th>Level</th>
<th>Resource Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating (site-wide)</td>
<td>Token</td>
<td>External license manager</td>
<td>Server</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Floating (site-wide)</td>
<td>Token</td>
<td>PBS</td>
<td>Server</td>
<td>Static</td>
</tr>
<tr>
<td>Node-locked</td>
<td>Host</td>
<td>PBS</td>
<td>Host</td>
<td>Static</td>
</tr>
<tr>
<td>Node-locked</td>
<td>CPU</td>
<td>PBS</td>
<td>Host</td>
<td>Static</td>
</tr>
<tr>
<td>Node-locked</td>
<td>Instance of Application</td>
<td>PBS</td>
<td>Host</td>
<td>Static</td>
</tr>
</tbody>
</table>
5.14.1.6 Using Custom Resources for Scratch Space

You can configure a custom resource to report how much scratch space is available on machines. Jobs requiring scratch space can then be scheduled onto machines which have enough. This requires dynamic host-level resources. See section 5.14.6 “Using Scratch Space” on page 383 and section 5.14.5.1 “Dynamic Host-level Resources” on page 374.

5.14.2 Defining New Custom Resources

When you define a new custom resource to be used by jobs, you must do the following:

1. Define the resource to the server in the server’s `resourcedef` file.

2. Put the resource in the “resources:” line in `/PBS_HOME/sched_priv/sched_config`. If the resource is a host-level boolean, you do not need to add it here.

3. Set the resource, either via `qmgr` or by adding it to the correct configuration line.

4. If the resource is dynamic, it must be added to the correct line in the scheduler’s configuration file:
   - If it’s a host-level dynamic resource, it must be added to the `mom_resources` line.
   - If it’s a server-level resource, it must be added to the `server_dyn_res` line.

5. Depending on the type of resource, the server, scheduler and MOMs must be restarted. See section 5.14.3 “Restart Steps for Custom Resources” on page 367.

5.14.2.1 Dynamic Resource Scripts/Programs

You create the script or program that PBS uses to query the external source. The external source can be a license manager or a command, as when you use the `df` command to find the amount of available disk space. If the script is for a server-level dynamic resource, it is placed on the server. If it is for a host-level resource, it is placed on the host(s) where it will be used.
5.14.2.1.1 **Requirements for Scripts/Programs**

- The script must be available to the scheduler, which runs the script
- If you have set up peer scheduling, make sure that the script is available to any scheduler that must run it
- The script must return its output via `stdout`, and the output must be in a single line ending with a newline
- In Windows, if you use Notepad to create the script, be sure to explicitly put a newline at the end of the last line, otherwise none will appear, causing PBS to be unable to properly parse the file

5.14.2.2 **Defining and Setting Static and Dynamic Custom Resources**

The following table lists the differences in defining and setting static and dynamic custom resources at the Server, Queue and host level.

**Table 5-6: Defining and Setting New Custom Resources**

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Server-level</th>
<th>Queue-level</th>
<th>Host-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>Set via <code>qmgr</code></td>
<td>Set via <code>qmgr</code></td>
<td>Set via <code>qmgr</code></td>
</tr>
<tr>
<td>dynamic</td>
<td>Add to <code>server_dyn_res</code> line in PBS_HOME/sched_priv/sched_config</td>
<td>Cannot be used.</td>
<td>Add to MOM config file PBS_HOME/mom_priv/config and mom_resources line in PBS_HOME/sched_priv/sched_config</td>
</tr>
</tbody>
</table>

5.14.2.3 **The `resourcedef` File**

Global custom resources are defined in PBS_HOME/server_priv/resourcedef. The format of each line in PBS_HOME/server_priv/resourcedef is:

```plaintext
<resource name> [type=<resource type>] [flag=<resource flag>]
```
<resource_name> is any string made up of alphanumeric characters, where the first character is alphabetic. Resource names must start with an alphabetic character and can contain alphanumeric, underscore ("_"), and dash ("-") characters: [a-zA-Z][a-zA-Z0-9_]#.

The length of each line in PBS_HOME/server_priv/resourcedef file should not be more than 254 characters. There is no limit to the number of custom resources that can be defined.

<resource type> is the type of the resource value, which can be one of the following keywords:

- **Boolean**
- **long**
- **string**
- **string_array**
- **size**
- **float**

You cannot create a custom resource of type "time" or "duration". For these resources, use "long".

The default for <resource type> is "long".

The format of custom **Boolean**, **size**, **string** or **string_array** resources must be the same as built-in resources.

<resource flag> is zero or more resource accumulation or resource permission flags. See the following sections.

See “Resource Data Types” on page 347 of the PBS Professional Reference Guide for a description of each resource type.

You must restart the server after defining resources in the resourcedef file. See section 5.14.3.1 “Restarting the Server” on page 367.

### 5.14.2.4 Resource Accumulation Flags

When you define a custom resource, you can specify whether it is server-level or host-level, and whether it is consumable or not. This is done by setting resource accumulation flags in the resource definition in PBS_HOME/
server_priv/resourcedef. A consumable resource is tracked, or accumulated, in the server, queue or vnode resources_assigned attribute. The resource accumulation flags determine where the value of resources_assigned.<resource> is incremented.

5.14.2.4.1 Allowable Values for Resource Accumulation Flags

The value of <resource flags> is a concatenation of one or more of the following letters:

(none of h, n, f, or q)
Indicates a queue-level or server-level resource that is not consumable.

h
Indicates a host-level resource. Used alone, means that the resource is not consumable. Required for any resource that will be used inside a select statement. This flag selects hardware. This flag indicates that the resource must be requested inside of a select statement.

Example: for a Boolean resource named "green":

   green type=boolean flag=h

n
The amount is consumable at the host level, for all vnodes assigned to the job. Must be consumable or time-based. Cannot be used with Boolean or string resources. The “h” flag must also be used.

This flag specifies that the resource is accumulated at the vnode level, meaning that the value of resources_assigned.<resource> is incremented at relevant vnodes when a job is allocated this resource.

This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.

f
The amount is consumable at the host level for only the first vnode allocated to the job (vnode with first task.) Must be consumable or time-based. Cannot be used with Boolean or string resources. The “h” flag must also be used.
This flag specifies that the resource is accumulated at the first vnode, meaning that the value of \texttt{resources\_assigned.<resource>} is incremented only at the first vnode when a job is allocated this resource.

\texttt{q}

The amount is consumable at the Queue and Server level. When a job is assigned one unit of a resource with this flag, the \texttt{resources\_assigned.<resource>} attribute at the server and any queue is incremented by one. Must be consumable or time-based.

This flag specifies that the resource is accumulated at the queue and server level, meaning that the value of \texttt{resources\_assigned.<resource>} is incremented at each queue and at the server when a job is allocated this resource.

This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.

5.14.2.4.2 When to Use Accumulation Flags

The following table shows when to use accumulation flags.

\textbf{Table 5-7: When to Use Accumulation Flags}

<table>
<thead>
<tr>
<th>Resource Category</th>
<th>Server</th>
<th>Queue</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static, consumable</td>
<td>flags = q</td>
<td>flags = q</td>
<td>flags = nh or fh</td>
</tr>
<tr>
<td>Static, not consumable</td>
<td>flags = (none of h, n, q or f)</td>
<td>flags = (none of h, n, q or f)</td>
<td>flags = h</td>
</tr>
<tr>
<td>Dynamic</td>
<td>server_dyn_res line in sched_config, flags = (none of h, n, q or f)</td>
<td>(cannot be used)</td>
<td>MOM config and mom_resources line in sched_config, flags = h</td>
</tr>
</tbody>
</table>
5.14.2.4.3 Example of Resource Accumulation Flags

When defining a static consumable host-level resource, such as a node-locked license, you would use the “n” and “h” flags.

When defining a dynamic resource such as a floating license, you would use no flags.

5.14.2.4.4 Resource Accumulation Flag Restrictions and Caveats

- Numeric dynamic resources cannot have the q or n flags set. This would cause these resources to be underused. These resources are tracked automatically by the scheduler.

5.14.2.5 Resource Permission Flags

When you define a custom resource, you can specify whether unprivileged users have permission to view or request the resource, and whether users can qalter a request for that resource. This is done by setting two resource permission flags in the resource definition in $PBS_HOME/server_priv/resourcedef.

5.14.2.5.1 Allowable Values for Resource Permission Flags

- “Invisible”. Users cannot view or request the resource. Users cannot qalter a resource request for this resource.
- “Read only”. Users can view the resource, but cannot request it or qalter a resource request for this resource.
- (neither i nor r) Users can view and request the resource, and qalter a resource request for this resource.

5.14.2.5.2 Effect of Resource Permission Flags

- PBS Operators and Managers can view and request a resource, and qalter a resource request for that resource, regardless of the i and r flags.
- Users, operators and managers cannot submit a job which requests a
restricted resource. Any job requesting a restricted resource will be rejected. If a manager needs to run a job which has a restricted resource with a different value from the default value, the manager must submit the job without requesting the resource, then \texttt{qalter} the resource value.

- While users cannot request these resources, their jobs can inherit default resources from \texttt{resources_default.<resource>} and \texttt{default_chunk.<resource>}.

If a user tries to request a resource or modify a resource request which has a resource permission flag, they will get an error message from the command and the request will be rejected. For example, if they try to \texttt{qalter} a job’s resource request, they will see an error message similar to the following:

```
"qalter: Cannot set attribute, read only or insufficient permission Resource_List.hps 173.mars"
```

Example \texttt{resourcedef} file:

```
W_prio    type=long    flag=i
B_prio    type=long    flag=r
P_prio    type=long    flag=i
```

### 5.14.2.5.3 Resource Permission Flag Restrictions and Caveats

- You can specify only one of the \texttt{i} or \texttt{r} flags per resource. If both are specified, the resource is treated as if only the \texttt{i} flag were specified, and an error message is logged at the default log event class and printed to standard error.

- Resources assigned from the \texttt{default_qsub_arguments} server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag whether that resource was requested by the user or came from \texttt{default_qsub_arguments}.

- The behavior of several command-line interfaces is dependent on resource permission flags. These interfaces are those which view or request resources or modify resource requests:

  - \texttt{pbsnodes}

    Users cannot view restricted host-level custom resources.
Chapter 5  

PBS Resources

- **pbs_rstat**: Users cannot view restricted reservation resources.
- **pbs_rsub**: Users cannot request restricted custom resources for reservations.
- **qalter**: Users cannot alter a restricted resource.
- **qmgr**: Users cannot print or list a restricted resource.
- **qselect**: Users cannot specify restricted resources via -l Resource_List.
- **qsub**: Users cannot request a restricted resource.
- **qstat**: Users cannot view a restricted resource.
**5.14.2.6 Example of Defining Each Type of Custom Resource**

In this example, we add five custom resources: a static and a dynamic host-level resource, a static and a dynamic server-level resource, and a static queue-level resource.

1. The resource must be defined to the server, with appropriate flags set:

   Add resource to `PBS_HOME/server_priv/resourcedef`:
   - `staticserverresource` type=long flag=q
   - `statichostresource` type=long flag=nh
   - `dynamicserverresource` type=long
   - `dynamichostresource` type=long flag=h
   - `staticqueueresource` type=long flag=q

2. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

3. The resource must be added to the scheduler’s list of resources:

   Add resource to the “resources:” line in `PBS_HOME/sched_priv/sched_config`:
   - `resources: ”[…] staticserverresource, statichostresource, dynamicserverresource, dynamichostresource, staticqueueresource”`

   Host-level Boolean resources do not need to be added to the “resources:” line.

4. HUP the scheduler:

   `kill -HUP <scheduler PID>`

5. If the resource is static, use `qmgr` to set it at the host, queue or server level:

   - `Qmgr: set node Host1
     resources_available.statichostresource=1`
   - `Qmgr: set queue Queue1
     resources_available.staticqueueresource=1`
   - `Qmgr: set server`
6. If the resource is dynamic:

a. If it’s a host-level resource, add it to the “mom_resources” line in PBS_HOME/sched_priv/sched_config:

   mom_resources: "dynamichostresource"

b. Add it to the MOM config file PBS_HOME/mom_priv/config:

   UNIX or Windows:
   
   dynamichostresource !path-to-command

   Windows, spaces in path:
   
   dynamichostresource !"path-to-command"

c. If it’s a server-level resource, add it to the “server_dyn_res” line in PBS_HOME/sched_priv/sched_config:

   UNIX:
   
   server_dyn_res: “dynamicserverresource !path-to-command”

   Windows, no spaces in path:
   
   server_dyn_res: ‘dynamicserverresource !path-to-command’

   or:
   
   server_dyn_res: “dynamicserverresource !path-to-command”

   Windows, spaces in path:
   
   server_dyn_res: ‘dynamicserverresource !"path-to-command including spaces”’

resources_available.staticserverresource=1

See “qmgr” on page 178 of the PBS Professional Reference Guide.
5.14.2.7 Custom Resource Values

Allowable values for float and long resources are the same as for built-in resources.

If a string resource value contains spaces or shell metacharacters, enclose the string in quotes, or otherwise escape the space and metacharacters. Be sure to use the correct quotes for your shell and the behavior you want. If the string resource value contains commas, the string must be enclosed in an additional set of quotes so that the command (e.g. qsub, qalter) will parse it correctly. If the string resource value contains quotes, plus signs, equal signs, colons or parentheses, the string resource value must be enclosed in yet another set of additional quotes.

5.14.3 Restart Steps for Custom Resources

In order to have new custom resources recognized by PBS, you must restart or reinitialize any PBS daemon whose files were changed.

5.14.3.1 Restarting the Server

The server must always be restarted for a new custom resource to be recognized.

5.14.3.1.1 Restarting the Server on UNIX/Linux

qterm -t quick
PBS_EXEC/sbin/pbs_server

5.14.3.1.2 Restarting the Server on Windows

Admin> qterm -t quick
Admin> net start pbs_server
5.14.3.1.3 Restarting the Server with Failover Configured

Using `qterm -t quick` leaves the secondary server running; it will become active. If you have configured failover, see section 10.2.7.1 “Stopping Servers” on page 732 and section 10.2.7.2 “Starting Servers” on page 732.

5.14.3.2 Restarting or Reinitializing MOM

On UNIX/Linux, whether the MOM must be restarted or reinitialized depends on which MOM configuration file has been changed.

- If only the Version 1 MOM configuration file was changed, you only need to HUP the MOM.
- If you used the `pbs_mom -s insert` command to add to or change anything in the Version 2 MOM config file, you can HUP the MOM.
- If you used the `pbs_mom -s insert` command to remove anything from the Version 2 MOM config file, you must restart the MOM.

On Windows, you must restart MOM when any MOM configuration file has been changed.

5.14.3.2.1 Reinitializing MOM on UNIX/Linux

1. Use the `ps` command to determine MOM’s process ID. Note that `ps` arguments vary among UNIX systems, thus “`-ef`” may need to be replaced by “`-aux`”.

   \[\text{'ps –ef | grep pbs_mom'}\]

2. HUP MOM using the `kill` command, with MOM’s PID as an argument:

   \[\text{'kill -HUP <MOM PID>}'\]

See “`pbs_mom`” on page 66 of the PBS Professional Reference Guide.
5.14.3.2.2 Restarting MOM on UNIX/Linux

1. Use the `ps` command to determine MOM’s process ID. Note that `ps` arguments vary among UNIX systems, thus “-ef” may need to be replaced by “-aux”.

   ```
   ps -ef | grep pbs_mom
   ```

2. Terminate MOM using the `kill` command, with MOM’s PID as an argument. The syntax will vary depending on your system:

   ```
   kill -INT <MOM PID>
   ```
   or
   ```
   kill -s INT <MOM PID>
   ```

3. Restart MOM, allowing running jobs to continue running through the restart. If your custom resource query script/program takes longer than the default ten seconds, you can change the alarm timeout via the `-a alarm` command line start option:

   ```
   PBS_EXEC/sbin/pbs_mom -p [-a timeout]
   ```

   See “pbs_mom” on page 66 of the PBS Professional Reference Guide.

5.14.3.2.3 Restarting MOM on Windows

If your custom resource query script/program takes longer than the default ten seconds, you can change the alarm timeout via the `-a alarm` command line start option.

   ```
   Admin> net stop pbs_mom
   Admin> net start pbs_mom
   ```

   See section 6.2.3.1 “Startup Options to PBS Services” on page 220 in the PBS Professional Installation & Upgrade Guide.)

5.14.3.3 Restarting the Scheduler

You must restart the scheduler if you added the new custom resource to the `resources: line in PBS_HOME/sched_priv/sched_config`. 
5.14.3.3.1 Reinitializing the Scheduler on UNIX/Linux

   ps –ef | grep pbs_sched  
   kill -HUP <Scheduler PID>

5.14.3.3.2 Restarting the Scheduler on Windows

   Admin> net stop pbs_sched  
   Admin> net start pbs_sched

5.14.4 Configuring Server-level Resources

5.14.4.1 Dynamic Server-level Resources

The availability of a dynamic server-level resource is determined by running a script or program specified in the server_dyn_res line of PBS_HOME/sched_priv/sched_config. The value for resources_available.<resource> is updated at each scheduling cycle with the value returned by the script. This script is run at the host where the scheduler runs, once per scheduling cycle. The script must return the value via stdout in a single line ending with a newline.

The scheduler tracks how much of each numeric dynamic server-level custom resource has been assigned to jobs, and will not overcommit these resources.

The format of a dynamic server-level resource query is a shell escape:

   server_dyn_res: “<resource name> !<path to command>”

In this query,

<resource name> is identical to the name in the resourcedef file.

<path to command> is typically the full path to the script or program that performs the query in order to determine the status and/or availability of the new resource you have added.

The scheduler runs the query and waits for it to finish.

Dynamic server-level resources are usually used for site-wide externally-managed floating licenses.
Server dynamic resource values are never visible in `qstat`, and have no `resources_available.<resource>` representation anywhere in PBS.

### 5.14.4.1.1 Example of Configuring Dynamic Server-level Resource

For a site-wide externally-managed floating license you will need two resources: one to represent the licenses themselves, and one to mark the vnodes on which the application can be run. The first is a server-level dynamic resource and the second is a host-level Boolean, set on the vnodes to send jobs requiring that license to those vnodes.
These are the steps for configuring a dynamic server-level resource for a site-wide externally-managed floating license. If this license could be used on all vnodes, the Boolean resource would not be necessary.

1. Define the resources, for example `floatlicense` and `CanRun`, in the server resource definition file PBS_HOME/server_priv/resourcedef:
   ```
   floatlicense type=long
   CanRun type=boolean flag=h
   ```

2. Write a script, for example `serverdyn.pl`, that returns the available amount of the resource via stdout, and place it on the server’s host. For example, it could be placed in `/usr/local/bin/serverdyn.pl`

3. Restart the server. See section 5.14.3 “Restart Steps for Custom Resources” on page 367.

4. Configure the scheduler to use the script by adding the resource and the path to the script in the `server_dyn_res` line of PBS_HOME/sched_priv/sched_config:
   ```
   UNIX:
   server_dyn_res: "floatlicense !/usr/local/bin/serverdyn.pl"

   Windows:
   server_dyn_res: 'floatlicense !"C:\Program Files\PBS Pro\serverdyn.pl"'
   ```

5. Add the new dynamic resource to the `resources:` line in PBS_HOME/sched_priv/sched_config:
   ```
   resources: "ncpus, mem, arch, [...], floatlicense"
   ```


7. Set the Boolean resource on the vnodes where the floating licenses can be run. Here we designate `vnode1` and `vnode2` as the vnodes that can run the application:
   ```
   Qmgr: active node vnode1,vnode2
   Qmgr: set node resources_available.CanRun=True
   ```
To request this resource, the job’s resource request would include:

```
-l floatlicense=<number of licenses or tokens required>
-l select=1:ncpus=N:CanRun=1
```

### 5.14.4.2 Static Server-level Resources

Static server-level resources are used for resources like floating licenses that PBS will manage. PBS keeps track of the number of available licenses instead of querying an external license manager.

#### 5.14.4.2.1 Example of Configuring Static Server-level Resource

These are the steps for configuring a static server-level resource:

1. Define the resource, for example sitelicense, in the server resource definition file PBS_HOME/server_priv/resourcedef:
   ```
sitelicense type=long flag=q
   ```

2. Restart the server. See section 5.14.3 “Restart Steps for Custom Resources” on page 367.

3. Use the `qmgr` command to set the value of the resource on the server:
   ```
   Qmgr: set server
   resources_available.sitelicense=<number of licenses>
   ```

4. Add the new resource to the `resources:` line in PBS_HOME/sched_priv/sched_config.
   ```
   resources: “ncpus, mem , arch, [...], sitelicense”
   ```

5. Restart the scheduler. See section 5.14.3 “Restart Steps for Custom Resources” on page 367.

### 5.14.5 Configuring Host-level Custom Resources

Host-level custom resources can be static and consumable, static and not consumable, or dynamic. Dynamic host-level resources are used for things like scratch space.
5.14.5.1 Dynamic Host-level Resources

For dynamic host-level custom resources, the scheduler sends a resource query to each MOM to get the current availability for the resource, and uses that value for scheduling. If the MOM returns a value, this value replaces the resources_available value reported by the Server. If the MOM returns no value, the value from the Server is kept. If neither specifies a value, the Scheduler sets the resource value to 0.

The available amount of the resource is determined by running a script or program which returns the amount via stdout. This script or program is specified in the mom_resources line in PBS_HOME/sched_priv/sched_config.

The script is run once per scheduling cycle. For a multi-vnode machine, the script is run for the natural vnode. The resource is shared among the MOM’s vnodes.

The scheduler tracks how much of each numeric dynamic server-level custom resource has been assigned to jobs, and will not overcommit these resources.

The format of a dynamic host-level resource query is a shell escape:

<resource name> !<path to command>

In this query,

<resource name> is identical to the name in the resourcedef file.

<path to command> is typically the full path to the script or program that performs the query in order to determine the status and/or availability of the new resource you have added.

MOM starts the query and waits for output. The default amount of time that MOM waits is 10 seconds; this period can be set via the -a alarm_timeout command line option to pbs_mom. See section 5.14.3.2 “Restarting or Reinitializing MOM” on page 368 and section 6.2.3.1 “Startup Options to PBS Services” on page 220 in the PBS Professional Installation & Upgrade Guide. If the timeout is exceeded and the shell escape process has not finished, a log message, “resource read alarm” is written to the MOM’s log file. The process is given another alarm period to finish and if it does not, another log message is written. The user’s job may not run.
An example of a dynamic host-level resource is scratch space on the execution host.

Host dynamic resource values are never visible in `qstat`, and have no `resources_available.<resource>` representation anywhere in PBS.
5.14.5.1.1 Example of Configuring Dynamic Host-level Resource

In this example, we configure a custom resource to track host-level scratch space. The resource is called `dynscratch`. These are the steps for configuring a dynamic host-level resource:

1. Write a script, for example `hostdyn.pl`, that returns the available amount of the resource via `stdout`. The script must return the value in a single line, ending with a newline. Place the script on each host where it will be used. For example, it could be placed in `/usr/local/bin/hostdyn.pl`.

2. Configure each MOM to use the script by adding the resource and the path to the script in `PBS_HOME/mom_priv/config`:
   
   **UNIX:**
   ```
   dynscratch !/usr/local/bin/hostdyn.pl
   ```
   
   **Windows:**
   ```
   dynscratch !"C:\Program Files\PBS Pro\hostdyn.pl"
   ```

3. Reinitialize the MOMs. See section 5.14.3.2 “Restarting or Reinitializing MOM” on page 368.

4. Define the resource, for example `dynscratch`, in the server resource definition file `PBS_HOME/server_priv/resourcedef`:
   ```
   dynscratch type=size flag=h
   ```

5. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

6. You may optionally specify any limits on that resource via `qmgr`, such as the maximum amount available, or the maximum that a single user can request. For example:

   **Qmgr:** `set server resources_max.scratchspace=1gb`

7. Add the new resource to the `resources:` line in `PBS_HOME/sched_priv/sched_config`:
   ```
   resources: “ncpus, mem, arch, [...], dynscratch”
   ```

8. Add the new resource to the `mom_resources:` line in `PBS_HOME/sched_priv/sched_config`. Create the line if necessary:
mom_resources: “dynscratch”

9. Restart the scheduler. See section 5.14.3.3 “Restarting the Scheduler” on page 369.

To request this resource, the resource request would include

```
-l select=1:ncpus=N:dynscratch=10MB
```

### 5.14.5.2 Static Host-level Resources

Use static host-level resources for things that are managed by PBS and available at the host level, such as GPUs.

#### 5.14.5.2.1 Example of Configuring Static Host-level Resource

In this example, we configure a consumable host-level resource to track GPUs. These are the steps for configuring a static host-level resource:

1. Define the resource, for example `ngpus`, in the server resource definition file `PBS_HOME/server_priv/resourcedef`:
   ```
   ngpus type=long flag=nh
   ```

2. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

3. Use the `qmgr` command to set the value of the resource on the host:
   ```
   Qmgr: set node Host1 ngpus=<number of GPUs>
   ```

4. Add the new resource to the `resources` line in `PBS_HOME/sched_priv/sched_config`:
   ```
   resources: “ncpus, mem, arch, […], ngpus”
   ```

5. Restart the scheduler. See section 5.14.3.3 “Restarting the Scheduler” on page 369.

6. If the GPU host is a multi-vnode machine, you may want to define which GPUs belong in which vnodes. In this case, do the following:
   a. Create a vnode definition file. See section 3.5.3 “Creating Version 2 MOM Configuration Files” on page 60.
   b. Restart the MOM. See section 5.14.3.2 “Restarting or Reinitializing MOM” on page 368.
See section 5.14.7.4.4 “Example of Per-host Node-locked Licensing” on page 395, section 5.14.7.4.5 “Example of Per-use Node-locked Licensing” on page 397, and section 5.14.7.4.6 “Example of Per-CPU Node-locked Licensing” on page 400. These sections give examples of configuring each kind of node-locked license.

### 5.14.5.3 Shared Host-level Resources

Two or more vnodes can share the use of a resource. The resource is managed at one vnode, but available for use at other vnodes. The MOM manages the sharing of the resource, allocating only the available amount to jobs. For example, if you want jobs at two separate vnodes to be able to use the same 4GB of memory, you can make the memory be a shared resource. This way, if a job at one vnode uses all 4GB, no other jobs can use it, but if one job at one vnode uses 2GB, other jobs at either vnode can use up to 2GB.

#### 5.14.5.3.1 Shared Resource Terminology

- **Borrowing vnode**
  - The vnode where a shared vnode resource is available, but not managed.

- **Indirect resource**
  - A shared vnode resource at vnode(s) where the resource is not defined, but which share the resource.

- **Managing vnode**
  - The vnode where a shared vnode resource is defined, and which manages the resource.

- **Shared resource**
  - A vnode resource defined at managed at one vnode, but available for use at others.

#### 5.14.5.3.2 Configuring Shared Host-level Resources

The resource to be shared is defined as usual at one vnode. This is the managing vnode for that resource. For example, to make memory be managed at Vnode1:

```
Qmgr: set node Vnode1 mem = 4gb
```
At vnodes which will use the same resource, the resource is defined to be indirect. For example, to make memory be shared and borrowed at Vnode2:

```
Qmgr: set node Vnode2 mem = @Vnode1
```

### 5.14.5.3.3 Shared Dynamic Host-level Resources

Vnode-level dynamic resources, meaning those listed in the `mom_resources: line in PBS_HOME/sched_priv/sched_config`, are shared resources.

### 5.14.5.3.4 Shared Static Host-level Resources

You can define a static host-level resource to be shared between vnodes. The resource is not shared if you set it to a value at each vnode.
5.14.5.3.5 Configuring Shared Static Resources

1. If the resource to be shared is a custom resource, you must define the resource in PBS_HOME/server_priv/resourcedef before setting its value:
   
   `<resource name> type=<resource type> [flag = <flags>]`

2. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

3. Set the resource on the managing vnode:
   
   To set a static value via qmgr:
   
   ```
   Qmgr: s n managing_vnode
   resources_available.<resource> =<value>
   ```
   
   To set a static value, in MOM Version 2 configuration file:
   
   ```
   managing_vnode:<resource>=<value>
   ```

4. Next, set the resource on the borrowing vnode:
   
   To set a shared resource on a borrowing vnode via qmgr:
   
   ```
   Qmgr: s n borrowing_vnode
   resources_available.<resource>=@managing_vnode
   ```
   
   To set a shared resource in MOM Version 2 configuration file:
   
   ```
   borrowing_vnode:<resource>=@managing_vnode
   ```

5. HUP the MOMs involved; see section 5.14.3.2 “Restarting or Reinitializing MOM” on page 368.

Example 1: To make a static host-level license `dyna-license` on hostA be managed by the natural vnode at hostA and indirect at vnodes hostA0 and hostA1:

```
Qmgr: set node hostA resources_available.dyna-license=4
Qmgr: set node hostA0 resources_available.dyna-license=@hostA
Qmgr: set node hostA1 resources_available.dyna-license=@hostA
```
5.14.5.3.6 Restrictions on Shared Host-level Resources

- If your vnodes represent physical units such as blades, sharing resources like ncpus across vnodes may not make sense.

- If you want to make a resource shared across vnodes, remember that you do not want to schedule jobs on the natural vnode. To avoid this, the following resources should not be explicitly set on the natural vnode:
  - ncpus
  - mem
  - vmem

5.14.5.3.7 Defining Shared and Non-shared Resources for the Altix

On an Altix where you are running pbs_mom.cpuset, you can manage the resources at each vnode. For dynamic host-level resources, the resource is shared across all the vnodes on the machine, and MOM manages the sharing. For static host-level resources, you can either define the resource as shared or not. Shared resources are usually set on the natural vnode and then made indirect at any other vnodes on which you want the resource available. For resources that are not shared, you can set the value at each vnode.
Example 1: To set the resource `string_res` to `round` on the natural vnode of altix03 and make it indirect at altix03[0] and altix03[1]:

```
Qmgr: set node altix03
    resources_available.string_res=round
Qmgr: s n altix03[0]
    resources_available.string_res=@altix03
Qmgr: s n altix03[1]
    resources_available.string_res=@altix03
```

```
pbsnodes -va
altix03
...
    string_res=round
...
altix03[0]
...
    string_res=@altix03
...
altix03[1]
...
    string_res=@altix03
...
```

If you had set the resource `string_res` individually on altix03[0] and altix03[1]:

```
Qmgr: s n altix03[0]
    resources_available.string_res=round
Qmgr: s n altix03[1]
    resources_available.string_res=square
```

```
pbsnodes -va
altix03
...
    <--------string_res not set on natural vnode
...
altix03[0]
...
```
5.14.5.3.8 Shared Resource Restrictions for Multi-vnode Machines

Do not set the values for \texttt{mem}, \texttt{vmem} or \texttt{ncpus} on the natural vnode. If any of these resources has been explicitly set to a non-zero value on the natural vnode, set \texttt{resources\_available.ncpus}, \texttt{resources\_available.mem} and \texttt{resources\_available.vmem} to zero on each natural vnode. See section 3.5.2.3 “Configuring Machines with cpusets” on page 59.

5.14.6 Using Scratch Space

5.14.6.1 Dynamic Server-level (Shared) Scratch Space

If you have scratch space set up so that it’s available to all execution hosts, you can use a server-level custom dynamic resource to track it. The following are the steps for configuring a dynamic server-level resource called \texttt{globalscratch} to track globally available scratch space:

1. Define the resource in the server resource definition file \texttt{PBS\_HOME/server\_priv/resourcedef}:
   \begin{verbatim}
   globalscratch  type=long
   \end{verbatim}

2. Write a script, for example \texttt{serverdynscratch.pl}, that returns the available amount of the resource via \texttt{stdout}, and place it on the
server’s host. For example, it could be placed in /usr/local/bin/serverdynscratch.pl

3. Restart the server. See section 5.14.3.1 “Restarting the Server” on page 367.

4. Configure the scheduler to use the script by adding the resource and the path to the script in the server_dyn_res line of PBS_HOME/sched_priv/sched_config:

   UNIX:
   
   server_dyn_res: “globalscratch !/usr/local/bin/serverdynscratch.pl”

   Windows:
   
   server_dyn_res: ‘globalscratch !”C:\Program Files\PBS Pro\serverdynscratch.pl”’

5. Add the new dynamic resource to the resources: line in PBS_HOME/sched_priv/sched_config:

   resources: “ncpus, mem, arch, […], globalscratch”

6. Restart the scheduler. See section 5.14.3.3 “Restarting the Scheduler” on page 369.

To request this resource, the job’s resource request would include:

-1 globalscratch=<space required>

### 5.14.6.2 Dynamic Host-level Scratch Space

Say you have jobs that require a large amount of scratch disk space during their execution. To ensure that sufficient space is available during job startup, create a custom dynamic resource so that jobs can request scratch space. To create this resource, take the steps outlined in section 5.14.5.1.1 “Example of Configuring Dynamic Host-level Resource” on page 376.

### 5.14.6.3 Static Server-level Scratch Space

If you want to prevent jobs from stepping on each others’ scratch space, you can define additional vnodes that are used only to allocate scratch devices, with one vnode per scratch device. Set the sharing attribute on
each scratch vnode to \textit{force\_excl}, so that only one job can request each
scratch device. To set the \texttt{sharing} attribute, follow the rules in section
3.5.2 “Choosing Configuration Method” on page 58. For example, the
scratch devices are /\texttt{scratch1}, /\texttt{scratch2}, /\texttt{scratch3}, etc. On
each scratch device, set resources as follows:

\begin{verbatim}
resources\_available.ncpus = 0
resources\_available.mem = 0
resources\_available.scratch = 1
sharing = force\_excl
\end{verbatim}

Jobs then request one additional chunk to represent the scratch device, for
example:

\texttt{-l 16:ncpus=1+1:scratch=1}

If a job needs to request a specific scratch device, for example /\texttt{scratch2},
that can be done by additionally asking for the vnode explicitly:

\texttt{:vnode=scratch2}

\subsection*{5.14.6.4 Static Host-level Scratch Space}

If the scratch areas are not mounted on all execution hosts, you can specify
which scratch areas are shared among which subsets of vnodes using indi-
rect resources. See section 5.14.5.3 “Shared Host-level Resources” on
page 378.

\subsection*{5.14.6.5 Caveats for Scratch Space and Jobs}

When more than one job uses scratch space, or when a job is suspended,
scratch space usage may not be handled correctly. See section 5.9.5 “Allo-
cation of Dynamic Resources” on page 346.
5.14.7 Supplying Application Licenses

5.14.7.1 Types of Licenses

Application licenses may be managed by PBS or by an external license manager. Application licenses may be floating or node-locked, and they may be per-host, where any number of instances can be running on that host, per-CPU, where one license allows one CPU to be used for that application, or per-run, where one license allows one instance of the application to be running. Each kind of license needs a different form of custom resource.

5.14.7.1.1 Externally-managed Licenses

Whenever an application license is managed by an external license manager, you must create a custom dynamic resource for it. This is because PBS has no control over whether these licenses are checked out, and must query the external license manager for the availability of those licenses. PBS does this by executing the script or program that you specify in the dynamic resource. This script returns the amount via stdout, in a single line ending with a newline.

5.14.7.1.2 PBS-managed Licenses

When an application license is managed by PBS, you can create a custom static resource for it. You set the total number of licenses using qmgr, and PBS will internally keep track of the number of licenses available.

Use static host-level resources for node-locked application licenses managed by PBS, where PBS is in full control of the licenses. These resources are static because PBS tracks them internally, and host-level because they are tracked at the host.

5.14.7.2 License Units and Features

Different licenses use different license units to track whether an application is allowed to run. Some licenses track the number of CPUs an application is allowed to run on. Some licenses use tokens, requiring that a certain number of tokens be available in order to run. Some licenses require a certain number of features to run the application.
When using units, after you have defined the license resource called `license_name` to the server, be sure to set `resources_available.license_name` to the correct number of units.

Before starting you should have answers to the following questions:

- How many units of a feature does the application require?
- How many features are required to execute the application?
- How do I query the license manager to obtain the available licenses of particular features?

With these questions answered you can begin configuring PBS Professional to query the license manager servers for the availability of application licenses. Think of a license manager feature as a resource. Therefore, you should associate a resource with each feature.

### 5.14.7.3 Server-level (Floating) Licenses

#### 5.14.7.3.1 Example of Floating, Externally-managed License

Here is an example of setting up floating licenses that are managed by an external license server.

For this example, we have a 6-host complex, with one CPU per host. The hosts are numbered 1 through 6. On this complex we have one licensed application which uses floating licenses from an external license manager. Furthermore we want to limit use of the application only to specific hosts. The table below shows the application, the number of licenses, the hosts on which the licenses should be used, and a description of the type of license used by the application.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licenses</th>
<th>Hosts</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppF</td>
<td>4</td>
<td>3-6</td>
<td>uses licenses from an externally managed pool</td>
</tr>
</tbody>
</table>

For the floating licenses, we will use two resources. One is a dynamic server resource for the licenses themselves. The other is a Boolean resource used to indicate that the floating license can be used on a given host.
Server Configuration

1. Define the new resource in the Server’s `resourcedef` file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).

   cd $PBS_HOME/server_priv/
   [edit] resourcedef

   Example `resourcedef` file with new resources added:
   
   ```
   AppF type=long
   runsAppF type=boolean flag=h
   ```

2. Restart the Server. See section 5.14.3.1 “Restarting the Server” on page 367.

Host Configuration

3. Set the Boolean resource on the hosts where the floating licenses can be used.

   ```
   Qmgr: active node host3,host4,host5,host6
   Qmgr: set node resources_available.runsAppF = True
   ```
Scheduler Configuration

4. Edit the Scheduler configuration file:

    cd $PBS_HOME/sched_priv/
    [edit] sched_config

5. Append the new resource names to the resources: line:

    resources: “ncpus, mem, arch, host, [...], AppF, run-
    sAppF”

6. Edit the server_dyn_res: line:

    UNIX:
    server_dyn_res: “AppF !/local/flex_AppF”

    Windows:
    server_dyn_res: ‘AppF !”C:\Program Files\PBS
    Pro\flex_AppF’”

7. Restart the Scheduler. See section 5.14.3.3 “Restarting the Scheduler” on page 369.

To request a floating license for AppF and a host on which AppF can run:

    qsub -l AppF=1
    -l select=runsAppF=True
The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well.

```plaintext
host1
host2
host3  
resources_available.runsAppF = True
host4  
resources_available.runsAppF = True
host5  
resources_available.runsAppF = True
host6  
resources_available.runsAppF = True
```

### 5.14.7.3.2 Example of Floating, Externally-managed License with Features

This is an example of a floating license, managed by an external license manager, where the application requires a certain number of features to run. Floating licenses are treated as server-level dynamic resources. The license server is queried by an administrator-created script. This script returns the value via `stdout` in a single line ending with a newline.

The license script runs on the server’s host once per scheduling cycle and queries the number of available licenses/tokens for each configured application.

When submitting a job, the user's script, in addition to requesting CPUs, memory, etc., also requests licenses.

When the scheduler looks at all the enqueued jobs, it evaluates the license request alongside the request for physical resources, and if all the resource requirements can be met the job is run. If the job’s token requirements cannot be met, then it remains queued.

PBS doesn't actually check out the licenses; the application being run inside the job’s session does that. Note that a small number of applications request varying amounts of tokens during a job run.
Our example needs four features to run an application, so we need four custom resources.

1. Write four scripts, one to query the license server for each of your four features. Complexity of the script is entirely site-specific due to the nature of how applications are licensed.

2. Define four non-consumable server-level features in PBS_HOME/server_priv/resourcedef. These features are defined with no flags:

   feature1   type=long
   feature3   type=long
   feature6   type=long
   feature8   type=long


4. Add the feature resources to the resources: line in PBS_HOME/sched_priv/sched_config:

   resources: “ncpus, mem, arch, [...], feature1, feature3, feature6, feature8”

5. Add each feature’s script path to the server_dyn_res: line in PBS_HOME/server_priv/config:

   UNIX:

   server_dyn_res: “feature1 !/path/to/script [args]”
   server_dyn_res: “feature3 !/path/to/script [args]”
   server_dyn_res: “feature6 !/path/to/script [args]”
   server_dyn_res: “feature8 !/path/to/script [args]”
Windows:

    server_dyn_res: ‘feature1 !"C:\Program Files\PBS Pro\script [args]”’
    server_dyn_res: ‘feature3 !"C:\Program Files\PBS Pro\script [args]”’
    server_dyn_res: ‘feature6 !"C:\Program Files\PBS Pro\script [args]”’
    server_dyn_res: ‘feature8 !"C:\Program Files\PBS Pro\script [args]”’

6. Restart the scheduler. See section 5.14.3.3 “Restarting the Scheduler” on page 369.

5.14.7.3.3 Example of Floating License Managed by PBS

Here is an example of configuring custom resources for a floating license that PBS manages. For this you need a server-level static resource to keep track of the number of available licenses. If the application can run only on certain hosts, then you will need a host-level Boolean resource to direct jobs running the application to the correct hosts.

In this example, we have six hosts numbered 1-6, and the application can run on hosts 3, 4, 5 and 6. The resource that will track the licenses is called AppM. The Boolean resource is called RunsAppM.
Server Configuration

1. Define the new resource in the Server’s `resourcedef` file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).

   ```
cd $PBS_HOME/server_priv/
[edit] resourcedef
```

   Example `resourcedef` file with new resources added:

   ```
   AppM type=long flag=q
   runsAppM type=boolean flag=h
   ```

2. Restart the Server. See section 5.14.3.1 “Restarting the Server” on page 367.

3. Set a value for `AppM` at the Server. Here, we’re allowing 8 copies of the application to run at once:

   ```
   Qmgr: set server resources_available.AppM=8
   ```

Host Configuration

4. Set the value of `runsAppM` on the hosts. Each `qmgr` directive is typed on a single line:

   ```
   Qmgr: active node host3,host4,host5,host6
   Qmgr: set node resources_available.runsAppM = True
   ```

Scheduler Configuration

5. Edit the Scheduler configuration file:

   ```
cd $PBS_HOME/sched_priv/
[edit] sched_config
```

6. Append the new resource name to the `resources:` line. Note that it is not necessary to add a host-level Boolean resource to this line.

   ```
   resources: “ncpus, mem, arch, host, [...], AppM, runsAppM”
   ```

7. Restart the Scheduler. See section 5.14.3.3 “Restarting the Scheduler” on page 369.
To request both the application and a host that can run AppM:

```
qsub -l AppM=1
-l select=1:runsAppM=1 <jobscript>
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well. Since unset Boolean resources are the equivalent of `False`, you do not need to explicitly set them to `False` on the other hosts. Unset Boolean resources will not be printed.

```
host1
host2
host3
    resources_available.runsAppM = True
host4
    resources_available.runsAppM = True
host5
    resources_available.runsAppM = True
host5
    resources_available.runsAppM = True
```

### 5.14.7.4 Host-level (Node-locked) Licenses

#### 5.14.7.4.1 Per-host Node-locked Licenses

If you are configuring a custom resource for a per-host node-locked license, where the number of jobs using the license does not matter, use a host-level Boolean resource on the appropriate host. This resource is set to `True`. When users request the license, they can use the following requests:

For a two-CPU job on a single vnode:

```
-l select=1:ncpus=2:license=1
```

For a multi-vnode job:

```
-l select=2:ncpus=2:license=1
-l place=scatter
```
Users can also use “license=True”, but this way they do not have to change their scripts.

### 5.14.7.4.2 Per-CPU Node-locked Licenses

If you are configuring a custom resource for a per-CPU node-locked license, use a host-level consumable resource on the appropriate vnode. This resource is set to the maximum number of CPUs you want used on that vnode. Then when users request the license, they will use the following request:

For a two-CPU, two-license job:

```
-l select=1:ncpus=2:license=2
```

### 5.14.7.4.3 Per-use Node-locked License

If you are configuring a custom resource for a per-use node-locked license, use a host-level consumable resource on the appropriate host. This resource is set to the maximum number of instances of the application allowed on that host. Then when users request the license, they will use:

For a two-CPU job on a single host:

```
-l select=1:ncpus=2:license=1
```

For a multi-vnode job where each chunk needs two CPUs:

```
-l select=2:ncpus=2:license=1
-l place=scatter
```

### 5.14.7.4.4 Example of Per-host Node-locked Licensing

Here is an example of setting up node-locked licenses where one license is required per host, regardless of the number of jobs on that host.

For this example, we have a 6-host complex, with one CPU per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-host node-locked licenses. We want to limit use of the
application only to specific hosts. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licenses</th>
<th>Hosts</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppA</td>
<td>1</td>
<td>1-4</td>
<td>uses a local node-locked application license</td>
</tr>
</tbody>
</table>

For the per-host node-locked license, we will use a Boolean host-level resource called `resources_available.runsAppA`. This will be set to `True` on any hosts that should have the license, and will default to `False` on all others. The resource is not consumable so that more than one job can request the license at a time.

Server Configuration

1. Define the new resource in the Server’s `resourcedef` file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).
   
   ```
   cd $PBS_HOME/server_priv/
   [edit] resourcedef
   Example resourcedef file with new resources added:
   runsAppA type=boolean flag=h
   AppA  type=long flag=h
   ```

2. Restart the Server. See section 5.14.3.1 “Restarting the Server” on page 367.

Host Configuration

3. Set the value of `runsAppA` on the hosts. Each `qmgr` directive is typed on a single line:

   ```
   Qmgr: active node host1,host2,host3,host4
   Qmgr: set node resources_available.runsAppA = True
   ```
Scheduler Configuration

4. Edit the Scheduler configuration file.

   ```bash
cd $PBS_HOME/sched_priv/
[edit] sched_config
   ```

5. Append the new resource name to the “resources:” line. Note that it is not necessary to add the host-level Boolean resource to this line.

   ```
resources: “ncpus, mem, arch, [...], AppA, runsAppA”
```

6. Restart the Scheduler. See section 5.14.3.3 “Restarting the Scheduler” on page 369.

To request a host with a per-host node-locked license for AppA:

   ```
qusub -l select=1:runsAppA=1 <jobscript>
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well. Since unset Boolean resources are the equivalent of `False`, you do not need to explicitly set them to `False` on the other hosts. Unset Boolean resources will not be printed.

   ```
host1
   resources_available.runsAppA = True
host2
   resources_available.runsAppA = True
host3
   resources_available.runsAppA = True
host4
   resources_available.runsAppA = True
host5
host6
```

5.14.7.4.5 Example of Per-use Node-locked Licensing

Here is an example of setting up per-use node-locked licenses. Here, while a job is using one of the licenses, it is not available to any other job.
For this example, we have a 6-host complex, with 4 CPUs per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-use node-locked licenses. We want to limit use of the application only to specific hosts. The licensed hosts can run two instances each of the application. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licenses</th>
<th>Hosts</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppB</td>
<td>2</td>
<td>1-2</td>
<td>Uses a local node-locked application license</td>
</tr>
</tbody>
</table>

For the node-locked license, we will use one static host-level resource called `resources_available.AppB`. This will be set to 2 on any hosts that should have the license, and to 0 on all others. The “nh” flag combination means that it is host-level and it is consumable, so that if a host has 2 licenses, only two jobs can use those licenses on that host at a time.

Server Configuration

1. Define the new resource in the Server’s `resourcedef` file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).

   ```
cd $PBS_HOME/server_priv/
[edit] resourcedef

Example `resourcedef` file with new resources added:

AppB type=long flag=nh
```

2. Restart the Server. See “Restarting the Server” on page 367.

Host Configuration

3. Set the value of `AppB` on the hosts to the maximum number of instances allowed. Each `qmgr` directive is typed on a single line:

   ```
Qmgr: active node host1,host2
Qmgr: set node resources_available.AppB = 2
Qmgr: active node host3,host4,host5,host6
Qmgr: set node resources_available.AppB = 0
```
Scheduler Configuration

4. Edit the Scheduler configuration file.

   \texttt{cd \$PBS\_HOME/sched\_priv/}
   \[edit\] sched\_config

5. Append the new resource name to the \texttt{resources:} line:

   \begin{verbatim}
   resources: "ncpus, mem, arch, host, [...], AppB"
   \end{verbatim}

6. Restart the Scheduler. See “Restarting the Scheduler” on page 369.

To request a host with a node-locked license for \texttt{AppB}, where you’ll run one instance of \texttt{AppB} on two CPUs:

   \texttt{qsub -l select=1:ncpus=2:AppB=1}

The example below shows what the host configuration would look like. What is shown is actually truncated output from the \texttt{pbsnodes -a} command. Similar information could be printed via the \texttt{qmgr -c "print node @default"} command as well.

   host1
       resources\_available.AppB = 2
   host2
       resources\_available.AppB = 2
   host3
       resources\_available.AppB = 0
   host4
       resources\_available.AppB = 0
   host5
       resources\_available.AppB = 0
   host6
       resources\_available.AppB = 0
5.14.7.4.6 Example of Per-CPU Node-locked Licensing

Here is an example of setting up per-CPU node-locked licenses. Each license is for one CPU, so a job that runs this application and needs two CPUs must request two licenses. While that job is using those two licenses, they are unavailable to other jobs.

For this example, we have a 6-host complex, with 4 CPUs per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-CPU node-locked licenses. We want to limit use of the application to specific hosts only. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licenses</th>
<th>Hosts</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppC</td>
<td>4</td>
<td>3-4</td>
<td>uses a local node-locked application license</td>
</tr>
</tbody>
</table>

For the node-locked license, we will use one static host-level resource called $resources_available.AppC$. We will provide a license for each CPU on hosts 3 and 4, so this will be set to 4 on any hosts that should have the license, and to 0 on all others. The “nh” flag combination means that it is host-level and it is consumable, so that if a host has 4 licenses, only four CPUs can be used for that application at a time.

Server Configuration

1. Define the new resource in the Server’s $resourcedef$ file. Create a new file if one does not already exist by adding the resource names, type, and flag(s).

   cd $PBS_HOME/server_priv/
   [edit] resourcedef

   Example $resourcedef$ file with new resources added:

   AppC type=long flag=nh

2. Restart the Server. See “Restarting the Server” on page 367.
Host Configuration

3. Set the value of AppC on the hosts. Each `qmgr` directive is typed on a single line:

```bash
Qmgr: active node host3,host4
Qmgr: set node resources_available.AppC = 4
Qmgr: active node host1,host2,host5,host6
Qmgr: set node resources_available.AppC = 0
```

Scheduler Configuration

4. Edit the Scheduler configuration file:

```bash
cd $PBS_HOME/sched_priv/
[edit] sched_config
```

5. Append the new resource name to the `resources:` line:

```bash
resources: "ncpus, mem, arch, host, [...], AppC"
```

6. Restart the Scheduler. See “Restarting the Scheduler” on page 369.

To request a host with a node-locked license for AppC, where you’ll run a job using two CPUs:

```bash
qsub -l select=1:ncpus=2:AppC=2
```
The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well.

```plaintext
host1
  resources_available.AppC = 0
host2
  resources_available.AppC = 0
host3
  resources_available.AppC = 0
host4
  resources_available.AppC = 4
host5
  resources_available.AppC = 0
host6
  resources_available.AppC = 0
```

### 5.14.8 Using GPUs

You can configure PBS to support GPU scheduling. We describe how to configure both basic and advanced GPU scheduling. Basic GPU scheduling will meet the needs of most job submitters; it allows a job to request the number of GPUs it needs, as long as the job requests exclusive use of each node containing the GPUs. Advanced GPU scheduling allows jobs to request specific GPUs.

PBS Professional allocates GPUs, but does not bind jobs to any particular GPU; the application itself, or the CUDA library, is responsible for the actual binding.

#### 5.14.8.1 Basic GPU Scheduling

Basic scheduling consists of prioritizing jobs based on site policies, controlling access to nodes with GPUs, ensuring that GPUs are not over-subscribed, and tracking use of GPUs in accounting logs.
Configuring PBS to perform basic scheduling of GPUs is relatively simple, and only requires defining and configuring a single custom resource to represent the number of GPUs on each node.

This method allows jobs to request unspecified GPUs. Jobs should request exclusive use of the node to prevent other jobs being scheduled on their GPUs.

### 5.14.8.2 Advanced GPU Scheduling

Advanced scheduling allows a job to separately allocate (request and/or identify) each individual GPU on a node.

In this case, both PBS and the applications themselves must support individually allocating the GPUs on a node. Advanced scheduling requires defining a PBS vnode for each GPU.

This capability is useful for sharing a single multi-GPU node among multiple jobs, where each job requires exclusive use of its GPUs.

### 5.14.8.3 Configuring PBS for Basic GPU Scheduling

You configure a single custom consumable resource to represent all GPU devices on an execution host. Create a host-level global consumable custom resource to represent GPUs. We recommend that the custom GPU resource is named `ngpus`. Set the value for this resource at each vnode to the number of GPUs on the vnode.

The `ngpus` resource is used exactly the way you use the `ncpus` resource.
5.14.8.3.1 Example of Configuring PBS for Basic GPU Scheduling

In this example, there are two execution hosts, HostA and HostB, and each execution host has 4 GPU devices.

1. Stop the server and scheduler. On the server's host, type:

   `/etc/init.d/pbs stop`

2. Edit `PBS_HOME/server_priv/resourcedef`, and add the following line:

   `ngpus type=long flag=nh`

3. Edit `PBS_HOME/sched_priv/sched_config` to add `ngpus` to the list of scheduling resources:

   `resources: "ncpus, mem, arch, host, vnode, ngpus"`

4. Restart the server and scheduler. On the server's host, type:

   `/etc/init.d/pbs start`

5. Add the number of GPU devices available to each execution host in the cluster via `qmgr`:

   `Qmgr: set node HostA resources_available.ngpus=4`
   `Qmgr: set node HostB resources_available.ngpus=4`

5.14.8.4 Configuring PBS for Advanced GPU Scheduling

You configure each GPU device in its own vnode, and each GPU vnode has a resource to contain the device number of its GPU.

Create and set two custom resources:

- Create a host-level global consumable resource to represent the GPUs on a vnode. We recommend that this resource is called `ngpus`.

  Set `ngpus` on each node to the number of GPUs on that node.

- Create a host-level global non-consumable resource containing the GPU device number, which serves to tie the individual GPU to the vnode. We recommend that this resource is called `gpu_id`.

  Set `gpu_id` for each GPU to the device number of that GPU.
5.14.8.4.1 Example of Configuring PBS for Advanced GPU Scheduling

In this example, there is one execution host, HostA, that has two vnodes, HostA[0] and HostA[1], as well as the natural vnode. HostA has 4 CPUs, 2 GPUs, and 16 GB of memory.

1. Stop the server and scheduler. On the server's host, type:

   `/etc/init.d/pbs stop`

2. Edit `PBS_HOME/server_priv/resourcedef` to add the new custom resources:

   - `ngpus` type=long flag=nh
   - `gpu_id` type=string flag=h

3. Edit `PBS_HOME/sched_priv/sched_config` to add `ngpus` and `gpu_id` to the list of scheduling resources:

   `resources: "ncpus, mem, arch, host, vnode, ngpus, gpu_id"`

4. Restart the server and scheduler. On the server’s host, type:

   `/etc/init.d/pbs start`

5. Create a vnode configuration file for each execution host where GPUs are present. The script for HostA is named `hostA_vnodes`, and is shown here:

   ```
   $configversion 2
   hostA: resources_available.ncpus = 0
   hostA: resources_available.mem = 0
   hostA[0]: resources_available.ncpus = 2
   hostA[0]: resources_available.mem = 8gb
   hostA[0]: resources_available.ngpus = 1
   hostA[0]: resources_available.gpu_id = gpu0
   hostA[0]: sharing = default_excl
   hostA[1]: resources_available.ncpus = 2
   hostA[1]: resources_available.mem = 8gb
   hostA[1]: resources_available.ngpus = 1
   ```
```plaintext
hostA[1] : resources_available.gpu_id = gpu1
hostA[1]: sharing = default_excl

6. Add vnode configuration information in the following manner, for each node with GPUs:

```makefile
PBS_EXEC/sbin/pbs_mom -s insert HostA_vnodes HostA_vnodes
```

7. Signal each MOM to re-read its configuration files:

```bash
kill -HUP <pbs_mom PID>
```

### 5.14.9 Using FPGAs

You can configure a custom resource that allows PBS to track the usage of FPGAs. The FPGAs are detected outside of PBS at boot time. There are two basic methods for automatic configuration of the FPGA resource:

- Create a global static host-level resource called `nfpgas`. Create a boot-up script in init.d that detects the presence of the FPGAs, and sets the value of the `nfpgas` resource.
- Create a global dynamic host-level resource called `nfpgas`. This resource calls a script to detect the presence of FPGAs.

We recommend the static resource, because FPGAs are static, and there is a performance penalty for a dynamic resource.

### 5.14.10 Custom Resource Caveats

- Because some custom resources are external to PBS, they are not completely under the control of PBS. Therefore it is possible for PBS to query and find a resource available, schedule a job to run and use that resource, only to have an outside entity take that resource before the job is able to use it. For example, say you had an external resource of “scratch space” and your local query script simply checked to see how much disk space was free. It would be possible for a job to be started on a host with the requested space, but for another application to use the free space before the job did.
- If a resource is not put in the scheduler’s `resources:` line, when jobs
request the resource, that request will be ignored. If the resource is ignored, it cannot be used to accept or reject jobs at submission time. For example, if you create a string resource String1 on the server, and set it to foo, a job requesting “-l String1=bar” will be accepted. The only exception is host-level Boolean resources, which are considered when scheduling, whether or not they are in the scheduler’s resources: line.

- Do not create resources with the same names or prefixes that PBS uses when creating custom resources for specific systems. See “Custom Cray Resources” on page 357 of the PBS Professional Reference Guide.

5.14.11 Deleting Custom Resources

Removing any custom resource definition should be done with care. It is important to delete a custom resource completely and in the correct order. These steps are described below.

5.14.11.1 Alter Any Jobs Requesting Custom Resource

If you delete a resource definition from PBS_HOME/server_priv/resourcedef and restart the server, all jobs requesting that resource will be purged from the server when it is restarted. To avoid losing jobs requesting a deleted custom resource, use the qalter command on those jobs before restarting the server.

5.14.11.2 Remove References to Custom Resource

**Before** you delete a custom resource, you must remove all references to that resource, including where it is used in hooks, the scheduling formula, queue and server settings such as resources_available, etc. Any attributes containing the custom resource must be unset for that resource.
5.14.11.3 Steps in Deleting Custom Resource

To remove custom resources:

1. Remove all references to the resource
2. Make sure that the pbs_server daemon is running
3. Set scheduling to False
4. For each custom resource to be removed, use qmgr to unset that resource at the server, queue, or node level:
   
   Qmgr: unset server <resource name>
   Qmgr: unset queue <resource name>
   Qmgr: unset node <resource name>

5. Quit qmgr
6. Edit the PBS_HOME/server_priv/resourcedef file to remove the unwanted resources
7. Edit the resources: line in PBS_HOME/sched_priv/sched_config to remove the unwanted resource name
   - If the resource is a server dynamic resource, remove the resource name from the server_dyn_res: line
   - If the resource is a MOM dynamic resource, remove the resource from the mom_resources: line
8. For each MOM whose Version 2 configuration file contains references to the resource, use the pbs_mom -s insert command to update
the Version 2 configuration file. See section 3.5.3 “Creating Version 2 MOM Configuration Files” on page 60.

9. If the resource is a local dynamic resource, defined in the MOM Version 1 configuration file:

For each host where the unwanted resource is defined, edit PBS_HOME/mom_priv/config and remove the resource entry line.

10. Restart the pbs_server daemon; see section 5.14.3.1 “Restarting the Server” on page 367

11. HUP each MOM; see section 5.14.3.2 “Restarting or Reinitializing MOM” on page 368

12. Set scheduling to True

5.15 Managing Resource Usage

You can manage resource usage from different directions:

• You can manage resource usage by users and groups, and the number of jobs, at the server and queue level. See section 5.15.1 “Managing Resource Usage By Users at Server & Queues” on page 410.
  - You can manage the total amount of each resource that is used by users or groups, at the server or queue level. For example, you can manage how much memory is being used by jobs in queue QueueA.
  - You can manage the number of jobs being run by users or groups, at the server or queue level. For example, you can limit the number of jobs enqueued in queue QueueA by any one group to 30, and by any single user to 5.

• You can specify how much of each resource any job is allowed to use, at the server and queue level. See section 5.15.3 “Placing Resource Limits on Jobs” on page 436 and section 5.13 “Using Resources to Restrict Server, Queue Access” on page 351.

• You can set default limits for usage for each resource, at the server or queue level, so that jobs that do not request a given resource inherit that default, and are limited to the inherited amount. For example, you can
specify that any job entering queue QueueA not specifying walltime is limited to using 4MB of memory. See section 5.9.3 “Specifying Job Default Resources” on page 338.

- You can set limits on the number of jobs that can be run at each vnode by users, by groups, or overall. See section 5.15.2 “Limiting Number of Jobs at Vnode” on page 435.

### 5.15.1 Managing Resource Usage By Users at Server & Queues

You can set separate limits for resource usage by individual users, individual groups, generic users, generic groups, and the total used by all users. You can limit the amount of resources used, and the number of running and queued jobs. These limits can be defined separately for each queue and for the server. You define the limits by setting server and queue limit attributes.

There are **two incompatible sets of server and queue limit attributes** used in limiting resource usage. The first set existed in PBS Professional before Version 10.1, and we call them the **old limit attributes**. The old limit attributes are discussed in section 5.15.1.14 “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1” on page 432. The set introduced in Version 10.1 is called simply the limit attributes, and they are discussed here.

You can use either the limit attributes or the old limit attributes for the server and queues, but not both. See section 5.15.1.12.6 “Do Not Mix Old And New Limits” on page 430.

There is a set of limit attributes for vnodes which existed before Version 10.1 and can be used with either the limit attributes or the old limit attributes. These are discussed in section 5.15.2 “Limiting Number of Jobs at Vnode” on page 435.

The server and queues each have per-job limit attributes which operate independently of the limits discussed in this section. The **resources_min.<resource>** and **resources_max.<resource>** server and queue attributes are limits on what each individual job may use. See section 5.13 “Using Resources to Restrict Server, Queue Access” on page 351 and section 5.15.3 “Placing Resource Limits on Jobs” on page 436.
5.15.1.1 Examples of Managing Resource Usage at Server and Queues

You can limit resource usage and job count for specific users and groups:

- UserA can use no more than 6 CPUs, and UserB can use no more than 4 CPUs, at one time anywhere in the PBS complex.
- The crashest group can use no more than 16 CPUs at one time anywhere in the PBS complex.
- UserC accidentally submitted 200,000 jobs last week. UserC can now have no more than 25 jobs enqueued at one time.
- All jobs request the server-level custom resource `nodehours`, which is used for allocation. UserA cannot use more than 40 `nodehours` in the PBS complex. Once UserA reaches the `nodehours` limit, then all queued jobs owned by UserA are not eligible for execution.
- You wish to allow UserD to use 12 CPUs but limit all other users to 4 CPUs.

You can limit the number of jobs a particular user or group runs in a particular queue:

- UserE can use no more than 2 CPUs at one time at Queue1, and 6 CPUs at one time at Queue2.
- You wish to limit UserF to 10 running jobs in queue Queue3, but allow all other users unlimited jobs running in the same queue.
- UserG is a member of Group1. You have a complex-wide limit of 5 running jobs for UserG. You have a limit at Queue1 of 10 running jobs for Group1. This way, up to 10 of the running jobs in Queue1 can belong to Group1, and 5 of these can belong to UserG.
- UserH is a member of Group1. You have a complex-wide limit of 5 running jobs for UserH. You have a limit at Queue1 of 10 running jobs for any group in Queue1. This way, no group in Queue1 can run more than 10 jobs total at one time, and 5 of these can belong to UserH.
- UserJ is a member of Group1. You have a complex-wide limit of 10 running jobs for UserJ. You also have a limit at Queue1 of 5 running jobs for Group1. This means that there may be up to 5 running jobs owned by users belonging to Group1 in Queue1, and up to 5 of these can be owned by UserJ. UserJ can also have another 5 running jobs
owned by Group1 in any other queue, or owned by a different group in Queue1.

You can ensure fairness in the use of resources:

- You have multiple departments which have shared the purchase of a large Altix. Each department would like to ensure fairness in the use of the Altix, by setting limits on individual users and groups.

- You have multiple departments, each of which purchases its own machines. Each department would like to limit the use of its machines so that all departmental users have specific limits. In addition, each department would like to allow non-departmental users to use its machines when they are under-utilized, while giving its own users priority on its machines. A non-departmental user can run jobs on a departmental machine, as long as no departmental users’ jobs are waiting to run.

5.15.1.2 Terminology

**Limit**

The maximum amount of a resource that can be consumed at any time by running jobs or allocated to queued jobs, or the maximum number of jobs that can be running, or the maximum number of jobs that can be queued.

**Limit for all users**

Limit for everyone submitting jobs. In the context of server limits, this is the limit for users submitting jobs to the PBS complex. In the context of queue limits, this is the limit for everyone submitting jobs to the queue. A limit for all users is applied to the total usage by all users at the specified location. A separate limit for all users can be specified at the server and each queue.

**Generic user limit**

Applies separately to users at the server or a queue. The limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.
Generic group limit
Applies separately to groups at the server or a queue. The limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Individual user limit
Applies separately to users at the server or a queue. Limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.

Individual group limit
Applies separately to groups at the server or a queue. Limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

User limit
A limit placed on one or more users, whether generic or individual.

Group limit
This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

Queued jobs
In a queue, queued jobs are the jobs that are waiting in that queue and the jobs that are running in that queue.

Note the very important difference between the term all users and the term generic users. A limit defined for all users is the total limit for usage of that resource by everyone, whereas a limit defined for generic users is the limit set for any single generic user.
Example 1: Difference between all users and generic users

Given the following:
- The server limit for all users for running jobs is 100
- The server limit for generic users is 10
- No individual limits have been set

This means:
- Generic users (any single user) can run no more than 10 jobs
- No more than 100 jobs can be running at any time

All users can mean all users at the server or at the queue in question. Generic users can mean generic users at the server or at the queue in question.

5.15.1.3 Hard and Soft Limits

Hard limits are limits which cannot be exceeded. Soft limits are limits which mark the point where a user or group is using “extra, but acceptable” amounts of a resource. When this happens, the jobs belonging to that user or group are eligible for preemption. See section 4.8.33 “Using Preemption” on page 258. Soft limits are discussed in section 4.8.33.4.1 “The Soft Limits Preemption Level” on page 262.

5.15.1.4 Scope of Limits at Server and Queues

Each of the limits described above can be set separately at the server and at each queue. Each limit’s scope is the PBS object where it is set. The individual and generic user and group limits that are set within one scope interact with each other only within that scope. For example, a limit set at one queue has no effect at another queue.

The scope of limits set at the server encompasses queues, so that the minimum, more restrictive limit of the two is applied. For precedence within a server or queue, see section 5.15.1.6 “Precedence of Limits at Server and Queues” on page 418.
5.15.1.5 Ways To Limit Resource Usage at Server and Queues

You can create a complete set of limits at the server, and you can create another complete set of limits at each queue. You can set hard and soft limits. See section 4.8.33.4.1 “The Soft Limits Preemption Level” on page 262. You can limit resource usage at the server and the queue level for the following:

- Running jobs
  - Number of running jobs
  - Number of running jobs (soft limit)
  - Amount of each resource allocated for running jobs
  - Amount of each resource allocated for running jobs (soft limit)

- Queued jobs (this means both jobs that are waiting to run and jobs that are running from that queue)
  - Number of queued jobs (includes jobs running from that queue)
  - Amount of each resource allocated for queued jobs (includes amount of each resource allocated to jobs running from that queue)

These limits can be applied to the following:

- The total usage at the server
- The total usage at each queue
- Amount used by a single user
  - Generic users
  - Individual users
- Amount used by a single group
  - Generic groups
  - Individual groups

5.15.1.5.1 Limits at Queues

Routing queues do not run jobs, so you cannot set a limit for the number of running jobs, or the amount of resources being used by running jobs, at a routing queue.
When queued jobs are counted for a queue, the number includes the jobs that are running from that queue. Jobs queued at a queue are counted the same whether they were submitted to that queue via the `qsub` command or its equivalent API, moved to that queue via the `qmove` command or its equivalent API, or routed to that queue from another queue.

When resources allocated to queued jobs are summed at a queue, the number includes the resources allocated to jobs running from that queue.

**5.15.1.5.2 Generic and Individual Limits**

You can set a generic limit for groups, so that each group must obey the same limit. You can likewise set a generic limit for users so that each user must obey the same limit. Each generic limit can be set separately at the server and at each queue.

You can set a different individual limit for each user, and you can set individual limits for groups. Each user and group can have a different individual limit at the server and at each queue.

You can use a combination of generic and individual user or group limits, at the server and at each queue. Within the scope of the server or a queue, all users or groups except the ones with the individual limits must obey the generic limit, and the individual limits override the generic limits.
Example 1: Generic and individual user limits on running jobs at QueueA and QueueB

At QueueA:
- At QueueA, the generic user limit is 5
- At QueueA, Bob’s individual limit is 8
- Tom has no individual limit set at QueueA; the generic limit applies

At QueueB:
- At QueueB, the generic user limit is 2
- At QueueB, Tom’s individual limit is 1
- Bob has no individual limit at QueueB; the generic limit applies

This means:
- Bob can run 8 jobs at QueueA
- Bob can run 2 jobs at QueueB
- Tom can run 5 jobs at QueueA
- Tom can run 1 job at QueueB

5.15.1.5.3 Limits For All Users

The limit over all users places a cap on the total amount of the resource that can be used within the scope in question (server or queue), regardless of the whether user or group limits have been reached. A user or group at the server or a queue cannot use any more of a resource for which the limit for all users has been reached, even if that user or group limit has not been reached.
Chapter 5

PBS Resources

Example 1: Limit for all users at server

Given the following:
- Server limit on running jobs is 100
- Bob’s user limit is 10 running jobs
- 98 jobs are already running
- Bob is running zero jobs

This means:
- Bob can start only 2 jobs

5.15.1.6 Precedence of Limits at Server and Queues

5.15.1.6.1 Interactions Between Limits Within One Scope

Within the scope of a PBS object (server or queue), there is an order of precedence for limits when more than one applies to a job. The order of precedence for the limits at a queue is the same as the order at the server. The following table shows how limits interact within one scope:

<table>
<thead>
<tr>
<th></th>
<th>Individual User</th>
<th>Generic User</th>
<th>Individual Group</th>
<th>Generic Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual User</td>
<td>Individual user</td>
<td>Individual user</td>
<td>More restrictive</td>
<td>More restrictive</td>
</tr>
<tr>
<td>Generic User</td>
<td>Individual user</td>
<td>Generic user</td>
<td>More restrictive</td>
<td>More restrictive</td>
</tr>
<tr>
<td>Individual Group</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>Individual group</td>
<td>Individual group</td>
</tr>
<tr>
<td>Generic Group</td>
<td>More restrictive</td>
<td>More restrictive</td>
<td>Individual group</td>
<td>Generic group</td>
</tr>
</tbody>
</table>

An individual user limit overrides a generic user limit.
Example 1: Individual user limit overrides generic user limit

Given the following:
- Bob has a limit of 10 running jobs
- The generic limit is 5

This means:
- Bob can run 10 jobs

An individual group limit overrides a generic group limit in the same manner as for users.

If the limits for a user and the user’s group are different, the more restrictive limit applies.

Example 2: More restrictive user or group limit applies

Given the following:
- Tom’s user limit for running jobs is 8
- Tom’s group limit is 7

This means:
- Tom can run only 7 jobs in that group

If a user belongs to more than one group, that user can run jobs up to the lesser of his user limit or the sum of the group limits.

Example 3: User can run jobs in more than one group

Given the following:
- Tom’s user limit is 10 running jobs
- GroupA has a limit of 2 and GroupB has a limit of 4
- Tom belongs to GroupA and GroupB

This means:
- Tom can run 6 jobs, 2 in GroupA and 4 in GroupB

5.15.1.6.2 Interactions Between Queue and Server Limits

If the limits for a queue and the server are different, the more restrictive limit applies.
Example 4: More restrictive queue or server limit applies

Given the following:
- Server limit on running jobs for generic users is \textit{10}
- Queue limit for running jobs from QueueA for generic users is \textit{15}
- Queue limit for running jobs from QueueB for generic users is \textit{5}

This means:
- Generic users at QueueA can run 10 jobs
- Generic users at QueueB can run 5 jobs

Example 5: More restrictive queue or server limit applies

Given the following:
- Bob’s user limit on running jobs, set on the server, is \textit{7}
- Bob’s user limit on running jobs, set on QueueA, is \textit{6}

This means:
- Bob can run 6 jobs from QueueA

5.15.1.7 Resource Usage Limit Attributes for Server and Queues

Each of the following attributes can be set at both the server and each queue:

\texttt{max\_run}

The maximum number of jobs that can be running.

\texttt{max\_run\_soft}

The soft limit on the maximum number of jobs that can be running.

\texttt{max\_run\_res.<resource>}

The maximum amount of the specified resource that can be allocated to running jobs.

\texttt{max\_run\_res\_soft.<resource>}

The soft limit on the amount of the specified resource that can be allocated to running jobs.
max_queued
The maximum number of jobs that can be queued and running. Any job running from a queue is counted as being queued in that queue. At the server level, this includes all jobs in the complex. Queueing a job includes the qsub and qmove commands and the equivalent APIs.

max_queued_res.<resource>
The maximum amount of the specified resource that can be allocated to queued and running jobs. Any job running from a queue is counted as being queued in that queue. At the server level, this includes all jobs in the complex. Queueing a job includes the qsub and qmove commands and the equivalent APIs.

Each attribute above can be used to specify all of the following:

- A limit on the total, summed over all users (at the queue or server)
- A limit for generic users
- Individual limits for specific users
- A limit for generic groups
- Individual limits for specific groups

For example, you can specify the limits for the number of running jobs:

- In the complex:
  - The limit for all users (usage by all users in the entire complex) is 10,000
  - The limit for generic users is 5
  - The limit for Bob is 10
  - The limit for generic groups is 50
  - The limit for group GroupA is 75

- At QueueA:
  - The limit for all users (usage by all users with jobs in QueueA) is
200
- The limit for generic users is 2
- The limit for Bob is 1
- The limit for generic groups is 3
- The limit for group GroupA is 7

- At QueueB:
  - The limit for all users (usage by all users with jobs in QueueB) is 500
  - The limit for generic users is 6
  - The limit for Bob is 8
  - The limit for generic groups is 15
  - The limit for group GroupA is 11

### 5.15.1.8 How to Set Limits at Server and Queues

You can set, add, and remove limits by using the `qmgr` command to set limit attributes.

#### 5.15.1.8.1 Syntax

Format for setting a limit attribute:

```
set server <limit attribute> = "[limit-spec=<limit>], [limit-spec=<limit>],...
```

```
set <queue> <queue name> <limit attribute> = "[limit-spec=<limit>], [limit-spec=<limit>],...
```

Format for adding a limit to an attribute:

```
set server <limit attribute> += "[limit-spec=<limit>], [limit-spec=<limit>],...
```

```
set <queue> <queue name> <limit attribute> += "[limit-spec=<limit>], [limit-spec=<limit>],...
```
Format for removing a limit from an attribute:

\[
\text{set server } <\text{limit attribute}> = \text{"}[\text{limit-spec}], [\text{limit-spec}],\ldots\text{"}
\]

\[
\text{set } <\text{queue}> <\text{queue name}> <\text{limit attribute}> = \text{"}[\text{limit-spec}], [\text{limit-spec}],\ldots\text{"
\]

Alternate format for removing a limit from an attribute; note that the value
of \(<\text{limit}>\) used when removing a limit must match the value of the limit:

\[
\text{set server } <\text{limit attribute}> = \text{"}[\text{limit-spec}=<\text{limit}>], [\text{limit-spec}=<\text{limit}>],\ldots\text{"
\]

\[
\text{set } <\text{queue}> <\text{queue name}> <\text{limit attribute}> = \text{"}[\text{limit-spec}=<\text{limit}>], [\text{limit-spec}=<\text{limit}>],\ldots\text{"
\]

where \text{limit-spec} specifies a user limit, a group limit, or a limit for all users:

**Table 5-9: Specifying Limits**

<table>
<thead>
<tr>
<th>Limit</th>
<th>limit-spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>All users</td>
<td>o:PBS_ALL</td>
</tr>
<tr>
<td>Generic users</td>
<td>u:PBS GENERIC</td>
</tr>
<tr>
<td>An individual user</td>
<td>u:&lt;username&gt;</td>
</tr>
<tr>
<td>Generic groups</td>
<td>g:PBS GENERIC</td>
</tr>
<tr>
<td>An individual group</td>
<td>g:&lt;groupname&gt;</td>
</tr>
</tbody>
</table>

The \text{limit-spec} can contain spaces anywhere except after the colon (":").

If there are comma-separated \text{limit-specs}, the entire string must be enclosed in double quotes.

A username or groupname containing spaces must be enclosed in quotes.

If a username or groupname is quoted using double quotes, and the entire string requires quotes, the outer enclosing quotes must be single quotes. Similarly, if the inner quotes are single quotes, the outer quotes must be double quotes.

\text{PBS_ALL} is a keyword which indicates that this limit applies to all users.

\text{PBS GENERIC} is a keyword which indicates that this limit applies to generic users or groups.
When removing a limit, the limit value does not need to be specified. 

*PBS_ALL* and *PBS_GENERIC* are case-sensitive.

5.15.1.8.2 Examples of Setting Server and Queue Limits

Example 1: To set the maxqueued limit on QueueA to 5 for all users, and to limit user bill to 3:

```
Qmgr: s q QueueA maxqueued = "[o:PBS_ALL=5],
[u:bill =3]"
```

Example 2: On QueueA, set the maximum number of CPUs and the maximum amount of memory that user bill can request in his queued jobs:

```
Qmgr: s q QueueA maxqueued_res.ncpus ="[u:bill=5]", maxqueued_res.mem =
"[u:bill=100mb]"
```

Example 3: To set a limit for a username with a space in it, and to set a limit for generic groups:

```
Qmgr: s q QueueA maxqueued = ’[u:"\PROG\Named User" = 1], [g:PBS_GENERIC=4]’
```

5.15.1.8.3 Examples of Adding Server and Queue Limits

Example 1: To add a limit for the maximum number of jobs that can be queued by all users at QueueA to 10:

```
Qmgr: s q QueueA maxqueued += [o:PBS_ALL=10]
```

Example 2: To add an individual user limit, an individual group limit, and a generic group limit on queued jobs at QueueA:

```
Qmgr: s q QueueA maxqueued += "[u:user1= 5],
[g:GroupMath=5],[g:PBS_GENERIC=2]"
```

Example 3: To add a limit at QueueA on the number of CPUs allocated to queued jobs for an individual user, and a limit at QueueA on the amount of memory allocated to queued jobs for an individual user:

```
Qmgr: s q QueueA maxqueued_res.ncpus +=
[u:tom=5], maxqueued_res.mem += [u:tom=100mb]
```
5.15.1.8.4 Examples of Removing Server and Queue Limits

It is not necessary to specify the value of the limit when removing a limit, but you can specify the value of the limit.

Example 1: To remove the generic user limit at QueueA for queued jobs, use either of the following:

\texttt{Qmgr: set queue QueueA max\_queued \(-= \[u:PBS\_GENERIC\])}

\texttt{Qmgr: set queue QueueA max\_queued \(-= \[u:PBS\_GENERIC=2\])}

Example 2: To remove the limit on queued jobs at QueueA for \textit{Named User}, use either of the following:

\texttt{Qmgr: set queue QueueA max\_queued \(-= \[u:\"PROG\Named User\"]\})}

\texttt{Qmgr: set queue QueueA max\_queued \(-= \[u:\"PROG\Named User\"=1\])}

Example 3: To remove the limit at QueueA on the amount of memory allocated to an individual user, use either of the following:

\texttt{Qmgr: set queue QueueA max\_queued\_res.mem \(-= \[u:tom\])}

\texttt{Qmgr: set queue QueueA max\_queued\_res.mem \(-= \[u:tom=100mb\])}

To remove the limit on the number of CPUs allocated to queued jobs for user bill, use either of the following:

\texttt{Qmgr: set queue QueueA max\_queued\_res.ncpus \(-= \[u:bill\])}

\texttt{Qmgr: set queue QueueA max\_queued\_res.ncpus \(-= \[u:bill=5\])}

Example 4: To remove a generic user limit and an individual user limit, use either of the following:

\texttt{Qmgr: set queue QueueA max\_queued \(-= \"[u:user1], [u:PBS\_GENERIC]\”)}

\texttt{Qmgr: set queue QueueA max\_queued \(-= \"[u:user1=2], [u:PBS\_GENERIC=4]\”)}
5.15.1.9  Who Can Set Limits at Server and Queues

As with other server and queue attributes, only PBS Managers and Operators can set limit attributes.

5.15.1.10  Viewing Server and Queue Limit Attributes

5.15.1.10.1  Printing Server and Queue Limit Attributes

You can use the `qmgr` command to print the commands used to set the limit attributes at the server or queue.

Example 1: To print all the limit attributes for queue QueueA:

```bash
Qmgr: p q QueueA max_queued, max_queued_res
#
# Create queues and set their attributes.
#
#
# Create and define queue workq
#
create queue QueueA
set queue QueueA max_queued = "[o:PBS_ALL=10]"
set queue QueueA max_queued += "[u:PBS GENERIC=2]"
set queue QueueA max_queued += "[u:bill=3]"
set queue QueueA max_queued += "[u:tom=15]"
set queue QueueA max_queued += "[u:user1=3]"
set queue QueueA max_queued += "[u:"\PROG\Named User"=1]"
set queue QueueA max_queued += "[g:PBS GENERIC=2]"
set queue QueueA max_queued += "[g:GroupMath=5]"
set queue QueueA max_queued_res.ncpus = "[u:bill=5]"
set queue QueueA max_queued_res.ncpus += "[u:tom=5]"
set queue QueueA max_queued_res.mem = "[u:bill=100mb]"
set queue QueueA max_queued_res.mem += "[u:tom=100mb]"
```
5.15.1.10.2 Listing Server and Queue Limit Attributes

You can use the `qmgr` command to list the limit attributes for the queue or server.

Example 1: To list the `max_queued` and `max_queued_res` attributes for QueueA:

```
Qmgr: l q QueueA max_queued, max_queued_res
Queue: QueueA
    max_queued = [o:PBS_ALL=10]
    max_queued = [g:PBS_GENERIC=2]
    max_queued = [g:GroupMath=5]
    max_queued = [u:PBS_GENERIC=2]
    max_queued = [u:bill=3]
    max_queued = [u:tom=15]
    max_queued = [u:user1=3]
    max_queued = [u:"\PROG\Named User"=1]
    max_queued_res.ncpus = [u:bill=5]
    max_queued_res.ncpus = [u:tom=5]
    max_queued_res.mem = [u:bill=5]
    max_queued_res.mem = [u:bill=100mb]
    max_queued_res.mem = [u:tom=100mb]
```

5.15.1.10.3 Using the `qstat` Command to View Queue Limit Attributes

You can use the `qstat` command to see the limit attribute settings for the queue or server.
Example 1: To see the settings for the \texttt{max\_queued} and \texttt{max\_queued\_res} limit attributes for QueueA using the \texttt{qstat} command:

\texttt{qstat -Qf QueueA}

Queue: QueueA

\begin{verbatim}
... 
max\_queued = [o:PBS\_ALL=10]
max\_queued = [g:PBS\_GENERIC=2]
max\_queued = [g:GroupMath=5]
max\_queued = [u:PBS\_GENERIC=2]
max\_queued = [u:bill=3]
max\_queued = [u:tom=3]
max\_queued = [u:cs=3]
max\_queued = [u:"\PROG\Named User"=1]
max\_queued\_res.ncpus = [u:bill=5]
max\_queued\_res.ncpus = [u:tom=5]
max\_queued\_res.mem = [u:bill=5]
max\_queued\_res.mem = [u:tom=100mb]
max\_queued\_res.mem = [u:cs=100mb]
\end{verbatim}

5.15.1.11 How Server and Queue Limits Work

\textit{Affected jobs} are jobs submitted by the user or group whose limit has been reached. The following table shows what happens when a given limit is reached:

\begin{table}[h]
\centering
\caption{Actions Performed When Limits Are Reached}
\begin{tabular}{|l|p{10cm}|}
\hline
\textbf{Limit} & \textbf{Action} \\
\hline
Running jobs & No more affected jobs are run at this server or queue until the number of affected running jobs drops below the limit. \\
\hline
\end{tabular}
\end{table}
5.15.1.12 Caveats and Advice for Server and Queue Limits

5.15.1.12.1 Queued Jobs

The term *queued jobs* means, for a queue, all the jobs that are in that queue waiting to run, as well as all the jobs currently running from that queue.

The total number of queued jobs at the server level is the total number of queued and running jobs in the complex. This includes jobs in routing queues.

5.15.1.12.2 Avoiding Overflow

On PBS server platforms for which the native size of a `long` is less than 64 bits, you should refrain from defining a limit on a resource of type `long` whose cumulative sum over all queued jobs would exceed the storage capacity of the resource variable. For example, if each submitted job were to request 100 hours of the `cput` resource, overflow would occur on a 32-bit platform when 5965 jobs (which is \( \frac{2^{31} - 1}{360000} \) seconds) were queued.

---

**Table 5-10: Actions Performed When Limits Are Reached**

<table>
<thead>
<tr>
<th>Limit</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queued jobs</td>
<td>The queue does not accept any more affected jobs until the number of affected queued jobs drops below the limit. Affected jobs submitted directly to the queue are rejected. Affected jobs in a routing queue whose destination is this queue remain in the routing queue.</td>
</tr>
<tr>
<td>Resources for running jobs</td>
<td>The queue does not run any more affected jobs until the limit would not be exceeded if the next affected job were to start.</td>
</tr>
<tr>
<td>Resources for queued jobs</td>
<td>The queue does not accept any more affected jobs until the limit would not be exceeded if the next affected job were to start. Affected jobs submitted directly to the queue are rejected. Affected jobs in a routing queue whose destination is this queue remain in the routing queue.</td>
</tr>
</tbody>
</table>
5.15.12.3 Ensuring That Limits Are Effective

In order for limits to be effective, each job must specify each limited resource. This can be accomplished using defaults; see section 5.9.3 “Specifying Job Default Resources” on page 338. You can also use hooks; see section “Hooks” on page 461.

5.15.12.4 Array Jobs

An array job with N subjobs is considered to consume N times the amount of resources requested when it was submitted. For example, if there is a server limit of 100 queued jobs, no user would be allowed to submit an array job with more than 100 subjobs.

5.15.12.5 Avoiding Job Rejection

Jobs are rejected when users or groups who have reached their limit submit a job in the following circumstances:

- The job is submitted to the execution queue where the limit has been reached
- The job is submitted to the complex, and the server limit has been reached

If you wish to avoid having jobs be rejected, you can set up a routing queue as the default queue. Set the server’s `default_queue` attribute to the name of the routing queue. See section 2.2.6 “Routing Queues” on page 26.

5.15.12.6 Do Not Mix Old And New Limits

The new limit attributes are incompatible with the old limit attributes. See section 5.15.1.14 “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1” on page 432. You cannot mix the use of old and new resource usage limit attributes. This means that:

- If any old limit attribute is set, and you try to set a new limit attribute, you will get error 15141.
- If any new limit attribute is set, and you try to set an old limit attribute, you will get error 15141.

You must unset all of one kind in order to set any of the other kind.
5.15.1.13 Errors and Logging for Server and Queue Limits

5.15.1.13.1 Error When Setting Limit Attributes

Attempting to set a new limit attribute while an old limit attribute is set:

"use new/old qmgr syntax, not both"

"Attribute name <new> not allowed. Older name <old> already set"

Attempting to set an old limit attribute while a new limit attribute is set:

"use new/old qmgr syntax, not both"

"Attribute name <old> not allowed: Newer name <new> already set"

5.15.1.13.2 Logging Events

Whenever a limit attribute is set or modified, the server logs the event, listing which attribute was modified and who modified it.

Whenever a limit is reached, and would be exceeded by a job, the scheduler logs the event, listing the limit attribute and the reason.

5.15.1.13.3 Queued Limit Error Messages

When a limit for queued jobs or resources allocated to queued jobs is reached, the command involved presents a message. This command can be qsub, qmove or qalter.

5.15.1.13.4 Run Limit Error Messages

See “Run Limit Error Messages” on page 500 of the PBS Professional Reference Guide for a list of run limit error messages.
5.15.1.14 Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1

The old server and queue limit attributes discussed here existed in PBS Professional before Version 10.1. The old limit attributes continue to function as they did in PBS Professional 10.0. These attributes are incompatible with the limit attributes introduced in Version 10.1. See section 5.15.1.12.6 “Do Not Mix Old And New Limits” on page 430 and section 5.15.1.13.1 “Error When Setting Limit Attributes” on page 431. These limits are compatible with the limits discussed in section 5.15.2 “Limiting Number of Jobs at Vnode” on page 435.

The following table shows how the old limit attributes are used:

Table 5-11: Resource Usage Limits Existing Before Version 10.1

<table>
<thead>
<tr>
<th>Limit</th>
<th>All Users</th>
<th>Generic Users</th>
<th>Generic Groups</th>
<th>Individual Users</th>
<th>Individual Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of running jobs</td>
<td>max_ru</td>
<td>max_user_run</td>
<td>max_group_run</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>nning</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum number of running jobs (soft limit)</td>
<td>N/A</td>
<td>max_user_run_soft</td>
<td>max_group_run_soft</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum amount of specified resource allocated to running jobs</td>
<td>N/A</td>
<td>max_user_res</td>
<td>max_group_res</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
5.15.1.14.1 Precedence of Old Limits

If an old limit is defined at both the server and queue, the more restrictive limit applies.

5.15.1.14.2 Old Server Limits

For details of these limits, see “Server Attributes” on page 365 of the PBS Professional Reference Guide.

max_running
The maximum number of jobs allowed to be selected for execution at any given time.
max_group_res,
max_group_res_soft
The maximum amount of the specified resource that all members of the same UNIX group may consume simultaneously.

max_group_run,
max_group_run_soft
The maximum number of jobs owned by a UNIX group that are allowed to be running from this server at one time.

max_user_res,
max_user_res_soft
The maximum amount of the specified resource that any single user may consume.

max_user_run,
max_user_run_soft
The maximum number of jobs owned by a single user that are allowed to be running at one time.

5.15.1.14.3 Old Queue Limits

For details of these limits, see “Queue Attributes” on page 407 of the PBS Professional Reference Guide.

max_group_res,
max_group_res_soft
The maximum amount of the specified resource that all members of the same UNIX group may consume simultaneously, in the specified queue.

max_group_run,
max_group_run_soft
The maximum number of jobs owned by a UNIX group that are allowed to be running from this queue at one time

max_queuable
The maximum number of jobs allowed to reside in the queue at any given time. Once this limit is reached, no new jobs will be accepted into the queue.
max_user_res,
max_user_res_soft
The maximum amount of the specified resource that any single user may consume in submitting to this queue.

max_user_run,
max_user_run_soft
The maximum number of jobs owned by a single user that are allowed to be running at one time from this queue.

5.15.2 Limiting Number of Jobs at Vnode

You can set limits at each vnode on the number of jobs that can be run by any user, by any group, or by everyone taken together. You set these limits by specifying values for vnode attributes. The attributes that limit the number of running jobs at a vnode are compatible with both the old limit attributes and the limit attributes introduced in Version 10.1. They are listed here:

max_group_run
The maximum number of jobs owned by any users in a single group that are allowed to be run on this vnode at one time.

Format: integer

Qmgr: set node MyNode max_group_run=8

max_running
The maximum number of jobs allowed to be run on this vnode at any given time.

Format: integer

Qmgr: set node MyNode max_running=22

max_user_run
The maximum number of jobs owned by a single user that are allowed to be run on this vnode at one time.

Format: integer

Qmgr: set node MyNode max_user_run=4
5.15.3 Placing Resource Limits on Jobs

Jobs are assigned limits on the amount of resources they can use. Each limit is set at the amount requested or allocated by default. These limits apply to how much the job can use on each vnode (per-chunk limit) and to how much the whole job can use (job-wide limit). Limits are derived from both requested resources and applied default resources. For information on default resources, see section 5.9.3 “Specifying Job Default Resources” on page 338.

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are the amount of per-chunk resources requested, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are derived in this order from the following:

1. explicitly requested job-wide resources (e.g. -l resource=value)
2. the select specification (e.g. -l select=...)
3. the queue’s resources_default.<resource>
4. the server’s resources_default.<resource>
5. the queue’s resources_max.<resource>
6. the server’s resources_max.<resource>

The server’s default_chunk.<resource> does not affect job-wide limits.

The resources requested for chunks in the select specification are summed, and this sum makes a job-wide limit. Job resource limits from sums of all chunks override those from job-wide defaults and resource requests.

Various limit checks are applied to jobs. If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated.
5.15.3.1 Configuring Per-job Limits at Server and Queue

You can set per-job limits on the amount of each resource that any one job can use. You can set these limits at the server and at each queue. For example, you can specify the following limits:

- Jobs at the server can use no more than 48 hours of CPU time
- Jobs at QueueA can use no more than 12 hours of CPU time
- Jobs at QueueA must request more than 2 hours of CPU time

To set these limits, specify values for the server’s `resources_max.<resource>` attribute and each queue’s `resources_max.<resource>` and `resources_min.<resource>` attributes. The server does not have a `resources_min.<resource>` attribute. To set the maximum at the server, the format is:

```
Qmgr: set server resources_max.<resource> = value
```

To set the maximum and minimum at the queue, the format is:

```
Qmgr: set queue <queue name> resources_max.<resource> = value
Qmgr: set queue <queue name> resources_min.<resource> = value
```

For example, to set the 48 hour CPU time limit:

```
Qmgr: set server resources_max.cput = 48:00:00
```

5.15.3.2 Configuring Per-job Resource Limit Enforcement at Vnodes

For a job, enforcement of resource limits is per-MOM, not per-vnode. So if a job requests 3 chunks, each of which has 1MB of memory, and all chunks are placed on one host, the limit for that job for memory for that MOM is 3MB. Therefore one chunk can be using 2 MB and the other two using 0.5MB and the job can continue to run.

Job resource limits can be enforced for single-vnode jobs, or for multi-vnode jobs that are using LAM or a PBS-aware MPI. See the following table for an overview. Memory limits are handled differently depending on the operating system. See “Job Memory Limit Enforcement on UNIX” on
The `ncpus` limit can be adjusted in several ways. See “Job ncpus Limit Enforcement” on page 442 for a discussion. The following table summarizes how resource limits are enforced at vnodes:

**Table 5-12: Resource Limit Enforcement at Vnodes**

<table>
<thead>
<tr>
<th>Limit</th>
<th>What determines when limit is enforced</th>
<th>Scope of limit</th>
<th>Enforcement method</th>
</tr>
</thead>
<tbody>
<tr>
<td>file size</td>
<td>automatically</td>
<td>per-process</td>
<td><code>setrlimit()</code></td>
</tr>
<tr>
<td>vmem</td>
<td>If job requests or inherits vmem</td>
<td>job-wide</td>
<td>MOM poll</td>
</tr>
<tr>
<td>pvmem</td>
<td>If job requests or inherits pvmem</td>
<td>per-process</td>
<td><code>setrlimit()</code></td>
</tr>
<tr>
<td>pmem</td>
<td>If job requests or inherits pmem</td>
<td>per-process</td>
<td><code>setrlimit()</code></td>
</tr>
<tr>
<td>pcput</td>
<td>If job requests or inherits pcput</td>
<td>per-process</td>
<td><code>setrlimit()</code></td>
</tr>
<tr>
<td>cput</td>
<td>If job requests or inherits cput</td>
<td>job-wide</td>
<td>MOM poll</td>
</tr>
<tr>
<td>wall-time</td>
<td>If job requests or inherits walltime</td>
<td>job-wide</td>
<td>MOM poll</td>
</tr>
<tr>
<td>mem</td>
<td>if <code>$enforce mem</code> in MOM’s config</td>
<td>job-wide</td>
<td>MOM poll</td>
</tr>
<tr>
<td>ncpus</td>
<td>if <code>$enforce cpuaverage</code>, <code>$enforce cpuburst</code>, or both, in MOM’s config. See “Job ncpus Limit Enforcement” on page 442.</td>
<td>job-wide</td>
<td>MOM poll</td>
</tr>
</tbody>
</table>
5.15.3.3  Job Memory Limit Enforcement

You may wish to prevent jobs from swapping memory. To prevent this, you can set limits on the amount of memory a job can use. Then the job must request an amount of memory equal to or smaller than the amount of physical memory available.

PBS measures and enforces memory limits in two ways:

- On each host, by setting OS-level limits, using the limit system calls
- By periodically summing the usage recorded in the /proc entries.

Enforcement of mem is dependent on the following:

- Adding $enforce mem to the MOM's config file
- The job requesting or inheriting a default value for mem

You can configure default qsub parameters in the default_qsub_arguments server attribute, or set memory defaults at the server or queue. See section 5.9.3 “Specifying Job Default Resources” on page 338.

5.15.3.3.1  Job Memory Limit Enforcement on UNIX

By default, memory limits are not enforced. To enforce mem resource usage, put $enforce mem into MOM’s config file, and set defaults for mem so that each job inherits a value if it does not request it.

The mem resource can be enforced at both the job level and the vnode level. The job-wide limit is the smaller of a job-wide resource request and the sum of that for all chunks. The vnode-level limit is the sum for all chunks on that host.

Job-wide limits are enforced by MOM polling the working set size of all processes in the job’s session. Jobs that exceed their specified amount of physical memory are killed. A job may exceed its limit for the period between two polling cycles. See section 3.6.1 “Configuring MOM’s Polling Cycle” on page 65.

Per-process limits are enforced by the operating system kernel. PBS calls the kernel call setrlimit() to set the limit for the top process (the shell), and any process started by the shell inherits those limits. PBS does not know whether the kernel kills a process for exceeding the limit.
If a user submits a job with a job limit, but not per-process limits (\texttt{qsub -l cput=10:00}) then PBS sets the per-process limit to the same value. If a user submits a job with both job and per-process limits, then the per-process limit is set to the lesser of the two values.

Example: a job is submitted with \texttt{qsub -lcput=10:00}

- There are two CPU-intensive processes which use 5:01 each. The job will be killed by PBS for exceeding the \texttt{cput} limit. 5:01 + 5:01 is greater than 10:00.
- There is one CPU-intensive process which uses 10:01. It is very likely that the kernel will detect it first.
- There is one process that uses 0:02 and another that uses 10:00. PBS may or may not catch it before the kernel does depending on exactly when the polling takes place.

If a job is submitted with a \texttt{pmem} limit, or without \texttt{pmem} but with a \texttt{mem} limit, PBS uses the \texttt{setrlimit(2)} call to set the limit. For most operating systems, \texttt{setrlimit()} is called with \texttt{RLIMIT\_RSS} which limits the Resident Set (working set size). This is not a hard limit, but advice to the kernel. This process becomes a prime candidate to have memory pages reclaimed.

If \texttt{vmem} is specified and no single process exceeds that limit, but the total usage by all the processes in the job does, then PBS enforces the \texttt{vmem} limit, but not the \texttt{pvmem} limit, and logs a message. PBS uses MOM polling to enforce \texttt{vmem}.

The limit for \texttt{pmem} is enforced if the job specifies, or inherits a default value for, \texttt{pmem}. When \texttt{pmem} is enforced, the limit is set to the smaller of \texttt{mem} and \texttt{pmem}. Enforcement is done by the kernel, and applies to any single process in the job.

The limit for \texttt{pvmem} is enforced if the job specifies, or inherits a default value for, \texttt{pvmem}. When \texttt{pvmem} is enforced, the limit is set to the smaller of \texttt{vmem} and \texttt{pvmem}. Enforcement is done by the kernel, and applies to any single process in the job.
The following table shows which OS resource limits can be used by each operating system.

### Table 5-13: RLIMIT Usage in PBS Professional

<table>
<thead>
<tr>
<th>OS</th>
<th>file</th>
<th>mem/pmem</th>
<th>vmem/pvmem</th>
<th>cput/pcput</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>RLIMIT_FSIZE</td>
<td>RLIMIT_RSS</td>
<td>RLIMIT_DATA</td>
<td>RLIMIT_CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RLIMITSTACK</td>
<td></td>
</tr>
<tr>
<td>HP-UX</td>
<td>RLIMIT_FSIZE</td>
<td>RLIMIT_RSS</td>
<td>RLIMIT_AS</td>
<td>RLIMIT_CPU</td>
</tr>
<tr>
<td>Linux</td>
<td>RLIMIT_FSIZE</td>
<td>RLIMIT_RSS</td>
<td>RLIMIT_AS</td>
<td>RLIMIT_CPU</td>
</tr>
<tr>
<td>SunOS</td>
<td>RLIMIT_FSIZE</td>
<td>RLIMIT_DATA</td>
<td>RLIMIT_VMEM</td>
<td>RLIMIT_CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RLIMIT_STACK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Super-</td>
<td>RLIMIT_FSIZE</td>
<td>RLIMIT_UMEM</td>
<td>ignored</td>
<td>RLIMIT_CPU</td>
</tr>
<tr>
<td>UX</td>
<td></td>
<td>RLIMIT_DATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RLIMIT_STACK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that RLIMIT_RSS, RLIMIT_UMEM, and RLIMIT_VMEM are not standardized (i.e. do not appear in the Open Group Base Specifications Issue 6).

#### 5.15.3.3.2 Sun Solaris-specific Memory Enforcement

Solaris does not support RLIMIT_RSS, but instead has RLIMIT_DATA and RLIMIT_STACK, which are hard limits. On Solaris or another Open Group standards-compliant OS, a malloc() call that exceeds the limit will return NULL. This behavior is different from other operating systems and may result in the program (such as a user’s application) receiving a SIGSEGV signal.
5.15.3.3 Memory Enforcement on cpusets

There should be no need to do so: either the vnode containing the memory in question has been allocated exclusively (in which case no other job will also be allocated this vnode, hence this memory) or the vnode is shareable (in which case using mem_exclusive would prevent two CPU sets from sharing the memory). Essentially, PBS enforces the equivalent of mem_exclusive by itself.

5.15.3.4 Job ncpus Limit Enforcement

Enforcement of the ncpus limit (number of CPUs used) is available on all platforms. The ncpus limit can be enforced using average CPU usage, burst CPU usage, or both. By default, enforcement of the ncpus limit is off. See “$enforce <limit>” on page 315 of the PBS Professional Reference Guide.

5.15.3.4.1 Average CPU Usage Enforcement

Each MOM enforces cpuaverage independently, per MOM, not per vnode. To enforce average CPU usage, put $enforce cpuaverage in MOM’s config file. You can set the values of three variables to control how the average is enforced. These are shown in the following table.

<p>| Table 5-14: Variables Used in Average CPU Usage |
|-----------------|---------|------------------------------------------------|</p>
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpuaverage</td>
<td>Boolean</td>
<td>If present (=True), MOM enforces ncpus when the average CPU usage over the job’s lifetime usage is greater than the specified limit.</td>
</tr>
<tr>
<td>average_trialperiod</td>
<td>integer</td>
<td>Modifies cpuaverage. Minimum job walltime before enforcement begins. Seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: False</td>
</tr>
</tbody>
</table>


Enforcement of cpuaverage is based on the polled sum of CPU time for all processes in the job. The limit is checked each poll period. Enforcement begins after the job has had average_trialperiod seconds of wall-time. Then, the job is killed if the following is true:

\[
\left(\frac{\text{cput}}{\text{walltime}}\right) > \left(\text{ncpus} \times \text{average_cpufactor} + \frac{\text{average_percent_over}}{100}\right)
\]

### 5.15.3.4.2 CPU Burst Usage Enforcement

To enforce burst CPU usage, put `$enforce cpuburst` in MOM’s `config` file. You can set the values of four variables to control how the burst usage is enforced. These are shown in the following table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpuburst</td>
<td>Boolean</td>
<td>If present (=True), MOM enforces ncpus when CPU burst usage exceeds specified limit.</td>
<td>False</td>
</tr>
<tr>
<td>delta_percent_over</td>
<td>integer</td>
<td>Modifies cpuburst. Percentage over limit to be allowed.</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>average_percent_over</td>
<td>integer</td>
<td>Modifies cpuaverage. Percentage by which the job may exceed ncpus limit.</td>
<td>50</td>
</tr>
<tr>
<td>average_cpufactor</td>
<td>float</td>
<td>Modifies cpuaverage. ncpus limit is multiplied by this factor to produce actual limit.</td>
<td>1.025</td>
</tr>
</tbody>
</table>

Table 5-14: Variables Used in Average CPU Usage

Table 5-15: Variables Used in CPU Burst
MOM calculates an integer value called `cpupercent` each polling cycle. This is a moving weighted average of CPU usage for the cycle, given as the average percentage usage of one CPU. For example, a value of 50 means that during a certain period, the job used 50 percent of one CPU. A value of 300 means that during the period, the job used an average of three CPUs.

\[ new\_percent = \frac{\text{change\_in\_cpu\_time} \times 100}{\text{change\_in\_walltime}} \]

\[ weight = \text{delta\_weight[up|down]} \times \frac{\text{walltime}}{\text{max\_poll\_period}} \]

\[ new\_cpupercent = (new\_percent \times weight) + (old\_cpupercent \times (1 - weight)) \]

`delta\_weight\_up` is used if `new\_percent` is higher than the old `cpupercent` value. `delta\_weight\_down` is used if `new\_percent` is lower than the old `cpupercent` value. `delta\_weight\_[up|down]` controls the speed with which `cpupercent` changes. If `delta\_weight\_[up|down]` is 0.0, the value for `cpupercent` does not change over time. If it is 1.0, `cpupercent` will take the value of `new\_percent` for the poll period. In this case `cpupercent` changes quickly.

However, `cpupercent` is controlled so that it stays at the greater of the average over the entire run or `ncpus\times100`.

`max\_poll\_period` is the maximum time between samples, set in MOM’s `config` file by `$max\_check\_poll`, with a default of 120 seconds.

### Table 5-15: Variables Used in CPU Burst

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>delta_cpufactor</td>
<td>float</td>
<td>Modifies <code>cpuburst</code>. <code>ncpus</code> limit is multiplied by this factor to produce actual limit.</td>
<td>1.5</td>
</tr>
<tr>
<td>delta_weightup</td>
<td>float</td>
<td>Modifies <code>cpuburst</code>. Weighting factor for smoothing burst usage when average is increasing.</td>
<td>0.4</td>
</tr>
<tr>
<td>delta_weightdown</td>
<td>float</td>
<td>Modifies <code>cpuburst</code>. Weighting factor for smoothing burst usage when average is decreasing.</td>
<td>0.1</td>
</tr>
</tbody>
</table>
The job is killed if the following is true:

\[ \text{new\_cpupercent} > ((\text{ncpus} \times 100 \times \text{delta\_cpufactor}) + \text{delta\_percent\_over}) \]

The following entries in MOM’s `config` file turn on enforcement of both average and burst with the default values:

- `$enforce cpuaverage`
- `$enforce cpuburst`
- `$enforce delta_percent_over 50`
- `$enforce delta_cpufactor 1.05`
- `$enforce delta_weightup 0.4`
- `$enforce delta_weightdown 0.1`
- `$enforce average_percent_over 50`
- `$enforce average_cpufactor 1.025`
- `$enforce average_trialperiod 120`

The `cpuburst` and `cpuaverage` information show up in MOM’s log file, whether or not they have been configured in `mom_priv/config`. This is so a site can test different parameters for `cpuburst/cpuaverage` before enabling enforcement. You can see the effect of any change to the parameters on your job mix before "going live".

Note that if the job creates a child process whose usage is not tracked by MOM during its lifetime, CPU usage can appear to jump dramatically when the child process exits. This is because the CPU time for the child process is assigned to its parent when the child process exits. MOM may see a big jump in `cpupercent`, and kill the job.

### 5.15.3.4.3 Job Memory Limit Restrictions

Enforcement of `mem` resource usage is available on all UNIX platforms, but not Windows.
5.15.3.5 Changing Job Limits

The qalter command is used to change job limits, with these restrictions:

- A non-privileged user may only lower the limits for job resources
- A Manager or Operator may lower or raise requested resource limits, except for per-process limits such as pcput and pmem, because these are set when the process starts, and enforced by the kernel.

See “qalter” on page 152 of the PBS Professional Reference Guide.

5.16 Where Resource Information Is Kept

Definitions and values for PBS resources are kept in the following files, attributes, and parameters. Attributes specifying resource limits are not listed here. They are listed in section 5.15.1.7 “Resource Usage Limit Attributes for Server and Queues” on page 420 and section 5.15.1.14 “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1” on page 432.

5.16.1 Files

PBS_HOME/server_priv/resourcedef

Contains definitions of custom resources. Format:

<resource name> [type=<type>] [flag=<flags>]

Example:

LocalScratch type=long flag=h
FloatLicense type=long
SharedScratch type=long

See section 5.14.2.3 “The resourcedef File” on page 358.

PBS_HOME/sched_priv/sched_config

resources: line
In order for scheduler to be able to schedule using a resource, the resource must be listed in the resources: line. Format:

resources: "<resource name>, [<resource name>, ...]"

Example:

resources: "ncpus, mem, arch, [...], LocalScratch, FloatLicense, SharedScratch"

The only exception is host-level Boolean resources, which do not need to appear in the resources: line.

server_dyn_res: line

Each dynamic server resource must be listed in its own server_dyn_res: line. Format:

server_dyn_res: "<resource name> !<path to script/command>"

Example:

server_dyn_res: "SharedScratch !/usr/local/bin/serverdynscratch.pl"

mom_resources: line

Dynamic host resources must be listed in the mom_resources: line. Format:

mom_resources: "<resource name>"

Example:

mom_resources: "LocalScratch"

PBS_HOME/mom_priv/config

Contains MOM configuration parameters and any local resources. Format:

<resource name> !<path to script/command>

Example:

LocalScratch !/usr/local/bin/localscratch.pl

See “MOM Parameters” on page 311 of the PBS Professional Reference Guide.
5.16.2 MOM Configuration Parameters

$\textit{cputmult <factor>}$
This sets a factor used to adjust CPU time used by each job. This allows adjustment of time charged and limits enforced where jobs run on a system with different CPU performance. If MOM’s system is faster than the reference system, set factor to a decimal value greater than 1.0. For example:

$cputmult 1.5$

If MOM’s system is slower, set factor to a value between 1.0 and 0.0. For example:

$cputmult 0.75$

$\textit{wallmult <factor>}$
Each job’s \textit{walltime} usage is multiplied by this factor. For example:

$wallmult 1.5$

5.16.3 Server Attributes

default\_chunk
The list of resources which will be inserted into each chunk of a job’s select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user.

Format: \textit{String}.

Form:

$default\_chunk.<resource>=<value>, default\_chunk.<resource>=<value>,...$

Default: None

default\_qsub\_arguments
Arguments that are automatically added to the \texttt{qsub} command. Any valid arguments to \texttt{qsub} command, such as job attributes. Setting a job attribute via
default_qsub_arguments sets that attribute for each job which does not explicitly override it. See qsub(1B). Settable by the administrator via the qmgr command. Overrides standard defaults. Overridden by arguments given on the command line and in script directives.

Example:

```
Qmgr: set server default_qsub_arguments="-r y
-N MyJob"
```

Form: String
Default: None

resources_available.<resource name>
The list of available resources and their values defined on the server. Each resource is listed on a separate line.

Format: String.
Form: resources_available.<resource>=<value>
Default: None

resources_default.<resource name>
The list of default resource values that are set as limits for jobs in this complex when a) the job does not specify a limit, and b) there is no queue default.

Format: String.
Form: resources_default.resource_name=value[,...]
Default: None

resources_assigned.<resource name>
The total of each type of consumable resource allocated to jobs running in this complex. Read-only.

Format: String.
Form: resources_assigned.<resource>=<val>[,resources_assigned.<resource>=<val>,...]
Default: None
5.16.4 Reservation Attributes

Resource_List.<resource name>
The list of resources allocated to the reservation. Jobs running in the reservation cannot use in aggregate more than the specified amount of a resource.

Format: String

Form: Resource_List.<res>=<val>, Resource_List.<res>=<val>, ...

Default: None

5.16.5 Queue Attributes

default_chunk.<resource name>
The list of resources which will be inserted into each chunk of a job’s select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user. Applies only to execution queues.

Format: String.

Form: default_chunk.<resource>=<value>, default_chunk.<resource>=<value>, ...

Default: None

resources_default.<resource name>
The list of default resource values which are set as limits for a job residing in this queue and for which the job did not specify a limit. If not set, the default limit for a job is determined by the first of the following attributes which is set: server’s resources_default, queue’s resources_max, server’s resources_max. If none of these is set, the job gets unlimited resource usage

Format: String.

Form: resources_default.<resource name>=<value>, resources_default.<resource_name>=<value>, ...
Default: None

resources_assigned.<resource name>

The total for each kind of consumable resource allocated to jobs running from this queue. Read-only.

Format: String.

Form: resources_assigned.<res>=<val><new-line>resources_assigned.<res>=<val><newline>...

Default: None

resources_available.<resource name>

The list of resources and amounts available to jobs running in this queue. The sum of the resource of each type used by all jobs running from this queue cannot exceed the total amount listed here. See “qmgr” on page 178 of the PBS Professional Reference Guide.

Format: String.

Form:
resources_available.<resource_name>=<value><new-line>resources_available.<resource_name>=<value><newline>...

Default: None

5.16.6 Vnode Attributes

resources_available.<resource name>

The list of resources and the amounts available on this vnode. If not explicitly set, the amount shown is that reported by the pbs_mom running on the vnode. If a resource value is explicitly set, that value is retained across restarts.

Format: String.

Form: resources_available.<resource_name>=<value>, resources_available.<resource_name> = <value>, ...

Default: None
sharing

Specifies whether more than one job at a time can use the resources of the vnode or the vnode’s host. Either (1) the vnode or host is allocated exclusively to one job, or (2) the vnode’s or host’s unused resources are available to other jobs. Can be set using `pbs_mom -s insert` only. Behavior is determined by a combination of the `sharing` attribute and a job’s placement directive. See “sharing” on page 426 of the PBS Professional Reference Guide.

pcpus

The number of physical CPUs on the vnode. This is set to the number of CPUs available when MOM starts. For a multiple-vnode MOM, only the natural vnode has `pcpus`.

Format: `Integer`
Default: Number of CPUs on startup

resources_assigned.<resource name>

The total amount of each consumable resource allocated to jobs running on this vnode. Applies only to execution queues. Read-only.

Format: `String`.

Form:

```
resources_assigned.<resource>=<value>[,resources_assigned.<resource>=<value>]
```

Default: None

### 5.16.7 Job Attributes

Resource_List.<resource name>

The list of resources required by the job. List is a set of `<name>=<value>` strings. The meaning of name and value is dependent upon defined resources. Each value establishes the limit of usage of that resource. If not set, the value for a resource may be determined by a queue or server default established by the administrator. See section 5.9.2 “Resources Requested by Job” on page 338.

Format: `String`. 
Form: `Resource_List.<res>=<value>`,  
`Resource_List.<res>=<value>, ...`  

Default: None

resources_used.<resource name>
- The amount of each resource actually used by the job.
- Read-only.

Format: *String*

Form: List of `<name>=<value>` pairs:

```
resources_used.<res>=<val>,
resources_used.<res>=<val>
```

## 5.17 Viewing Resource Information

You can see attribute values of resources for the server, queues, and vnodes using the `qmgr` or `pbsnodes` commands. The value in the server, queue, or vnode `resources_assigned` attribute is the amount explicitly requested by the job.

You can see job attribute values using the `qstat` command. The value in the job’s `Resource_List` attribute is the amount explicitly requested by the job. See section 5.9.2 “Resources Requested by Job” on page 338.

The following table summarizes how to find resource information:

<table>
<thead>
<tr>
<th>Location</th>
<th>Item to View</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>default_chunk,</td>
<td><code>qmgr</code>, <code>qstat</code>, <code>pbsnodes</code></td>
</tr>
<tr>
<td></td>
<td>default_qsub_arguments,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>resources_available,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>resources_assigned,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>resources_default</td>
<td></td>
</tr>
<tr>
<td></td>
<td>resources_used.&lt;resource name&gt;</td>
<td></td>
</tr>
<tr>
<td>scheduler</td>
<td>sched_config_file</td>
<td>Favorite editor or viewer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5-16: How to View Resource Information
Every consumable resource, for example mem, can appear in four PBS attributes. These attributes are used in the following elements of PBS:

**Table 5-16: How to View Resource Information**

<table>
<thead>
<tr>
<th>Location</th>
<th>Item to View</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>queues</td>
<td>default_chunk, resources_available, resources_assigned, resources_default</td>
<td>qmgr, qstat</td>
</tr>
<tr>
<td>MOM and vnodes</td>
<td>resources_available, sharing, pcpus, resources_assigned</td>
<td>qmgr, pbsnodes</td>
</tr>
<tr>
<td></td>
<td>mom_config file</td>
<td>Favorite editor or viewer</td>
</tr>
<tr>
<td>job</td>
<td>Resource_List</td>
<td>qstat</td>
</tr>
<tr>
<td>reservation</td>
<td>Resource_List</td>
<td>pbs_rstat -f</td>
</tr>
<tr>
<td>accounting</td>
<td>resource_assigned entry in accounting log</td>
<td>Favorite editor or viewer</td>
</tr>
</tbody>
</table>

**Table 5-17: Values Associated with Consumable Resources**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Node</th>
<th>Queue</th>
<th>Server</th>
<th>Accounting Log</th>
<th>Job</th>
<th>Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>resources_available</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>resources_assigned</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resources_used</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Resource_List</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
5.17.1 Resource Information in Accounting Logs

You can see accounting values in the accounting log file. The accounting log $S$ record is written at the start of a job, and the $E$ record is written at the end of a job. The accounting log $B$ record is written at the beginning of a reservation.

Each consumable resource allocated to or taken up by a job is reported separately in a `resource_assigned` accounting entry in the job’s $E$ and $S$ records. The `resource_assigned` entry is not a job attribute; it is simply an entry in the accounting log.

Consumable job resources actually used by the job are recorded in the job’s `resources_used` attribute, and are reported in the accounting log.

The value reported in the `resource_assigned` accounting entry is the amount assigned to a job or that a job prevents other jobs from using, which is different from the amount the job requested and used. For example, if a job requests one CPU on an Altix that has four CPUs per blade/vnode and that vnode is allocated exclusively to the job, even though the job requested one CPU, it is assigned all 4 CPUs. In this example, `resource_assigned` reports 4 CPUs, and `resources_used` reports 1 CPU.

Resources requested for a job are recorded in the job’s `Resource_List` attribute, and reported in the accounting log $E$ and $S$ records for the job.

Resources requested for a reservation are recorded in the reservation’s `Resource_List` attribute, and reported in the accounting log $B$ record for the reservation.

5.17.2 Resource Information in Daemon Logs

At the end of each job, the server logs the values in the job’s `resources_used` attribute, at event class 0x0010.

Upon startup, MOM logs the number of CPUs reported by the OS, at event class 0x0002.
At the end of each job, the MOM logs cput and mem used by each job, and cput used by each job task, at event class 0x0100.

### 5.17.3 Finding Current Value

You can find the current value of a resource by subtracting the amount being used from the amount that is defined.

Use the `qstat -Bf` command, and grep for `resources_available.<resource>` and `resources_used.<resource>`. To find the current amount not being used, subtract `resources_used.<resource>` from `resources_available.<resource>`.

### 5.17.4 Restrictions on Viewing Resources

- Dynamic resources shown in `qstat` do not display the current value, they display the most recent retrieval. Dynamic resources have no `resources_available.<resource>` representation anywhere in PBS.
- Local static host-level resources cannot be viewed via `qstat` or managed via `qmgr`.

### 5.18 Resource Recommendations and Caveats

- It is not recommended to set the value for `resources_available.ncpus`. The exception is when you want to oversubscribe CPUs. See section 10.4.5.1.3 “How To Share CPUs” on page 770.
- It is not recommended to change the value of `ncpus` at vnodes on a multi-vnoded machine.
- If you want to limit how many jobs are run, or how much of each resource is used, use the new limits. See section 5.15 “Managing Resource Usage” on page 409.
- It is not recommended to create local host-level resources by defining
them in the MOM configuration file.

- On the Altix, do not set the values for `mem`, `vmem` or `ncpus` on the natural vnode. If any of these resources has been explicitly set to a non-zero value on the natural vnode, set `resources_available.ncpus`, `resources_available.mem` and `resources_available.vmem` to zero on each natural vnode:

- Do not attempt to set values for `resources_available.<resource>` for dynamic resources.

- Externally-managed licenses may not be available when PBS thinks they are. PBS doesn't actually check out externally-managed licenses; the application being run inside the job's session does that. Between the time that the scheduler queries for licenses, and the time the application checks them out, another application may take the licenses. In addition, some applications request varying amounts of tokens during a job run.

- Jobs may be placed on different vnodes from those where they would have run in earlier versions of PBS. This is because a job’s resource request will no longer match the same resources on the server, queues and vnodes.

- While users cannot request custom resources that are created with the `x` flag, jobs can inherit these as defaults from the server or queue `resources_default.<resource>` attribute.

- A `qsub` or `pbs_rsub` hook does not have resources inherited from the server or queue `resources_default` or `default_chunk` as an input argument.

- Resources assigned from the `default_qsub_arguments` server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag, whether that resource was requested by the user or came from `default_qsub_arguments`. Be aware that creating custom resources with permission flags and then using these in the `default_qsub_arguments` server attribute can cause jobs to be rejected. See section 5.14.2.5 “Resource Permission Flags” on page 362.

- Numeric dynamic resources cannot have the `q` or `n` flags set. This would cause these resources to be underused. These resources are
tracked automatically by scheduler.

- The behavior of several command-line interfaces is dependent on resource permission flags. These interfaces are those which view or request resources or modify resource requests:

  **pbsnodes**
  Users cannot view restricted host-level custom resources.

  **pbs_rstat**
  Users cannot view restricted reservation resources.

  **pbs_rsub**
  Users cannot request restricted custom resources for reservations.

  **qalter**
  Users cannot alter a restricted resource.

  **qmgr**
  Users cannot print or list a restricted resource.

  **qselect**
  Users cannot specify restricted resources via -l Resource_List.

  **qsub**
  Users cannot request a restricted resource.

  **qstat**
  Users cannot view a restricted resource.

- Do not set values for any resources, except those such as shared scratch space or floating licenses, at the server or a queue, because the scheduler will not allocate more than the specified value. For example, if you set `resources_available.walltime` at the server to 10:00:00, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.

- If a job is submitted without a request for a particular resource, and no defaults for that resource are set at the server or queue, and either the server or queue has `resources_max.<resource>` set, the job inherits that maximum value. If the queue has `resources_max.<resource>` set, the job inherits the queue value, and if not, the job inherits the server value.

- When setting global static vnode resources on multi-vnode machines, follow the rules in section 3.5.2 “Choosing Configuration Method” on
• Do not create custom resources with the same names or prefixes that PBS uses when creating custom resources for specific systems. See “Custom Cray Resources” on page 357 of the PBS Professional Reference Guide.

• Do not set `resources_available.place` for a vnode.
Hooks

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code.

This chapter describes how hooks can be used, how they work, the interface to hooks provided by the pbs module, how to create and deploy hooks, and how to get information about hooks.

Please read the entire chapter before writing any hooks.

6.1 Chapter Contents

6.1 Chapter Contents ........................................ 461
6.2 Introduction to Hooks ................................. 463
6.3 Definitions ............................................. 464
6.2 Introduction to Hooks

A hook is a block of Python code that is triggered in response to queueing a job, modifying a job, moving a job, running a job, provisioning a vnode, or submitting a PBS reservation. Each hook can accept (allow) or reject (prevent) the action that triggers it. The hook can modify the input parameters given for the action. The hook can also make calls to functions external to PBS. PBS provides an interface for use by hooks. This interface allows hooks to read and/or modify things such as job, server, vnode, and queue attributes, and the event that triggered the hook.

The Administrator must create any hooks while working on the server host only. No special configuration of PBS is required in order to use hooks.
6.3 Definitions

**Action**
A PBS operation or state transition. The actions that hooks can affect are submitting a job, altering a job, running a job, making a reservation, and moving a job to another queue.

**Accept an Action**
The hook allows the action to take place.

**pbs Module**
The *pbs module* is an interface to PBS and the hook environment. The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code.

**Reject an Action**
The hook prevents the action from taking place.

6.4 How Hooks Can Be Used

Hooks can evaluate job or reservation attributes or perform other operations outside PBS such as look up projects in databases, update web pages, or verify available disk space. For *qsub* and *qalter*, a hook can change job attributes and resources before accepting the operation; for *qmove*, a hook can change the destination queue; for *pbs_rsub*, a hook can change reservation attributes before accepting the operation; for *qrun* and the *pbs_runjob()* API, a hook can put a hold on the job, release a hold on the job, change the job’s execution time, or change a vnode’s state. Hooks are the mechanism by which vnodes are provisioned.
6.4.1 Examples of Using Hooks

6.4.1.1 Routing Jobs Using Hooks

• Route jobs into specific queues or between queues:
  - Automatically route interactive jobs into a particular execution queue
  - Move a job to another queue; for example, if project allocation is used up, move job to “background” queue

• Reject job submissions that do not specify a valid queue, printing an error message explaining the problem

• Enable project-based ACLs for queues to make sure the appropriate job runs in the correct queue

6.4.1.2 Managing Job Resource Requests

• Reject improperly specified jobs:
  - Reject jobs which do not specify walltime
  - Reject jobs that request a number of processors that is not a multiple of 8
  - Reject jobs requesting a specific queue, but not requesting memory
  - Reject jobs whose processors per node is not specified or is not numeric

• Modify job resource requests:
  - Apply default memory limit to jobs that request a specific queue
  - Check on requested CPU and memory and modify these or supply them if missing
  - Adjust for the fact that users ask for 2GB on an Altix that has 2GB physical memory, but only 1.8 GB available memory, by changing the memory request to 1.8GB

6.4.1.3 Managing Access to Resources for Users and Jobs

• Compensate for dissimilar system capabilities; for example, allow users
to use more CPUs only if they use old, slow machines.

• Limit reservations submitted by users to a maximum amount of resources and walltime, but do not limit reservations submitted by PBS administrators.

6.4.1.4 Ensuring Efficient Use of Resources

• Reject parallel jobs for some queues.
• Set default properties, for example, if “myri” is not set, set it to “False” to ensure Myrinet is used only for Myrinet jobs.

6.4.1.5 Ensuring That Jobs Run Properly

• Make sure that jobs, or all jobs in a queue, request exclusive access (\(-1\ place=excl\)).
• Reject multi-host jobs, restricting all jobs to a single Altix.
• Put a hold on the job if there isn’t enough scratch space when the job is submitted.
• Reject jobs that could cause problems, based on the user and type of job that have caused previous problems. For example, if Bill’s Abaqus jobs crash the system, reject new Abaqus jobs from Bill.
• Validate an input deck before the job is submitted.

6.4.1.6 Controlling Interactive Jobs

• Control interactive job submission; for example, enable or disable interactive jobs at the server or queue level.

6.4.1.7 Communicating Information to Users

• Report useful error messages back to the user, e.g., "You do not have sufficient walltime left to run your job for 1:00:00. Your walltime balance is 00:30:00."
6.4.1.8  Converting Requests to Usable Format

- Convert from ALPS-specific resource request strings into PBS-specific job requirements.
- Automatically translate old syntax to new syntax.

6.4.1.9  Help Scheduling Jobs

- Change scheduling according to user and job:
  - Set initial user-dependent coefficients for the scheduling formula. For example, set values of custom resources based on job attributes and user
  - Set whether or not the job is rerunnable, based on user
  - Calculate CPH (CPH == total ncpus * walltime in hours) and set a custom CPH job resource to the value
- Set initial priorities for jobs

6.4.1.10  Managing User Activity

- Reject jobs from blacklisted users.
- Prevent users from using qalter to change their jobs in any way, allowing only administrators to qalter jobs.
- Prevent users from bypassing controls: disallow a job being submitted to queueA in a held state and then being qmoved to queueB where the job would not have passed hook checks for queueB initially. For example, if a qsub hook disallows interactive jobs for queueB, the administrator also needs to ensure that an interactive job is not initially submitted to queueA and later moved to queueB.
- Prevent users from overriding node_group_key with qsub -lplace = group = X, or with qalter.
- Restrict the ability to submit a reservation to PBS administrators only.

6.4.1.11  Enabling Accounting and Validation

- Make sure correct project designation is used: if no account string is
found, look up username in database to find appropriate project to use and add it as account string before submission.

- Submit job to correct queue based on project: check for project number and submit job to queues based on project type, e.g. project number 1234 jobs get submitted into “challenge” queue; similarly for “standard” queue, etc.

- Validate project before the job executes; if validation fails, do not start job, and print error message. Validation can be based on project name, and things like requested resources, for example, CPU hours.

### 6.4.1.12 Enforcing Security

- Reject jobs with invalid Kerberos tickets.

### 6.4.1.13 Provisioning Vnodes

- Provision a vnode with a new AOE. See Chapter 8, "Provisioning", on page 599.

### 6.5 How Hooks Work

Hooks accept (allow) or reject (prevent) actions, modify input parameters, and change internal or external values. Hooks can be run before running a job, and during job submission, job modification, job move, and reservation creation. Hooks work with both the primary and secondary servers during failover. Hooks can only be created, run, or modified by the Administrator, and only on the host on which the server runs.

#### 6.5.1 Accepting or Rejecting Actions

Each action can have zero or more hooks. Each hook must either accept or reject its action. All of an action’s hooks are run when that action is to be performed. For PBS to perform an action, all hooks enabled for that action must accept the action. If any hook rejects the action, the action is not per-
formed by PBS. If a hook script doesn’t call `accept()` or `reject()`, and it doesn’t encounter an exception, PBS behaves as if the hook accepted the action. An action is always accepted, unless:

- `pbs.event().reject()` is called
- An unhandled exception is encountered
- The hook alarm has been triggered due to hook timeout being reached

When PBS executes the hooks for an action, it stops processing hooks at the first hook that rejects the action.

### 6.5.1.1 Examples of Accepting and Rejecting Actions

**Example 1:** Accepting an action: In this example, userA submits a job to queue Queue1, and the job submission action has two hooks: hook1 disallows jobs submitted by UserB, and hook2 disallows jobs being submitted directly to Queue2. Both hook1 and hook2 accept userA’s job submission to Queue1, so the submission goes ahead.

**Example 2:** Rejecting an action: In this example, userA uses the `qmove` command to try to move jobA from Queue1 to Queue2. The job move action has two hooks: hook3 disallows jobs being moved into Queue2, and hook4 disallows userB moving jobs out of Queue1. In this example, hook3 rejects the action, so the move operation is disallowed, even though hook4 would have accepted the action.

### 6.5.2 Modifying Input Parameters

#### 6.5.2.1 Modifying Job Submission, Job Move, Job Run, and Job and Reservation Parameters

**6.5.2.1.1 Modifying Job Submission (`qsub`)**

- When a job is submitted via `qsub`, hooks can modify the following things explicitly specified in the job submission:
  - Job attributes
  - Requested resources
- When a job is submitted via `qsub`, hooks can add resource requests to
Chapter 6  

Hooks

those specified in the job submission

• The input job attributes on which hooks operate are those that exist after all qsub processing is completed. These input attributes include:
  - Command line arguments
  - Script directives
  - Server default_qsub_arguments

• When a hook runs at job submission, the hook can affect only that job.

• For qsub hooks, the input job attributes do not include:
  - Server or queue resources_default or default_chunk.
  - Conversions from old syntax (-lnodes & -lncpus) to new select and place syntax

6.5.2.1.2  Modifying Job Change (qalter)

• When a job is changed via qalter, hooks can modify the arguments passed to qalter

• When a qalter hook runs, it can change the attributes of the job being qaltered

6.5.2.1.3  Modifying Job Move (qmove)

• When a job is moved via qmove, hooks can modify the arguments passed to qmove

• When a qmove hook runs, it can change the job’s destination queue to any queue on the default server

6.5.2.1.4  Modifying Reservation Creation (pbs_rsub)

• When an advance or standing reservation is created, hooks can modify the reservation’s attributes

• When an advance or standing reservation is created, hooks can specify additional attributes

• The input reservation attributes on which hooks operate are those that exist after all pbs_rsub processing of command line arguments is
completed

- For pbs_rsub hooks, the input job attributes do not include:
  - server or queue resources_default or default_chunk.
  - conversions from old syntax (-lnodes & -lncpus) to new select and place syntax

6.5.2.1.5 Modifying Running of Job (qrun)

- When the scheduler runs a job or the administrator runs a job using the qrun command, hooks can:
  - place a hold on the job
  - release a hold on the job
  - change the time the job is allowed to begin execution
- A qrun hook can modify a job (hold/release/delay) only if the hook rejects the job (does not allow it to be executed). A qrun hook cannot make any modifications to a job it will then allow to execute.

6.5.3 Changing Vnode State (qrun)

- A runjob hook can change a vnode’s state by setting the value of its state attribute.
- A runjob hook can modify the state of a vnode only if the hook rejects the action.

6.5.4 Where Hooks Can Be Used

Hooks can be inserted before the following actions:

- A job is accepted for submission (qsub)
- A job is altered (qalter)
- A job is moved to another server/queue (qmove)
- A reservation is accepted for submission (pbs_rsub)
- A job is run (qrun or pbs_runjob())
6.5.5  Failover

When the secondary server takes over for the primary server after the latter's host has gone down or becomes inaccessible, any hooks registered with the primary server continue to function under the secondary server.

Likewise, if the primary server comes back up and takes over as primary again, hooks registered while the secondary server was acting as primary continue to function.

The secondary server uses the same filesystem as the primary server. If the administrator creates a new hook while the secondary server has control, that hook will persist once the primary server takes over. Any hooks created, whether the primary or the secondary server is running, are stored in the same place, and are accessible by both servers.

6.5.6  Permissions and Access

Hooks can be created, deleted or modified by the Administrator only, and only the Administrator on the primary or secondary server’s host.

Under UNIX/Linux, root privilege is required in order to operate on hooks. Under Windows, this must be done from the installation account. For domained environments, the installation account must be a local account that is a member of the local Administrators group on the local computer. For standalone environments, the installation account must be a local account that is a member of the local Administrators group on the local computer.

A normal user cannot circumvent, disable, add, delete, or modify hooks or the environment in which the hooks are run.

Hooks run as the Administrator, and only on the primary or secondary server’s host.

6.5.7  Hook Names

Each hook has a unique name. This name must be alphanumeric, and start with an alphabetic character. The name must not begin with “PBS”.
The name of a hook can be a legal PBS object name, such as the name of a queue.

Hook names are case-sensitive.

### 6.6 Creating Hooks

#### 6.6.1 Introduction to Creating and Deleting Hooks

You create hooks using the `qmgr` command to create, delete, import, or export the hook. The `qmgr` command operates on the hook object.

Format of `qmgr` hooks directive:

```
command hook [hook_name] [attr OP value[,attr OP value,...]]
```

- `command` is `create`, `delete`, `set`, `unset`, `list`, `print`, `import`, `export`
- `import` loads the contents of a hook from an input file.
- `export` dumps the hook contents to a file.

#### 6.6.1.1 Overview of Creating a Hook

The following is an overview of the steps to create a hook. Each step is described in the following sections.

1. Use the `create hook` `qmgr` command to create an empty hook with the name you specify
2. Import the contents of a hook script into the hook
3. Set the hook’s trigger event
4. Set the hook’s order of execution, if there is another hook for the same event
5. Optionally, set the hook’s timeout
6. Enable the hook
Chapter 6  Hooks

For example:
Create the hook:

```
Qmgr: create hook hook1
```

Import the hook script named `hook1_script.py` into the hook:

```
Qmgr: import hook hook1 application/x-python
default /hooks/hook1_script.py
```

Make `hook1` a `queuejob` hook:

```
Qmgr: set hook hook1 event = queuejob
```

Make this the second `queuejob` hook:

```
Qmgr: set hook hook1 order = 2
```

Set the hook to time out after 60 seconds:

```
Qmgr: set hook hook1 alarm = 60
```

Enable `hook1`:

```
Qmgr: set hook hook1 enabled = True
```

### 6.6.2 Creating Empty Hooks

To create a hook, use the `create hook qmgr` command to create an empty hook with the name you specify:

The `create hook qmgr` command creates an empty hook.

Format for creating a hook:

```
create hook <hook name>
```

#### 6.6.2.1 Example of Creating an Empty Hook

To create the hook named “`hook1`”, specify a filename, for example “`.`

```
Qmgr: create hook hook1
```
6.6.3 Deleting Hooks

To delete a hook, you use the `delete hook qmgr` command.

Format for deleting a hook:

```
qmgr: delete hook <hook name>
```

6.6.3.1 Example of Deleting a Hook

To delete hook hook1:

```
qmgr: delete hook hook1
```

6.6.4 Importing Hooks

To import a hook, you import the contents of a hook script into the hook. You must specify a filename that is locally accessible to `qmgr` and the PBS Server.

Format for importing a hook:

```
import hook <hook_name> <content-type> <content-encoding>
{<input_file>|-}
```

This uses the contents of `<input_file>` or `stdin` (-) as the contents of hook `<hook_name>`.

- The `<input_file>` or `stdin` (-) data must have a format `<content-type>` and must be encoded with `<content-encoding>`.
- The only `<content-type>` currently supported is “application/x-python”.
- The allowed values for `<content-encoding>` are “default” (7bit) and “base64”.
- If the source of input is `stdin` (-) and `<content-encoding>` is “default”, then `qmgr` expects the input data to be terminated by EOF.
- If the source of input is `stdin` (-) and `<content-encoding>` is “base64”, then `qmgr` expects input data to be terminated by a blank
line.

- `<input_file>` must be locally accessible to both qmgr and the requested batch server.

- A relative path in `<input_file>` is relative to the directory where qmgr was executed.

- If a hook already has a content script, then that is overwritten by this import call.

- If the name of `<input_file>` contains spaces, as are used in Windows filenames, then `<input file>` must be quoted.

- There is no restriction on the size of the hook script.

### 6.6.4.1 Examples of Importing Hooks

Example 1: Given a Python script in ASCII text file "hello.py", this makes its contents into the script contents of hook1:

```bash
# cat hello.py
import pbs
pbs.event().job.comment="Hello, world"

# qmgr -c 'import hook hook1 application/x-python default hello.py'
```

Example 2: Given a base64-encoded file "hello.py.b64", qmgr unencodes the file's contents, and then makes this the script contents of hook1:

```bash
# cat hello.py.b64
cHJpbnQgImhlbGxvLCB3b3JsZCIK

# qmgr -c 'import hook hook1 application/x-python base64 hello.py.b64'
```
Example 3: Read stdin for text containing data until EOF, and make this into the script contents of hook1:

```bash
# qmgr -c 'import hook hook1 application/x-python
default -'
import pbs
pbs.event().job.comment="Hello from stdin"

Ctrl-D (UNIX/Linux)
Ctrl-Z (Windows)
```

Example 4: Read stdin for a base64-encoded string of data terminated by a blank line. PBS unencodes the data and makes this the script contents of hook1.

```bash
# qmgr -c 'import hook hook1 application/x-python
base64 -'
cHJpbnQgImlhblBGxvLCB3b3JsZCIK

Ctrl-D (UNIX/Linux)
Ctrl-Z (Windows)
```

### 6.6.5 Exporting Hooks

Format for exporting a hook:

```
export hook <hook_name> <content-type> <content-encoding> [<output_file>]
```

This dumps the script contents of hook `<hook_name>` into `<output_file>`, or stdout if `<output_file>` is not specified.

- The resulting `<output_file>` or stdout data is of `<content-type>` and `<content-encoding>`.
- The only `<content-type>` currently supported is “application/x-python”.
- The allowed values for `<content-encoding>` are “default” (7bit) and “base64”.
- `<output_file>` must be a path that can be created by `qmgr`.
- Any relative path in `<output_file>` is relative to the directory where
Hooks

qmgr was executed.

- If `<output_file>` already exists it is overwritten. If PBS is unable to overwrite the file due to ownership or permission problems, then an error message is displayed in `stderr`.

- If the `<output_file>` name contains spaces, like the ones used in Windows file names, then `<output_file>` must be enclosed in quotes.

### 6.6.5.1 Examples of Exporting Hooks

Example 1: Dumps hook1's script contents directly into a file "hello.py.out":

```bash
# qmgr -c 'export hook hook1 application/x-python
default hello.py'
# cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
```

Example 2: To dump the script contents of a hook 'hook1' into a file in "\My Hooks\hook1.py":

```bash
Qmgr: export hook hook1 application/x-python
default "\My Hooks\hook1.py"
```

Example 3: Dumps hook1's script contents base64-encoded into a file called "hello.py.b64":

```bash
# qmgr -c "export hook hook1 application/x-python
base64 hello.py.b64"
# cat hello.py.b64
chJpbnQgImhlbGxvLCB3b3JsZCIK
```

Example 4: Dumps hook1's script contents directly to stdout:

```bash
# qmgr -c "export hook hook1 application/x-python
default"
import pbs
pbs.event().job.comment="Hello, world"
```
Example 5: Dumps hook1's script contents base64-encoded into stdout:

```
# qmgr -c "export hook hook1 application/x-python
base64"

\bhCJpbnQgImlhGxvLCB3b3JsZCIK
```

### 6.7 Configuring Hooks

#### 6.7.1 Hook Attributes

You configure a hook using the `qmgr` command to set or unset its attributes:

```
Qmgr: set hook <hook name> <attribute name> = <value>
```

An unset hook attribute takes the default value for that attribute.

Hook attributes can be viewed via `qmgr`:

```
Qmgr: list hook <hook name>
```

Hook attributes are listed in the following table:

<table>
<thead>
<tr>
<th>Attribute name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type=&quot;site&quot;</td>
<td>Hook type can be “site”. Type “site” is the only value allowed in a create or set command, and the only value listed in a “list” or “print” command. Default value: “site”.</td>
</tr>
</tbody>
</table>
## Hooks

### Table 6-1: Hook Attributes

<table>
<thead>
<tr>
<th>Attribute name and value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled=&lt;Boolean&gt;</td>
<td>Determines whether or not a hook is run when its triggering event occurs. If a hook's enabled attribute is True, the hook is run. Default value: True</td>
</tr>
<tr>
<td>User=pbsadmin</td>
<td>Specifies who executes the hook. On UNIX, this is root. On Windows, this is simply a substitute for the PBS service account; it is not the name of the PBS service account. Default value: pbsadmin</td>
</tr>
<tr>
<td>event=&lt;event_string_array&gt;</td>
<td>List of events that trigger the hook. Can be operated on with “=”, “+=”, or “-=” operators. Valid events are: queuejob, modifyjob, resvsub, movejob, runjob, provision. The provision event cannot be combined with any other events. See section 6.9.7.1 “Event Type Objects” on page 505. Default value: ““ =none, meaning the hook will not be triggered.</td>
</tr>
</tbody>
</table>
### 6.7.2 Setting Hook Trigger Events

To set the events that will cause a hook to be triggered, use the `set hook <hook name> event qmgr` command. You can add triggering events to a hook.

To set one event:

```
Qmgr: set hook <hook name> event = <event name>
```

Designate triggers for events by setting `<event name>` to one of:

<table>
<thead>
<tr>
<th>Table 6-2: Hook Trigger Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action (Event)</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Accepting job into queue</td>
</tr>
<tr>
<td>Modifying job</td>
</tr>
<tr>
<td>Moving job</td>
</tr>
<tr>
<td>Submitting reservation</td>
</tr>
<tr>
<td>Running job</td>
</tr>
</tbody>
</table>
To add an event:

\[ Qmgr: \text{set hook} \ <\text{hook name}> \ \text{event} \ += \ <\text{event name}> \]

### 6.7.2.1 Example of Setting Hook Trigger Events

To set the events that will cause hook UserFilter to be triggered:

\[ Qmgr: \text{set hook UserFilter event} = \text{queuejob} \]
\[ Qmgr: \text{set hook UserFilter event} += \text{modifyjob} \]

Set two events at once:

\[ Qmgr: \text{set hook UserFilter event} = \text{"queuejob, modifyjob"} \]

You must enclose the value in double quotes if it contains a comma.

### 6.7.3 Enabling and Disabling Hooks

A hook is either enabled, and will run when its action happens, or is disabled, and will not run. Hooks are enabled by default.

Format to enable a hook:

\[ Qmgr: \text{set hook} \ <\text{hook name}> \ \text{enabled}=\text{True} \]

Format to disable a hook:

\[ Qmgr: \text{set hook} \ <\text{hook name}> \ \text{enabled}=\text{False} \]

### 6.7.3.1 Example of Enabling and Disabling Hooks

To enable hook1:

\[ Qmgr: \text{set hook hook1 enabled}=\text{True} \]

To disable hook1:

\[ Qmgr: \text{set hook hook1 enabled}=\text{False} \]
6.7.4 Setting the Relative Order of Hook Execution

When there are multiple hooks for one action, you may wish to specify the order in which the hooks are run. The order in which the hooks for an action are run is determined by each hook’s order attribute. Hooks with a lower value for order will run before hooks with a higher value. To set the relative order in which the hooks for an action will be run, set each hook’s order attribute.

Format:

```
Qmgr: set hook <hook name> order=<ordering>
```

<ordering> is an integer. Hooks with lower values for <ordering> run before those with higher values; a hook with order=1 runs before a hook with order=2.

Valid values for hook ordering are between 1 and 1000.

The order in which hooks for unrelated actions execute is undefined. For example, there are two queuejob hooks, Hook1 and Hook2, and userA submits jobA and userB submits jobB. While Hook1 always runs before Hook2 for the same job, the order of execution is undefined for different jobs. So the order could be:

- Hook1 (jobB)
- Hook1 (jobA)
- Hook2 (jobA)
- Hook2 (jobB)

6.7.4.1 Example of Setting Relative Order of Hook Execution

To set hookA to run first and hookB to run second:

```
Qmgr: set hook hookA order=2
Qmgr: set hook hookB order=5
```
6.7.5 Setting Hook Timeout

You may wish to specify how long PBS should wait for a hook to run. Execution for each hook times out after the number of seconds specified in the hook’s alarm attribute. If the hook does not run in the specified time, PBS aborts the hook and rejects the hook’s action.

Format:

\[ \texttt{Qmgr: set hook } <\text{hook name}> \text{ alarm=<timeout>} \]

<timeout> is the number of seconds PBS will allow the hook to run.

When a hook timeout is triggered, the hook script gets a Python Keyboard-Interrupt from the PBS server. The server logs show:

06/17/2008 17:57:16;0001;Server@fest;Svr;Server@fest;PBS server internal error (15011) in Python script received a KeyboardInterrupt, <type 'exceptions.KeyboardInterrupt'>

6.7.5.1 Example of Setting Hook Timeout

To set the number of seconds that PBS will wait for hook hook1 to execute before aborting the hook and reject the action:

\[ \texttt{Qmgr: set hook hook1 alarm=20} \]

6.7.6 Setting and Unsetting Hook Attributes

To set a hook attribute:

Format:

\[ \texttt{Qmgr: set hook } <\text{hook name}> <\text{attribute}> = <\text{value}> \]

To unset a hook attribute:

\[ \texttt{Qmgr: unset hook } <\text{hook name}> <\text{attribute}> \]
When you unset a hook’s attribute, that attribute reverts to its default value. For a list of hook attributes, see section 6.7.1 “Hook Attributes” on page 479 or “Hook Attributes” on page 453 of the PBS Professional Reference Guide.

### 6.7.6.1 Example of Unsetting Hook Attributes

To unset hook1’s `alarm` attribute, causing its value to revert to its default value:

```
Qmgr: unset hook hook1 alarm
```

This causes hook1’s `alarm` to revert to the default of *30 seconds*.

### 6.8 Viewing Hook Information

#### 6.8.1 Listing Hooks

To list one hook and its attributes on the current server:

```
Qmgr: list hook <hook name>
```

To list all hooks and their attributes on the current server:

```
Qmgr: list hook
```

#### 6.8.2 Viewing Hook Contents

To view the contents of a hook, export the hook’s contents:

```
export hook <hook_name> <content-type> <content-encoding> [<output_file>]
```
6.8.3 Printing Hook Creation Commands

To view the commands to create one hook:

```
Qmgr: print hook <hook name>
```

To view the commands to create all the hooks on the default server:

```
Qmgr: print hook
qmgr -c "print hook"
```

For example, to see the commands used to create hook1 and hook2:

```
# qmgr -c "print hook"
create hook hook1
import hook hook1 application/x-python base64 -
   cHJpbnQgImhlbGxvLCB3b3JsZCIK<blank line>
set hook hook1 event=movejob
set hook hook1 alarm=10
set hook hook1 order=5
create hook hook2
import hook hook2 application/x-python base64 -
   aservaJLSDFSESF<newline>
set hook hook2 event=queuejob
set hook hook2 alarm=15
set hook hook2 order=60
...
```

6.8.4 Re-creating Hooks

To re-create a hook, you feed qmgr hook descriptions back into qmgr. These hook descriptions are the same information that qmgr prints out. To print out the statements needed to recreate a hook, use the `print hook` or `print hook <hook name>` qmgr commands.

For example, to save information for hook1 and hook2:

```
# qmgr -c "print hook" > hookInfo
```
To re-create hook1 and hook2:

```
# qmgr < hookInfo
```

## 6.9 Interface to Hooks

The *pbs module* provides an interface to PBS and the hook environment. The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code. In order to use the *pbs* module, you must begin your Python code by importing the module. For example, in a script that modifies a job:

```
import pbs
pbs.server().job.comment="Modified this job"
```

### 6.9.1 What Hooks Can Access and Do

- Hooks can allow or disallow the action for which the hook is executing
- Hooks (except *runjob* hooks) can alter any arguments or parameters passed to the action, e.g.:
  - Job attributes and resources specified for job submission
  - Arguments to *qalter* or *qmove*
- Hooks can read the user attempting to execute the action
  - The Server, Scheduler, and MoMs are treated as users
- Hooks can read and modify job attributes normally accessible via *qstat*
- Hooks can read and modify the reservation attributes normally accessible via *pbs_rstat*, during reservation creation
- Hooks can read environment variables from the user's session
- Hooks can read Server attributes normally visible via *qstat -Bf*, only on the local server
- Hooks can read queue attributes normally visible via *qstat -Qf*, only
for queues on the local server

- Hooks can specify a message to be printed by the daemon
- Hooks can read vnode attributes and global resources normally visible via qmgr and pbsnodes
- Hooks can set the state vnode attribute; hooks cannot set other vnode attributes

6.9.2 What Hooks Cannot Access or Do

- Hooks cannot read or modify anything not presented in the PBS hook interface
  - Server configuration information other than what is available via qstat, meaning hooks cannot access or modify anything visible via qmgr, resourcedef, or pbs.conf
  - Scheduling information other than what is available via qstat, meaning hooks cannot access or modify anything visible via qmgr, sched_config, dedicated, holidays
  - Vnode information other than what is available via qstat, meaning hooks cannot access or modify anything visible via qmgr or pbsnodes

Hooks cannot do the following:

- Hooks do not have access to other servers besides the default server.
  - Hooks cannot change the destination server to a non-default server.
    Hooks can allow a job submission or a qmove to a non-default server, or can change the destination server to be the default server.
- Hooks cannot directly print to stdout or stderr or read from stdin.
- Movejob hooks do not run on pbs_rsub -Wqmove=<job ID>

Hooks should not do the following:

- Hooks should not edit configuration files directly, meaning hooks
should not edit the following:

PBS_HOME/sched_priv/sched_config  
PBS_HOME/sched_priv/fairshare  
PBS_HOME/sched_priv/dedicated  
PBS_HOME/sched_priv/holidays  
/etc/pbs.conf  

- Hooks should not execute PBS commands

### 6.9.3 Using Attributes in Hooks

The job, vnode, or reservation object’s attributes appear to the hook as they would be after the event, not before it.

For a **movejob** event, the only attribute that can be changed is the job’s destination queue. The job’s current queue is shown in the `pbs.event().src_queue` event attribute.

For a **modifyjob** event, the original job with all its attributes is shown in `pbs.event().job_o`. Any settable job attribute can be changed before the job runs. While the job is running, the only job attributes that can be changed are those that can be changed by an unprivileged user via the `qalter` command. See “qalter” on page 152 of the PBS Professional Reference Guide and “Job Attributes” on page 430 of the PBS Professional Reference Guide.

For a **runjob** event, the only job attributes that can be changed are the job’s `Hold_Types` and `Execution_Time` attributes, and only if the job is rejected.

For a **runjob** event, the only vnode attribute that can be changed is the `state` attribute. The only hook that can change this attribute is a **runjob** hook.

For a **queuejob** event, any settable job attribute can be set. See “Job Attributes” on page 430 of the PBS Professional Reference Guide.

For a **resvsub** event, any settable reservation attribute that can be specified via `pbs_rsub` can be set.
6.9.4 Printing And Logging Messages

Hooks can log a custom string in the Server’s log, at message log event class `pbs.LOG_DEBUG (0x0004)`. This is done using the `pbs.log_jobmsg(job ID, message)` facility. See section 6.9.14.2.2 “Method Taking Job ID and Message as Arguments” on page 541.

Hooks can specify a message for use when the corresponding action is rejected. This message is printed to `stderr` by the command that triggered the event, and is printed in the daemon’s log. This is done using the `pbs.logmsg()` function. See section 6.9.14.2.1 “Method Taking Log Level and Message as Arguments” on page 541 for information on how to specify a rejection message.

Hooks cannot directly print to `stdout` or `stderr`, or read from `stdin`. See section 6.12.5.1 “Avoid Hook File I/O” on page 569, and section 6.11.3.6 “Hooks Attempting I/O” on page 561.

6.9.5 Resources

Hooks can read Server, queue, vnode, reservation, or job resources. Hooks (except `runjob` hooks) can modify:

- The resources requested by a job
- The resource arguments to `pbs_rsub`

The built-in PBS resources are represented as Python dictionaries, where the resource names are the dictionary keys. These resources are listed in “Resources” on page 347 of the PBS Professional Reference Guide. You reference a resource through either the Server or the event that triggered the hook, for example:

```python
pbs.server().resources_available["<resource name>"]
pbs.event().job.Resource_List["<resource name>”]
```

The resource name must be in quotes.

Example: Get the number of CPUs:

```python
ncpus=pbs.event().job.Resource_List["ncpus"]
```

An instance $R$ of a job resource can be set as follows:
\[ R["<resource name>"] = <resource value> \]

For example:

```python
pbs.event().job.Resource_List["mem"] = 8gb
```

### 6.9.6 PBS Interface Objects

The PBS interface contains different kinds of objects:

- **Constant objects** to represent event types, states, log event classes, queue types, and exceptions. These cannot be changed.
- **Objects to represent PBS entities**, e.g. jobs, servers, queues, vnodes, reservations, events, log messages, etc.
- **Objects to represent attributes**:
  - Objects to represent job attributes, e.g. checkpoint, dependency, hold types, mail points.
  - Objects to represent vnode attributes.
  - Objects to represent other attributes, e.g. state counters, etc.
  - Objects to represent values for attributes, e.g. ACLs, mail lists, group lists, etc.
- **Other objects**:
  - Objects to represent job `select` and `place` specifications.
  - Objects to represent PBS sizes and time intervals.
  - Objects to represent arguments to PBS commands, PBS version information, etc.

#### 6.9.6.1 Objects Requiring a Formatted Input String

Instantiation of some of the interface objects requires a formatted input string. You can operate on these objects only as if they are strings.

Instantiation of these objects requires a formatted input string. A `repr()` of the object's instance returns its full string representation. This representation can be manipulated using the built-in methods for Python `str`.
## 6.9.6.2 Table of PBS Interface Objects

The following table lists all of the PBS objects in alphabetical order. Each of these objects is described in detail later in the chapter.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.acl</td>
<td>Represents a PBS ACL type. See “ACL Objects” on page 539.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.args</td>
<td>Represents a space-separated list of PBS arguments to commands like <code>qsub</code>, <code>qdel</code>. See “Argument Objects” on page 540.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.BadAttributeValueError</td>
<td>Raised when setting the attribute value of a pbs.* object and the value given is invalid. See “Exceptions” on page 503.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.BadAttributeValueTypeError</td>
<td>Raised when setting the attribute value of a pbs.* object and the value type is invalid. See “Exceptions” on page 503.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.BadAttributeValueError</td>
<td>Raised when setting the resource value of a pbs.* object and the value given is invalid. See “Exceptions” on page 503.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.BadAttributeValueTypeError</td>
<td>Raised when setting the resource value of a pbs.* object and the value type is invalid. See “Exceptions” on page 503.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.checkpoint</td>
<td>Represents a job’s checkpoint attribute. See “Checkpoint Objects” on page 533.</td>
<td>Formatted input string</td>
</tr>
</tbody>
</table>
### Table 6-3: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.depend</td>
<td>Represents a job's dependency attribute. See “Dependency Objects” on page 533.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.duration</td>
<td>Represents a time interval. See “Duration Objects” on page 511.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.email_list</td>
<td>Represents the set of users to whom mail may be sent. Example: Job's Mail_Users attribute. See “Email List Objects” on page 539.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.event</td>
<td>Represents a PBS event. See “Event Objects” on page 512.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.EventIncompatibleError</td>
<td>Raised when referencing a nonexistent attribute in pbs.event(). See “Exceptions” on page 503.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.exec_host</td>
<td>Represents a job’s exec_host attribute. See “exec_host Objects” on page 534.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.exec_vnode</td>
<td>Represents a job’s exec_vnode attribute. See “exec_vnode Objects” on page 534.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.group_list</td>
<td>Represents a list of group names. See “Group List Objects” on page 539.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.hold_types</td>
<td>Represents the Hold_Types attribute of a job. See “Hold Type Objects” on page 536.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.job</td>
<td>Represents a PBS job. See “Job Objects” on page 523.</td>
<td>--</td>
</tr>
</tbody>
</table>
### Table 6-3: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.job_sort_formula</td>
<td>Represents the job_sort_formula server attribute. See “Job Sorting Formula Objects” on page 537.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.JOB_STATE_BEGIN</td>
<td>Job arrays only. Job array has started. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_EXITING</td>
<td>Job is exiting after having run. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_EXPIRED</td>
<td>Subjobs only. Subjob is finished (expired.) See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_FINISHED</td>
<td>Job is finished: job executed successfully, job was terminated while running, job execution failed, or job was deleted before execution. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_HELD</td>
<td>Job is held. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_MOVED</td>
<td>Job has been moved to another server. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_QUEUED</td>
<td>Job is queued, eligible to run or be routed. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_RUNNING</td>
<td>Job is running. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_SUSPEND</td>
<td>Job is suspended by server. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
### Table 6-3: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.JOB_STATE_SUSPEND_USERACTIVE</td>
<td>Job is suspended due to workstation becoming busy. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_TRANSIT</td>
<td>Job is in transit. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.JOB_STATE_WAITING</td>
<td>Job is waiting for its requested execution time to be reached, or the job’s stagein request has failed. See “Job State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.join_path</td>
<td>Represents the job’s Join_Path attribute. See “Join Path Objects” on page 537.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.keep_files</td>
<td>Represents the Keep_Files job attribute. See “Keep Files Objects” on page 537.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.license_count</td>
<td>Represents a set of licensing-related counters. See “License Count Objects” on page 538.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.LOG_DEBUG</td>
<td>Log event class 0x0004. See “Message Log Event Class Objects” on page 505.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.LOG_ERROR</td>
<td>Log event class 0x0004. See “Message Log Event Class Objects” on page 505.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.LOG_WARNING</td>
<td>Log event class 0x0004. See “Message Log Event Class Objects” on page 505.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.mail_points</td>
<td>Represents the Mail_Points attribute of a job. See “Mail Points Objects” on page 539.</td>
<td>Formatted input string</td>
</tr>
</tbody>
</table>
## Hooks

### Table 6-3: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.MODIFYJOB</td>
<td>The <code>modifyjob</code> hook event type. Triggered by <code>qalter</code> or <code>pbs_alterjob()</code> API call. See “Event Types” on page 513.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.MOVEJOB</td>
<td>The <code>movejob</code> hook event type. Triggered by <code>qmove</code> or <code>pbs_movejob()</code> API call. See “Event Types” on page 513.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_BUSY</td>
<td>Represents <code>busy</code> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_DEFAULT_EXCL</td>
<td>Represents <code>default_excl</code> sharing vnode attribute value. See section 6.9.7.9 “Vnode Sharing Objects” on page 510.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_DEFAULT_SHARED</td>
<td>Represents <code>default_shared</code> sharing vnode attribute value. See section 6.9.7.9 “Vnode Sharing Objects” on page 510.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_DOWN</td>
<td>Represents <code>down</code> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_FORCE_EXCL</td>
<td>Represents <code>force_excl</code> sharing vnode attribute value. See section 6.9.7.9 “Vnode Sharing Objects” on page 510.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_FREE</td>
<td>Represents <code>free</code> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
Table 6-3: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_GLOBUS</td>
<td>Represents <em>globus</em> vnode ntype. See section 6.9.7.8 “Vnode Type Objects” on page 510</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_IGNORE_EXCL</td>
<td>Represents <em>ignore_excl</em> sharing vnode attribute value. See section 6.9.7.9 “Vnode Sharing Objects” on page 510.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_JOBBUSY</td>
<td>Represents <em>job-busy</em> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_JOB_EXCLUSIVE</td>
<td>Represents <em>job-exclusive</em> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_OFFLINE</td>
<td>Represents <em>offline</em> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_PBS</td>
<td>Represents <em>pbs</em> vnode ntype. See section 6.9.7.8 “Vnode Type Objects” on page 510.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_PROV</td>
<td>Represents <em>provisioning</em> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_STALE</td>
<td>Represents <em>stale</em> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>Object</td>
<td>Description</td>
<td>Note</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>pbs.ND_STATE_UNKNOWN</td>
<td>Represents <em>state-unknown, down</em> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.ND_WAIT_PROV</td>
<td>Represents <em>wait-provisioning</em> vnode state. See section 6.9.7.7 “Vnode State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.node_group_key</td>
<td>Represents the <em>node_group_key</em> attribute. See “Node Group Key Objects” on page 539.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.path_list</td>
<td>Represents a list of pathnames. See “Path List Objects” on page 540.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.place</td>
<td>Represents the <em>place</em> specification when submitting a job. See “Place Objects” on page 532.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.QTYPE_EXECUTION</td>
<td>Represents <em>execution</em> value for <em>queue_type</em> queue attribute. See “Queue Type Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.QTYPE_ROUTE</td>
<td>Represents <em>route</em> value for <em>queue_type</em> queue attribute. See “Queue Type Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.queue</td>
<td>Represents a PBS queue. See “Queue Objects” on page 522.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.QUEUEJOB</td>
<td>The <em>queuejob</em> hook event type. Triggered by <em>qsub</em> or <em>pbs_submit()</em> API call.</td>
<td>Constant</td>
</tr>
</tbody>
</table>

Table 6-3: PBS Interface Objects
Table 6-3: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.range</td>
<td>Represents a range of numbers referring to job array indices. See “Range Objects” on page 510.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.resv</td>
<td>Represents a PBS reservation. See “Reservation Objects” on page 526.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.RESVSUB</td>
<td>The resvsub hook event type. Triggered by pbs_rsub or pbs_submitresv() API call. See “Event Types” on page 513.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
| pbs.RESV_STATE_BEING_DELETED | The reservation state 
RESV_BEING_DELETED. See “Reservation State Objects” on page 508. | Constant                  |
| pbs.RESV_STATE_CONFIRMED | The reservation state 
RESV_CONFIRMED. See “Reservation State Objects” on page 508. | Constant                  |
| pbs.RESV_STATE_DEGRADED | The reservation state 
RESV_DEGRADED. See “Reservation State Objects” on page 508. | Constant                  |
| pbs.RESV_STATE_DELETED  | The reservation state 
RESV_DELETED. See “Reservation State Objects” on page 508. | Constant                  |
| pbs.RESV_STATE_DELETING_JOBS | The reservation state 
RESV_DELETING_JOBS. See “Reservation State Objects” on page 508. | Constant                  |
# Hooks

## Table 6-3: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.RESV_STATE_FINISHED</td>
<td>The reservation state <code>RESV_FINISHED</code>. See “Reservation State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.RESV_STATE_NONE</td>
<td>The reservation state <code>RESV_NONE</code>. See “Reservation State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.RESV_STATE_RUNNING</td>
<td>The reservation state <code>RESV_RUNNING</code>. See “Reservation State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.RESV_STATE_TIME_TO_RUN</td>
<td>The reservation state <code>RESV_TIME_TO_RUN</code>. See “Reservation State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.RESV_STATE_UNCONFIRMED</td>
<td>The reservation state <code>RESV_UNCONFIRMED</code>. See “Reservation State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.RESV_STATE_WAIT</td>
<td>The reservation state <code>RESV_WAIT</code>. See “Reservation State Objects” on page 508.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.route_destinations</td>
<td>Represents <code>route_destinations</code> queue attribute. See “Routing Destination Objects” on page 538.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.RUNJOB</td>
<td>The <code>runjob</code> hook event type. Triggered by <code>qrun</code> or <code>pbs_runjob()</code> API call. See “Event Types” on page 513.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
Table 6-3: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.select</td>
<td>Represents the <code>select</code> specification when submitting a job. See “Select Objects” on page 531.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.server</td>
<td>Represents the local PBS server. See “Server Objects” on page 519.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.size</td>
<td>Represents a PBS size type. See “Size Objects” on page 511.</td>
<td>--</td>
</tr>
<tr>
<td>pbs.software</td>
<td>Represents a site-dependent software specification resource. See “Software Resource Objects” on page 542.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.staging_list</td>
<td>Represents a list of file stagein or stageout parameters. See “Staging List Objects” on page 537.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.state_count</td>
<td>Represents a set of job-related state counters. See “State Count Objects” on page 538.</td>
<td>Formatted input string</td>
</tr>
<tr>
<td>pbs.SV_STATE_ACTIVE</td>
<td>Server state is <em>Scheduling</em>. See “PBS Server State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.SV_STATE_HOT</td>
<td>Server state is <em>Hot_Start</em>. See “PBS Server State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.SV_STATE_IDLE</td>
<td>Server state is <em>Idle</em>. See “PBS Server State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.SV_STATE_SHUTDEL</td>
<td>Server state is <em>Terminating</em>, Delayed. See “PBS Server State Objects” on page 506.</td>
<td>Constant</td>
</tr>
</tbody>
</table>
### Table 6-3: PBS Interface Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.SV_STATE_SHUTIMM</td>
<td>Server state is <em>Terminating</em>. See “PBS Server State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.SV_STATE_SHUTSIG</td>
<td>Server state is <em>Terminating</em>. See “PBS Server State Objects” on page 506.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.UnsetAttributeNameError</td>
<td>Raised when referencing a non-existent attribute name of a pbs.* object. See “Exceptions” on page 503.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.UnsetResourceNameError</td>
<td>Raised when referencing a non-existent resource name of a pbs.* object. See “Exceptions” on page 503.</td>
<td>Constant</td>
</tr>
<tr>
<td>pbs.user_list</td>
<td>Represents a list of user names. See “User List Objects” on page 540.</td>
<td>Formatted</td>
</tr>
<tr>
<td>pbs.vchunk</td>
<td>Represents a chunk. See section 6.9.10.3 “Chunk Objects” on page 532.</td>
<td></td>
</tr>
<tr>
<td>pbs.version</td>
<td>Represents version information for PBS. See “PBS Version Objects” on page 541.</td>
<td>Formatted</td>
</tr>
<tr>
<td>pbs.vnode</td>
<td>Represents a PBS vnode. See section 6.9.9.6 “Vnode Objects” on page 528.</td>
<td></td>
</tr>
<tr>
<td>SystemExit</td>
<td>Raised when accepting or rejecting an action. See “Exceptions” on page 503.</td>
<td>--</td>
</tr>
</tbody>
</table>
6.9.6.3 Exceptions

The following exceptions may be raised when using the `pbs.*` objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pbs.BadAttributeValueError</code></td>
<td>Raised when setting attribute value of a <code>pbs.*</code> object to an invalid value.</td>
</tr>
<tr>
<td><code>pbs.BadAttributeValueTypeError</code></td>
<td>Raised when setting attribute value of a <code>pbs.*</code> object to an invalid type.</td>
</tr>
<tr>
<td><code>pbs.BadResourceValueError</code></td>
<td>Raised when setting resource value of a <code>pbs.*</code> object to an invalid value.</td>
</tr>
<tr>
<td><code>pbs.BadResourceValueTypeError</code></td>
<td>Raised when setting resource value of a <code>pbs.*</code> object to an invalid type.</td>
</tr>
<tr>
<td><code>pbs.EventIncompatibleError</code></td>
<td>Raised when referencing a non-existent attribute in <code>pbs.event</code>. Example: calling <code>ev.resv</code> for <code>ev.type of pbs.QUEUEJOB</code></td>
</tr>
<tr>
<td><code>pbs.UnsetAttributeNameError</code></td>
<td>Raised when referencing a non-existent attribute name of a <code>pbs.*</code> object.</td>
</tr>
<tr>
<td><code>pbs.UnsetResourceNameError</code></td>
<td>Raised when referencing a non-existent resource name of a <code>pbs.*</code> object.</td>
</tr>
<tr>
<td><code>SystemExit</code></td>
<td>1. Raised when <code>ev.reject()</code> terminates hook execution.</td>
</tr>
<tr>
<td></td>
<td>2. Raised when <code>ev.accept()</code> terminates hook execution.</td>
</tr>
</tbody>
</table>

6.9.6.3.1 Unhandled Exceptions

If a hook encounters an unhandled exception:

- PBS rejects the corresponding action. The command that initiates the
action results in the following message in stderr:

"<command_name>: request rejected as filter hook
<hook_name> encountered an exception. Please inform Admin"

• The following message appears in the appropriate PBS daemon log, logged under 0x0100 event class:

"<request type> hook <hook_name> encountered an exception, request rejected"

• The job is left unmodified.

The following statements will cause an unhandled exception if they appear in a hook script as is:

• ZeroDivisionError exception raised:
  
  ```python
  val = 5/0
  ```

• BadAttributeValueError exception raised; pbs.hold_types and strings don't mix:
  
  ```python
  pbs.event().Hold_Types = "z"
  ```

• EventIncompatibleError exception raised for the following runjob event; runjob event has job attribute, not resv attribute:
  
  ```python
  r = pbs.event().resv
  ```

### 6.9.7 Constant Objects

Constant objects are used to represent PBS elements such as event types, job, server, reservation, and vnode states, log event classes, queue and vnode types, and exceptions. These objects cannot be modified. When the PBS module is imported, the constant objects are imported.

Example of use:

```python
if pbs.event().type == pbs.QUEUEJOB:
```
6.9.7.1 Event Type Objects

This table lists the constant objects that represent event types. See section 6.9.9.1 “Event Objects” on page 512.

Table 6-5: Event Type Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Event Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.QUEUEJOB</td>
<td>queuejob</td>
<td>qsub or pbs_submit()</td>
</tr>
<tr>
<td>pbs.MODIFYJOB</td>
<td>modifyjob</td>
<td>qalter or pbs_alterjob()</td>
</tr>
<tr>
<td>pbs.RESVSUB</td>
<td>resvsub</td>
<td>pbs_rsub or pbs_submitresv()</td>
</tr>
<tr>
<td>pbs.MOVEJOB</td>
<td>movejob</td>
<td>qmove or pbs_movejob()</td>
</tr>
<tr>
<td>pbs.RUNJOB</td>
<td>runjob</td>
<td>qrun or pbs_runjob()</td>
</tr>
</tbody>
</table>

6.9.7.2 Message Log Event Class Objects

You can use the following objects to indicate log event class when placing messages in the server logs. Log levels are shown in hexadecimal. The messages are logged at the "Administrative Events" (0x0004) event class.

Table 6-6: Message Log Level Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Log Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.LOG_WARNING</td>
<td>0x0004</td>
</tr>
<tr>
<td>pbs.LOG_ERROR</td>
<td>0x0004</td>
</tr>
<tr>
<td>pbs.LOG_DEBUG</td>
<td>0x0004</td>
</tr>
</tbody>
</table>
6.9.7.3 Queue Type Objects

Table 6-7: Queue Type Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Queue Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.QTYPE_EXECUTION</td>
<td>Execution</td>
</tr>
<tr>
<td>pbs.QTYPE_ROUTE</td>
<td>Route</td>
</tr>
</tbody>
</table>

6.9.7.4 PBS Server State Objects

Table 6-8: Server State Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.SV_STATE_IDLE</td>
<td>Idle</td>
</tr>
<tr>
<td>pbs.SV_STATE_ACTIVE</td>
<td>Scheduling</td>
</tr>
<tr>
<td>pbs.SV_STATE_HOT</td>
<td>Hot_Start</td>
</tr>
<tr>
<td>pbs.SV_STATE_SHUTDEL</td>
<td>Terminating, Delayed</td>
</tr>
<tr>
<td>pbs.SV_STATE_SHUTIMM</td>
<td>Terminating</td>
</tr>
<tr>
<td>pbs.SV_STATE_SHUTSIG</td>
<td>Terminating</td>
</tr>
</tbody>
</table>

6.9.7.5 Job State Objects

See “qstat” on page 228 of the PBS Professional Reference Guide.

Table 6-9: Job State Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.JOB_STATE_BEGUN</td>
<td>B</td>
<td>Job arrays only: job array has started</td>
</tr>
<tr>
<td>pbs.JOB_STATE_EXITING</td>
<td>E</td>
<td>Job is exiting after having run</td>
</tr>
<tr>
<td>pbs.JOB_STATE_EXPIRED</td>
<td>X</td>
<td>Subjobs only; subjob is finished (expired.)</td>
</tr>
</tbody>
</table>
### Hooks

#### Table 6-9: Job State Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.JOB_STATE_FINISHED</td>
<td>F</td>
<td>Job is finished: job executed successfully, job was terminated while running, job execution failed, or job was deleted before execution</td>
</tr>
<tr>
<td>pbs.JOB_STATE_HELD</td>
<td>H</td>
<td>Job is held.</td>
</tr>
<tr>
<td>pbs.JOB_STATE_MOVED</td>
<td>M</td>
<td>Job has been moved to another server</td>
</tr>
<tr>
<td>pbs.JOB_STATE_QUEUED</td>
<td>Q</td>
<td>Job is queued, eligible to run or be routed</td>
</tr>
<tr>
<td>pbs.JOB_STATE_RUNNING</td>
<td>R</td>
<td>Job is running</td>
</tr>
<tr>
<td>pbs.JOB_STATE_SUSPEND</td>
<td>S</td>
<td>Job is suspended by PBS so that a higher-priority job can run.</td>
</tr>
<tr>
<td>pbs.JOB_STATE_SUSPEND_USERACTIVE</td>
<td>U</td>
<td>Job is suspended due to workstation becoming busy</td>
</tr>
<tr>
<td>pbs.JOB_STATE_TRANSIT</td>
<td>T</td>
<td>Job is in transition (being moved to a new location)</td>
</tr>
<tr>
<td>pbs.JOB_STATE_WAITING</td>
<td>W</td>
<td>Job is waiting for its requested execution time to be reached, or the job’s specified stage in request has failed for some reason.</td>
</tr>
</tbody>
</table>
6.9.7.6 Reservation State Objects

Table 6-10: Reservation State Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.RESV_STATE_NONE</td>
<td>RESV_NONE</td>
</tr>
<tr>
<td>pbs.RESV_STATE_UNCONFIRMED</td>
<td>RESV_UNCONFIRMED</td>
</tr>
<tr>
<td>pbs.RESV_STATE_CONFIRMED</td>
<td>RESV_CONFIRMED</td>
</tr>
<tr>
<td>pbs.RESV_STATE_WAIT</td>
<td>RESV_WAIT</td>
</tr>
<tr>
<td>pbs.RESV_STATE_TIME_TO_RUN</td>
<td>RESV_TIME_TO_Run</td>
</tr>
<tr>
<td>pbs.RESV_STATE_RUNNING</td>
<td>RESV_RUNNING</td>
</tr>
<tr>
<td>pbs.RESV_STATE_FINISHED</td>
<td>RESV_FINISHED</td>
</tr>
<tr>
<td>pbs.RESV_STATE_BEING_DELETED</td>
<td>RESV_BEING_DELETED</td>
</tr>
<tr>
<td>pbs.RESV_STATE_DELETED</td>
<td>RESV_DELETED</td>
</tr>
<tr>
<td>pbs.RESV_STATE_DELETING_JOBS</td>
<td>RESV_DELETING_JOBS</td>
</tr>
<tr>
<td>pbs.RESV_STATE_DEGRADED</td>
<td>RESV_DEGRADED</td>
</tr>
</tbody>
</table>

6.9.7.7 Vnode State Objects

Table 6-11: Vnode State Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_BUSY</td>
<td>Represents busy vnode state</td>
</tr>
<tr>
<td>pbs.ND_DOWN</td>
<td>Represents down vnode state</td>
</tr>
<tr>
<td>pbs.ND_FREE</td>
<td>Represents free vnode state</td>
</tr>
<tr>
<td>pbs.ND_JOBBUSY</td>
<td>Represents job-busy vnode state</td>
</tr>
<tr>
<td>pbs.ND_JOB_EXCLUSIVE</td>
<td>Represents job-exclusive vnode state</td>
</tr>
</tbody>
</table>
Table 6-11: Vnode State Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_OFFLINE</td>
<td>Represents offline vnode state</td>
</tr>
<tr>
<td>pbs.ND_PROV</td>
<td>Represents provisioning vnode state</td>
</tr>
<tr>
<td>pbs.ND_STALE</td>
<td>Represents stale vnode state</td>
</tr>
<tr>
<td>pbs.ND_STATE_UNKNOWN</td>
<td>Represents state-unknown, down vnode state</td>
</tr>
<tr>
<td>pbs.ND_WAIT_PROV</td>
<td>Represents wait-provisioning vnode state</td>
</tr>
</tbody>
</table>
### 6.9.7.8 Vnode Type Objects

**Table 6-12: Vnode Type Objects**

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_GLOBUS</td>
<td>Represents globus vnode ntype value</td>
</tr>
<tr>
<td>pbs.ND_PBS</td>
<td>Represents pbs vnode ntype value</td>
</tr>
</tbody>
</table>

### 6.9.7.9 Vnode Sharing Objects

**Table 6-13: Vnode Sharing Objects**

<table>
<thead>
<tr>
<th>Object</th>
<th>Sharing Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbs.ND_DEFAULT_EXCL</td>
<td>Represents default_excl vnode sharing attribute value</td>
</tr>
<tr>
<td>pbs.ND_DEFAULT_SHARED</td>
<td>Represents default_shared vnode sharing attribute value</td>
</tr>
<tr>
<td>pbs.ND_FORCE_EXCL</td>
<td>Represents force_excl vnode sharing attribute value</td>
</tr>
<tr>
<td>pbs.ND_IGNORE_EXCL</td>
<td>Represents ignore_excl vnode sharing attribute value</td>
</tr>
</tbody>
</table>

### 6.9.8 Measurement Objects

#### 6.9.8.1 Range Objects

Represents a range of values.

```bash
pbs.range("<start>-<stop>:<end>")
```

Creates a PBS object representing a range of values.

Example:

```bash
pbs.range("1-30:3")
```

Instantiation of these objects requires a formatted input string.
6.9.8.2 Size Objects

This represents a PBS size type. Size objects can be specified using either an integer or a string.

Integer specification:

\[ pbs.size(int) \]

Creates a PBS size object out of the given integer value, storing the value as the number of bytes.

String specification:

\[ pbs.size(“int[suffix]”) \]

Creates a PBS size object out of the given string specification. The suffix must be a multiplier defined in the table shown in “Size” on page 460 of the PBS Professional Reference Guide. The size of a word is the word size on the execution host.

To operate on \( pbs.size \) instances, use the “+” and “-” operators.

To compare \( pbs.size \) instances, use the “==”, “!=”, “>”, “<”, “>=”, and “<=” operators.

Example 1: The sizes are normalized to the lower of the 2 suffixes. In this case, “10gb” becomes “10240mb” and added to “10mb”

\[ sz = pbs.size(“10gb”) \]

\[ sz = sz + 10mb \]

10250mb

Example 2: The following returns True as \( sz \) is greater than 100 bytes.

\[ if \ sz > 100: \]

\[ gt100 = True \]

6.9.8.3 Duration Objects

Represents an interval or elapsed time in number of seconds. Duration objects can be specified using either a time or an integer.

Time specification:

\[ pbs.duration(“[[hours:]minutes:]seconds[.milliseconds]”) \]
Chapter 6  

Hooks

Creates a **duration** instance, returning the equivalent number of seconds from the given time string.

Integer specification:

```python
pbs.duration(int)
```

Creates a **duration** instance as some integer number of seconds.

A `pbs.duration` instance can be operated on by any of the Python `int` functions. When performing arithmetic operations on a `pbs.duration` type, ensure the resulting value is a `pbs.duration()` type, before assigning to a job attribute that expects such a type.

**6.9.8.3.1 Example of Duration**

The following will work:

```python
j.Resource_List['cput'] = pbs.duration(300 + d1)  # safe
```

**6.9.8.3.2 Duration Caveats**

The following will **not** work, since Python evaluates from left to right, and returns result as the type at left (`int`):

```python
d1 = pbs.duration(30)
j.Resource_List['cput'] = 300 + d1
```

**6.9.9 Entity Objects**

Entity objects represent the following PBS entities: event, Server, Scheduler, queue, job.

**6.9.9.1 Event Objects**

The event represented by this object is the one that triggers the hook. This event object can be passed to the hook script for use by the script.

```python
pbs.event()
```

This creates a PBS event object.

The event object is used to pass event information to the hook script.
6.9.9.1.1 Event Types

The type object that is returned by pbs.event().type.

This table shows the event types, their triggers, and when they are triggered:

**Table 6-14: Event Type Objects**

<table>
<thead>
<tr>
<th>Type</th>
<th>Trigger</th>
<th>When Triggered</th>
</tr>
</thead>
<tbody>
<tr>
<td>queuejob</td>
<td>Triggered by qsub and the pbs_submit() API call. Not triggered by requeueing a job (qrerun) or on node_fail_requeue, when a job is discarded by the MOM because the execution host went down.</td>
<td>A queuejob hook is executed after all processing of qsub input, and just before the job is queued.</td>
</tr>
<tr>
<td>modifyjob</td>
<td>Triggered by qalter and the pbs_alterjob() API call.</td>
<td>A modifyjob hook is executed after all processing of qalter input, and just before the job's attributes are modified.</td>
</tr>
<tr>
<td>resvsub</td>
<td>Triggered by pbs_rsub and the pbs_submitresv() API call.</td>
<td>A resvsub hook is executed after all processing of pbs_rsub input, and just before a reservation is created.</td>
</tr>
<tr>
<td>movejob</td>
<td>Triggered by qmove and the pbs_movejob() API call. Not triggered by pbs_rsub -Wqmove=&lt;job ID&gt;.</td>
<td>A movejob hook is executed after qmove arguments are processed, but before a job is moved from one queue to another.</td>
</tr>
<tr>
<td>runjob</td>
<td>Triggered by qrun and the pbs_runjob() API call.</td>
<td>A runjob hook is executed just before a job is placed into execution.</td>
</tr>
</tbody>
</table>
6.9.9.1.2 Event Object Attributes

Event objects have different attributes depending on the type of event. The attributes for each type are described in the following sections.

Given:

```python
ev = pbs.event()
```

Event objects all have the following attributes:

- **ev.type**
  - The event type. Valid values: one of the PBS event type constants listed in section 6.9.9.1.1 “Event Types” on page 513.
  - Type: constant event object

- **ev.hook_name**
  - A string constant containing the name of the hook being executed.

- **ev.requestor**
  - The requestor of the event. Values returned are in the form of Python strings. PBS daemons can request actions; in this case, the requestor attribute shows "PBS_Server", "Scheduler", or "pbs_mom". If the requestor is root, the attribute shows “root”. For Windows systems, if the requestor is the administrator, the attribute will show the account name of the administrator.

- **ev.requestor_host**
  - The name of the host from which the event was requested.

6.9.9.1.3 queuejob Events

In a queuejob event, the event’s job object attributes are as they would be if the job were to be successfully submitted.

Hooks cannot modify the server specified in a queuejob event.

For events whose ev.type is pbs.QUEUEJOB, the following attribute exists:
ev.job
A `pbs.job` object with the attributes and resources specified at submission for the job being queued. See section 6.9.9.4 "Job Objects" on page 523. Some job attributes are read-only, as described in “Job Attributes” on page 430 of the PBS Professional Reference Guide, and cannot be reset.

### 6.9.9.1.4 `modifyjob` Events

For `modifyjob` events, the job object’s attributes appear to the hook as they would be after the event, not before it.

For a `modifyjob` event, since many attributes can be changed, the original job with all its attributes is shown in `pbs.event().job_o`.

Hooks cannot modify the server specified in a `modifyjob` event.

For events whose `ev.type` is `pbs.MODIFYJOB`, the following attributes exist:

- **ev.job**
  A `pbs.job` object containing the attributes and resources specified for the job being modified. See section 6.9.9.4 "Job Objects" on page 523. Some job attributes are read-only, as described in “Job Attributes” on page 430 of the PBS Professional Reference Guide, and cannot be reset.

- **ev.job_o**
  A `pbs.job` object representing the original job, before the job was modified via `qalter`. See section 6.9.9.4 “Job Objects” on page 523.

### 6.9.9.1.5 Reservation (`resvsub`) Events

The only time that a reservation can be modified is during its creation. For `reservation` events, the reservation object’s attributes appear to the hook as they would be after the event, not before it.

For events whose `ev.type` is `pbs.RESVSUB`, the following attribute exists:

- **ev.resv**
  A `pbs.resv` object containing the attributes and resources specified for the reservation being requested. Some reserva-
Hook attributes are read-only, as described in “Reservation Attributes” on page 393 of the PBS Professional Reference Guide, and cannot be reset.

6.9.9.1.6 movejob Events

For movejob events, the job object’s attributes appear to the hook as they would be after the event, not before it.

The only attribute that can be changed is the job’s queue. The job’s current queue is shown in the pbs.event().src_queue event attribute.

Hooks cannot modify the server specified in a movejob event.

If ev.type is pbs.MOVEJOB, the following attributes exist:

- **ev.job**
  A pbs.job object representing the job being moved. See section 6.9.9.4 “Job Objects” on page 523.
  Note that ev.job.queue refers to the destination queue, not the current queue.

- **ev.src_queue**
  The pbs.queue object representing the original queue where ev.job came from.

The server runs its movejob hooks when the following happens:

- This server is the furnishing server when peer scheduling a job
- A job is moved from this server to another server via the qmove command
- A job is moved between two queues on this server

6.9.9.1.7 Runjob Events

A runjob event occurs when one of the following happens:

- The administrator uses the qrun command
- The scheduler chooses to run a job and calls pbs_runjob()

A runjob hook can modify a job or a vnode.
The `pbs.event().job.exec_vnode` object contains the vnodes specification (i.e. `<vnodeA>++<vnodeB>++...`) chosen by the scheduler or by the Administrator in a `qrun -H <vnode specification>` call.

If `ev.type` is `pbs.RUNJOB`, the following attributes exist:

```plaintext
ev.job
   A pbs.job object representing the job being run.
```

### 6.9.9.1.8 Modifying Jobs

A `runjob` hook can modify a job only if the hook rejects the job, and the hook can modify only the following:

- The job’s `Hold_Types` attribute
- The job’s `Execution_Time` attribute

### 6.9.9.1.9 Modifying Vnodes

A `runjob` hook can modify a vnode only if the hook rejects the event, and the hook can modify only the following:

- The vnode’s `state` attribute

### 6.9.9.1.10 Event Object Methods

```plaintext
ev.reject([ <msg> ])
   Terminates hook execution and instructs PBS to not perform the associated action. If the `<msg>` argument is given, it will be shown in the appropriate PBS daemon log, and in the `stderr` of the PBS command that caused this event to take place.

   ev.accept()
   Terminates hook execution and causes PBS to perform the associated event request action.
```
6.9.9.1.11 Event Object Method Caveats

ev.accept() terminates hook execution by throwing a SystemExit exception. So if hook content appears in a try...except clause that has no arguments to the except clause, always add the following to treat SystemExit as a normal occurrence:

```python
except SystemExit:
    pass
```

See section 6.12.4.1 “Treating SystemExit as a Normal Occurrence” on page 567.

6.9.9.1.12 Event Object Examples

Example 1: Inside a hook script, create a PBS event object:

```python
je = pbs.event()
```

Example 2: Get the event type:

```python
type = je.type
```

Example 3: Get the user who requested the event action:

```python
who = je.requestor
```

Example 4: Get the host where the request came from:

```python
host = je.requestor_host
```

Example 5: The event type is pbs.QUEUEJOB. Get the number of CPUs requested for the job being queued:

```python
j = je.job
res = j.Resource_List["ncpus"]
```

Example 6: Reset the number of CPUs requested by the job:

```python
j.Resource_List["ncpus"] = 1
```

Example 7: The event type is pbs.MOVEJOB. Get the request parameters:

```python
j = je.job
q = j.queue
```
Example 8: Accept an event request:

```python
je.accept()
```

Example 9: Reject an event request:

```python
je.reject("Can't set interactive attribute")
```

Example 10: Put a job into a wait state and requeue the job in 3600 seconds (1 hour):

```python
import time
...
pbs.event().job.Execution_Time = time.time() + 3600
```

Example 11: Put a hold on a job:

```python
j = pbs.event().job
j.Hold_Types = pbs.hold_types("u")
j.Hold_Types = pbs.hold_types("uo")
j.Hold_Types += pbs.hold_types("s")
or
pbs.event().job.Hold_Types =
pbs.hold_types("<hold_list>")
```

Example 12: Release a hold on a job:

```python
j.Hold_Types -= pbs.hold_types("un")
j.Hold_Types -= pbs.hold_types("sp")
j.Hold_Types -= pbs.hold_types("k")
or
pbs.event().job.Hold_Types -=
pbs.hold_types("<hold_list>")
```

### 6.9.9.2 Server Objects

This represents the PBS Server at which the triggering event is taking place, and at which the hook is executing. The only Server available to hooks is the local Server.

```python
pbs.server(['<name>'])
```
Creates an instance of a PBS server object, representing the local Server. If specified, <name> must be the name of the local Server. If <name> is not specified, the object represents the default server.

\texttt{pbs.server()} is used to pass server, queue, and job information to the hook script.

### 6.9.9.2.1 Server Object Attributes

An instance \texttt{s} of \texttt{pbs.server} has the following attributes:

\texttt{s.name}

The server hostname in Python \texttt{str} (string) format.

Example: myhost.mydomain.com

This attribute is read-only.

\texttt{s.<attribute name>}

The PBS server attribute named \texttt{<attribute name>}. Server attributes are listed in “Server Attributes” on page 365 of the PBS Professional Reference Guide.

### 6.9.9.2.2 Setting Server Object Attributes

You can set, but not unset, server object attributes. To set the value for the server attribute named \texttt{<attribute name>} to \texttt{<attribute value>}:

\texttt{s.<attribute name> = <attribute value>}

### 6.9.9.2.3 Server Object Methods

\texttt{s.queue( <queue_name> )}

Returns a queue object representing the queue named \texttt{<queue name>} that is managed by server \texttt{s}. See section 6.9.9.3 “Queue Objects” on page 522.

A value of \texttt{None} is returned if the queue named \texttt{<queue_name>} does not exist in server \texttt{s}.

\texttt{s.job('id')}  

Returns a \texttt{pbs.job} object for the job with ID \texttt{id}, residing on server \texttt{s}. Returns \texttt{None} if the job with ID \texttt{id} does not exist at server \texttt{s}. See section 6.9.9.4 “Job Objects” on page 523.
s.jobs()  
Returns a Python iterator that iterates over a list of pbs.job objects residing on server s. Returns an empty iterator if no jobs exist on server s. The iterator is described in section 6.9.14.5 “Iterator Objects” on page 542.
Example:
```python
for j in pbs.server().jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "found job %s" % (j.id))
```

s.queues()  
Returns a Python iterator that iterates over a list of queue objects managed by the server s. Returns an empty iterator if no queues exist on server s. The iterator is described in section 6.9.14.5 “Iterator Objects” on page 542.

s.resvs()  
Returns a Python iterator that iterates over a list of pbs.resv objects residing on server s. Returns an empty iterator if no reservations exist on server s. The iterator is described in section 6.9.14.5 “Iterator Objects” on page 542.

s.resv( <resvid> )  
Returns a pbs.resv object for <resvid> on server s. Returns None if <resvid> does not exist.

pbs.server().vnode('<vnode_name>')  
Returns a pbs.vnode object representing the vnode with <vnode_name> that is managed by the current server. Returns None if <vnode_name> does not exist.

pbs.server().vnodes()  
Returns a list of pbs.vnode objects managed by current server.
Returns an empty iterator if no vnodes exist on local server.
Example:
```python
for vn in pbs.server().vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "found vn %s" % (vn.name))
```
6.9.9.2.4 Examples of Using Server Object Attributes

Example 1: Get all the attributes and their values for the current PBS complex:
   ```python
s = pbs.server()
   
Example 2: Get server name:
   ```python
   name = s.name
   
Example 3: Get the value of the server attribute `pbs_license_min`:
   ```python
   min = s.pbs_license_min
   
Example 4: Get the queue object representing the queue workq, and its Priority value:
   ```python
   q = s.queue("workq")
   prior = q.Priority

6.9.9.3 Queue Objects

This represents the PBS queue associated with the triggering event and with the server at which the event is happening.

`pbs.queue`

Represents an instance of a PBS queue.

To get information about a particular queue with name `<name>`, you must go through the associated server. Use:

```python
pbs.server().queue("<name>")
```

6.9.9.3.1 Queue Object Attributes

An instance `q` of `pbs.queue` has the following attributes:

`q.name`

The queue name. This is a Python `str` (string). This attribute is read-only.
6.9.9.3.2 Setting Queue Object Attributes

You can set, but not unset, queue object attributes. To set the value of a queue object attribute named `<attribute name>`:

\[ q.<attribute name> = <attribute value> \]

6.9.9.3.3 Queue Object Methods

\[ q.job( <jobid> ) \]

Returns a `pbs.job` object representing PBS job with ID `<jobid>`. This job must be residing on the queue represented by `q`. Returns `None` if the job with the specified jobid does not exist, or if the job is not in the queue represented by `q`. See section 6.9.9.4 “Job Objects” on page 523.

\[ q.jobs() \]

Returns a Python iterator that iterates over a list of `pbs.job` objects representing the jobs on the queue `q`. Returns an empty iterator if no jobs exist on the queue `q`. The iterator is described in section 6.9.14.5 “Iterator Objects” on page 542.

Example:

\[
\text{for } j \text{ in pbs.server().queue("workq").jobs():}
\]

\[
\text{pbs.logmsg(pbs.LOG_DEBUG, "found job %s" \% (j.id))}
\]

6.9.9.4 Job Objects

The job object represents the PBS job associated with the event that triggers the hook. The job is also associated with the server at which the hook is executing.

`pbs.job`

Represents an instance of a PBS job.
To get information about a particular job with ID \(<id>\), you must go through the server. Use this:

\[
pbs.server().job("<id>")
\]

\(\text{pbs.event().job} \) can return only the job associated with the current event.

### 6.9.9.4.1 Job Object Attributes

An instance \(j\) of \(\text{pbs.job}\) has the following attributes:

\(j.id\)

- Refers to the PBS jobid.
- Type: Python \(\text{str}\)
- Read-only.

\(j.<\text{attribute name}>\)

- Refers to the named job attribute. Job attribute names (values for \(<\text{attribute name}>\)) are listed in the \(\text{pbs_job_attributes(7B)}\) man page.

### 6.9.9.4.2 Setting and Unsetting Job Object Attribute

You can set or unset a job object’s attributes.

Set the value of a job attribute named \(<\text{attribute name}>\):

\[
j.<\text{attribute name}> = \text{<attribute value>}
\]

Set \(<\text{attribute value}>\) to \(\text{None}\) to unset the attribute:

\[
j.<\text{attribute name}> = \text{None}
\]

### 6.9.9.4.3 Examples of Using Job Object Attributes

Get the job’s Priority value:

\[
prio = j.Priority
\]

Reset the Priority value of job \(j\):

\[
j.Priority = 5
\]

Get the job’s \(\text{PBS_O_WORKDIR}\) environment variable:

\[
pbs.event().job.Variable_List["PBS_O_WORKDIR"]
\]
6.9.9.4.4  Format of Job Object Time Attributes

For the attributes Execution_Time, ctime, etime, mtime, qtime, and stime, the pbs.job object expects or shows the number of seconds since Epoch. The only one of these that can be set is Execution_Time.

If you wish to set the value for Execution_Time using the [[CCYY]MMDDhhmm[.ss] format, or to see the value of any of the time attributes in the ASCII time format, then load the Python time module and use the functions time.mktime([CCYY, MM, DD, hh, mm, ss, -1, -1, -1]) and time.ctime().

Example:

```python
import time
j.Execution_Time = time.mktime([07, 11, 28, 14, 10, 15, -1, -1, -1])
time.ctime(j.Execution_Time)
'Wed Nov 28 14:10:15 2007'
```

6.9.9.4.5  Special Characters in Variable_List Job Attributes

When special characters are used in Variable_List job attributes, they must be escaped. For this attribute, special characters are comma (,), single quote (‘), double quote (“), and backslash (\). PBS requires each of these to be escaped with a backslash. However, Python requires that double quotes and backslashes also be escaped with a backslash. If the special character inside a string is a single quote, you must enclose the string in double quotes. If the special character inside the string is a double quote, you must enclose the string in single quotes. The following rules show how to use special characters in a Variable_List attribute when writing a Python script:

<table>
<thead>
<tr>
<th>Character</th>
<th>Example Value</th>
<th>How Value is Represented in Python Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>, (comma)</td>
<td>a,b</td>
<td>“a,b” or ‘a,b’</td>
</tr>
<tr>
<td>' (single quote)</td>
<td>c’d</td>
<td>“c\’d”</td>
</tr>
<tr>
<td>&quot; (double quote)</td>
<td>f”g”h</td>
<td>‘f&quot;g&quot;h’</td>
</tr>
</tbody>
</table>

Table 6-15: How to Use Special Characters in Python Scripts
Table 6-15: How to Use Special Characters in Python Scripts

<table>
<thead>
<tr>
<th>Character</th>
<th>Example Value</th>
<th>How Value is Represented in Python Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ (backslash)</td>
<td>\home\dir\files</td>
<td>“\home\dir\files” or ‘\home\dir\files’</td>
</tr>
</tbody>
</table>

For example:
The path is:

“\Documents and Settings\pbstest\bin:\windows\system32”

How the path shows up in a script:

```python
j.Variable_List[“PATH”] = “\Documents and Settings\pbstest\bin:\windows\system32”
```

6.9.9.4.6 Special Characters in string_array Attributes

For an attribute whose type is string_array and whose value contains one or more commas (“,“), the whole string must be enclosed in single quotes, outside of its double quotes. For example:

In PBS_HOME/server_priv/resourcedef:

```plaintext
test_string_array type=string_array
```

Value of test_string_array is:

```plaintext
glad, elated
```

In hook script:

```python
j.Resource_List[“test_string_array”] = ‘”glad, elated”’
```

6.9.9.5 Reservation Objects

This represents the PBS reservation associated with both the current server and the event that triggers the hook:

`pbs.resv`

Represents an instance representing a PBS reservation.
In order to retrieve information about the reservation associated with the triggering action, you must use a reference to the reservation object represented by:

```
pbs.event().resv
```

### 6.9.9.5.1 Reservation Object Attributes

An instance $r$ of `pbs.resv` has the following attributes:

- `r.resvid`  
  The reservation ID in Python string format.  
  Example: “R221.myhost”.  
  This attribute is read-only.

- `r.<attribute name>`  
  The reservation attribute named `<attribute name>`.  
  Attribute names are listed in “Reservation Attributes” on page 393 of the PBS Professional Reference Guide.

### 6.9.9.5.2 Setting Reservation Object Attribute Values

You can set, but not unset, reservation object attributes. To set a reservation object attribute:

```
r.<attribute name> = <attribute value>
```

### 6.9.9.5.3 Examples of Using Reservation Object Attributes

Example 1: Get the reservation’s owner:

```
owner = r.Reserve_Owner
```

Example 2: Reset the reservation’s name:

```
r.Reserve_Name = “Resv2008”
```

### 6.9.9.5.4 Format of Reservation Object Time Attributes

For the attributes `reserve_start`, `reserve_end`, and `ctime`, the `pbs.resv` object expects and shows the number of seconds since Epoch. The `ctime` attribute cannot be set.
If you want to set the value for `reserve_start` or `reserve_end` using the "[[CCYY]MMDDhhmm[.ss]]" format, or see the value for any time attribute displayed in the ASCII time format, load the Python time module and use the functions `time.mktime([CCYY, MM, DD, hh, mm, ss, -1, -1, -1])` and `time.ctime()`.

Example:

```python
import time
j.reserve_start = time.mktime([07, 11, 28, 14, 10, 15, -1, -1, -1])
time.ctime(j.reserve_start)
'Wed Nov 28 14:10:15 2007'
```

If `reserve_duration` is unset or set to `None`, the reservation’s duration is taken from the `walltime` resource attribute associated with the reservation request. If `reserve_duration` and `walltime` are both specified, meaning not set to `None`, `reserve_duration` will take precedence.

### 6.9.9.6 Vnode Objects

This represents a vnode associated with the current server.

`pbs.vnode`

In order to retrieve information about the vnode associated with the triggering action, you must use a reference to the vnode object represented by:

```python
pbs.server().vnode
```

### 6.9.9.6.1 Vnode Object Attributes

An instance `v` of `pbs.vnode` has the following attributes:

- `v.current_aoe`
  A Python `str` that identifies the AOE currently instantiated on this vnode.

- `v.comment`
  A Python `str` containing a vnode comment.

- `v.hpcbp_enable`
  A Python `bool` that indicates whether HPCBP features are enabled on `pbs_mom` managing vnode.
v.hpcbp_stage_protocol
   A Python str representing the protocol and optional port for staging files used by HPC Basic Profile Server.

v.hpcbp_user_name
   A Python str representing the HPCBP service account used by the HPCBP MOM managing vnode.

v.hpcbp_webservice_address
   A Python str representing the URL used by the HPCBP MOM to connect to the HPC Basic Profile Service.

v.jobs
   A Python str representing the jobs running on the vnode.
   Format: "<processor number>/jobid,..."

v.max_group_run
   A Python int value representing the maximum number of jobs owned by a single group that may be running on this vnode.

v.max_running
   A Python int value representing the maximum number of jobs that may be running on this vnode.

v.max_user_run
   A Python int value representing the maximum number of jobs owned by a single user that may be running on this vnode.

v.Mom
   Hostname where pbs_mom that manages the vnode runs.
   Type: string. Python type: str

v.name
   Specifies the name of the vnode. Type: string. Python type: str

v.no_multinode_jobs
   A Python bool that indicates whether or not jobs requesting more than one chunk are allowed to execute on this vnode.

v.ntype
   A Python int value representing the type of the vnode. Can have one of the PBS node type constant values in section 6.9.7.8 “Vnode Type Objects” on page 510.
Chapter 6  

**Hooks**

v.pbs_version  
A Python str value representing the PBS version of the corresponding pbs_mom.

v.pcpus  
A Python int value containing the number of physical CPUs on the vnode.

v.pnames  
Python str representing the list of resources being used for placement sets. Format: comma-separated list of resource names.

v.Port  
Network port number of the corresponding pbs_mom. Python type: int

v.Priority  
A Python int value representing the priority assigned to this vnode.

v.provision_enable  
A Python bool that indicates whether or not this vnode can be provisioned.

v.queue  
The pbs.queue object with which this vnode is associated. If none, value is *None*.

v.resv  
A Python str representing advance reservations pending on the vnode. Comma-separated list of the reservations to which the vnode has been allocated.

Format: "<reservation ID>, <reservation ID>, ..."

v.resv_enable  
A Python bool value that indicates whether or not this vnode can be used for reservations.

v.resources_available  
Python dictionary containing resource name=value pairs.

Format: resources_available['<resource name>'] = <resource val>

where <resource_name> is any custom or built-in resource. See Chapter 5, "Resources", on page 347.
v.resources_assigned
Python dictionary containing resource name=value pairs.
Format: resources_assigned['<resource name>'] = <resource val>
where <resource name> is any custom or built-in resource.
See Chapter 5, "Resources", on page 347.

v.sharing
A Python int value representing the value of the vnode sharing attribute. Can have one of the PBS sharing constant values in section 6.9.7.9 “Vnode Sharing Objects” on page 510.

v.state
A Python int representing the state of the vnode. Can have one or more of the PBS vnode state constant values in section 6.9.7.7 “Vnode State Objects” on page 508. Can be changed by runjob hook only.

6.9.9.2 Vnode Attribute Object Restrictions
• The only vnode attribute that can be changed is the state attribute
• The only hook that can change the vnode state attribute is the runjob hook

6.9.10 Select, Place, and Chunk Objects

6.9.10.1 Select Objects

pbs.select("[N:]res=val[:res=val].../[N:]res=val[:res=val] ... ]")
Creates a select object representing the select specification job attribute.
Instantiation of these objects requires a formatted input string.
Example:

```python
sel = pbs.select("2:ncpus=1:mem=5gb+3:ncpus=2:mem=5gb")
s = repr(sel)            (or s = `sel``)
letter = s[3]            (assigns 'c' to letter)
s = s + "+5:scratch=10gb" (append to string)
sel = pbs.select(s)      (reset the value of sel)
```

### 6.9.10.2 Place Objects

`pbs.place("[arrangement][:sharing][group]")`

- **arrangement** can be "pack", "scatter", "free", "vscatter"
- **sharing** can be "shared", "excl", "exclhost"
- **group** can be of the form "group=<resource>"

[arrangement], [sharing], and [group] can be given in any order or combination.

Creates a **place** object representing the given input string.

Instantiation of these objects requires a formatted input string.

Example:

```python
pl = pbs.place("pack:excl")
s = repr(pl)            (or s = `pl``)
letter = pl[0]          (assigns 'p' to letter)
s = s + ":group=host"   (append to string)
pl = pbs.place(s)       (update original pl)
```

### 6.9.10.3 Chunk Objects

`pbs.chunk`

Represents a chunk assigned to a job.

#### 6.9.10.3.1 Chunk Object Attributes

An instance `vc` of `pbs.vchunk` has the following attributes:
vc.vnode_name
Python str (string) which specifies the name of the vnode from which the chunk is taken.

vc.chunk_resources
Python dictionary containing resource name=value pairs.
Format: chunk_resources['<resource name>'] = <resource val>
where <resource_name> is any custom or built-in resource. See Chapter 5, "Resources", on page 347.

vc.chunk_resources.keys()
The list of <resource name> keys of chunk_resources. This list makes it convenient to list all the values of chunk_resources. Example:

    for r in vc.chunk_resources.keys():
        pbs.logmsg(pbs.LOG_DEBUG,
                   "c.chunk_resources[%s]=%s" % (r, vc.chunk_resources[r]))

6.9.11 Job Attribute Objects

6.9.11.1 Checkpoint Objects

    pbs.checkpoint("<checkpoint_string>")

where <checkpoint_string> must be one of "n", "s", "c", "c=mmm", "w", or "w=mmm"

Creates an object representing the job Checkpoint attribute, using the given string.
Instantiation of these objects requires a formatted input string.

6.9.11.2 Dependency Objects

    pbs.depend("<depend_string>")

<depend_string> must be of format "<type>:<jobid>[,<jobid>...]", or "on:<count>".
where \texttt{<type>\texttt{}} is one of "\texttt{after\texttt{}}", "\texttt{afterok\texttt{}}", "\texttt{afterany\texttt{}}", "\texttt{before\texttt{}}", "\texttt{befor-\texttt{eok\texttt{}}", and "\texttt{beforenotok\texttt{}}".

Creates a PBS dependency specification object representing the job \texttt{depend\texttt{}} attribute, using the given \texttt{<depend\_string>\texttt{}}.

Instantiation of these objects requires a formatted input string.

### 6.9.11.3 exec\texttt{-\texttt{host\texttt{}}} Objects

\begin{verbatim}
pbs.exec_host("host/N[*C][+...]")
\end{verbatim}

Create an object representing the \texttt{exec\_host\texttt{}} job attribute, using the given host and resource specification.

Instantiation of these objects requires a formatted input string.

### 6.9.11.4 exec\texttt{-\texttt{vnode\texttt{}}} Objects

\begin{verbatim}
pbs.exec_vnode("<vchunk>[+<vchunk> ...]")
\end{verbatim}

\texttt{<vchunk>\texttt{}} is \texttt{(\texttt{<vnode\_name\texttt{:ncpus=N\texttt{:mem=M}>\texttt{)}})}

Creates an object representing the \texttt{exec\_vnode\texttt{}} job attribute, using the given vnode and resource specification. When the \texttt{qrun -H} command is used, or when the scheduler runs a job, the \texttt{pbs.event().job.exec\_vnode\texttt{}} object contains the vnode specification for the job.

Instantiation of these objects requires a formatted input string.

Example:

\begin{verbatim}
pbs.exec_vnode("(vnodeA:ncpus=N:mem=X)+(nodeB:ncpus=P:
mem=Y+nodeC:mem=Z)"")
\end{verbatim}

This object is managed and accessed via the \texttt{str\texttt{()}} or \texttt{repr\texttt{()}} functions.

Example:

\begin{verbatim}
Python> ev = pbs.server().job("10").exec\_vnode
Python> str(ev)
"(vnodeA:ncpus=2:mem=200m)+(vnodeB:ncpus=5:mem=1g)"
\end{verbatim}
### 6.9.11.4.1 exec_vnode Object Attributes

`pbs.exec_vnode.chunks`

List of `pbs.vchunk` objects. These objects represent the chunks assigned to a job.

Example:

```python
j = pbs.event().job
pbs.logmsg(pbs.LOG_DEBUG, "job %s exec_vnode = %s" % (j.id, j.exec_vnode))

evc = j.exec_vnode.chunks

for c in evc:
    pbs.logmsg(pbs.LOG_DEBUG, "c.vnode_name=%s " % (c.vnode_name))

for r in c.chunk_resources.keys():
    pbs.logmsg(pbs.LOG_DEBUG, "c.chunk_resources[%s]=%s" % (r, c.chunk_resources[r]))
```
Sample output:

```
10:16:53;0006;Server@jobim;Hook;Server@jobim;job
  jobim exec_vnode =
    (jobim[2]:ncpus=2:mem=10240kb)+(jobim[1]:ncpus=2:mem=10240kb)+(jobim[3]:ncpus=2:mem=2048kb)
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.vnode_name=jobim[2]
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.chunk_resources[ncpus]=2
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.chunk_resources[mem]=10240kb
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.vnode_name=jobim[1]
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.chunk_resources[ncpus]=2
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.chunk_resources[mem]=10240kb
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.vnode_name=jobim[3]
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.chunk_resources[ncpus]=2
  10:16:53;0006;Server@jobim;Hook;Server@jobim;c.chunk_resources[mem]=2048kb
```

6.9.11.4.2  Restrictions on exec_vnode Object Attributes

You cannot build an `exec_vnode` object using chunks.

6.9.11.5  Hold Type Objects

```
pbs.hold_types("<hold_type_str>")
```

where `<hold_type_str>` is one of "u", "o", "s", or "n".

Creates an object representing the `Hold_Types` job attribute.

Instantiation of these objects requires a formatted input string.
6.9.11.6 Join Path Objects

\[ pbs.join\_path\{\"oe\"|\"eo\"|\"n\"}\]  
Creates an object representing the Join\_Path job attribute. 
Instantiation of these objects requires a formatted input string.

6.9.11.7 Keep Files Objects

\[ pbs.keep\_files\{\"<keep\_files\_str>\\}\]  
where \(<keep\_files\_str>\) is one of "o", "e", "oe", "eo".  
Creates an object representing the Keep\_Files job attribute.  
Instantiation of these objects requires a formatted input string.

6.9.11.8 Staging List Objects

\[ pbs.staging\_list\{\"<filespec>[,<filespec>,...]\\}\]  
where \(<filespec>\) is  
\(<execution\_path>@<storage\_host>:<storage\_path>\)  
Creates an object representing a job file staging parameters list.  
To use a staging list object:  
\[ pbs.event().job.stagein = pbs.staging\_list(...); \]  
Instantiation of these objects requires a formatted input string.

6.9.12 Other Attribute Objects

6.9.12.1 Job Sorting Formula Objects

\[ pbs.job\_sort\_formula\{\"<formula\_string>\\}\]  
where \(<formula\_string>\) is a string containing a math formula.  See section 4.8.20 “Using a Formula for Computing Job Execution Priority” on page 205.
Chapter 6  

Hooks

Creates an object representing the job_sort_formula server attribute.
Instantiation of these objects requires a formatted input string.

6.9.12.2 License Count Objects

\texttt{pbs.license\_count("Avail\_Global:<W> Avail\_Local:<X> Used:<Y> High\_Use:<Z>")}

Instantiates an object representing a license_count attribute.
Instantiation of these objects requires a formatted input string.

6.9.12.3 Routing Destination Objects

\texttt{pbs.route\_destinations("<queue\_spec>[,<queue\_spec>,...]")}

where \texttt{<queue\_spec>} is \texttt{queue\_name[@server\_host[:port]]}

Creates an object that represents a route_destinations routing queue attribute.
Instantiation of these objects requires a formatted input string.

6.9.12.4 State Count Objects


Instantiates an object representing a state_count attribute.
Instantiation of these objects requires a formatted input string.

6.9.13 Generic Attribute Objects

These objects can be used for the value of more than one PBS object attribute. For example, the \texttt{mail\_list()} object can be used for both reservation and job Mail_Users attributes, and the \texttt{path\_list()} object can be used for a job’s Shell_Path_List attribute.
6.9.13.1 **ACL Objects**

\[ pbs.acl("[+|-]<entity>][,...])\]

Creates an object representing a PBS ACL, using the given string parameter.

Instantiation of these objects requires a formatted input string.

6.9.13.2 **Email List Objects**

\[ pbs.email_list("<email_address1>,<email_address2>...]\]

Creates an object representing a mail list.

Instantiation of these objects requires a formatted input string.

6.9.13.3 **Group List Objects**

\[ pbs.group_list("<group_name>[@<host>][,<group_name>[@<host>]..\]

Creates an object representing a PBS group list.

To use a group list object:

\[ pbs.event().job.group_list = pbs.group_list(...)\]

Instantiation of these objects requires a formatted input string.

6.9.13.4 **Mail Points Objects**

\[ pbs.mail_points("<mail_points_string>")\]

where \( <mail_points_string> \) is "a", "b", and/or "θ", or "n".

Creates an object representing a Mail_Points attribute.

Instantiation of these objects requires a formatted input string.

6.9.13.5 **Node Group Key Objects**

\[ pbs.node_group_key("<resource>")\]
Chapter 6

Hooks

Creates an object representing the resource to be used for node grouping, using the specified resource.

6.9.13.6  Path List Objects

```plaintext
pbs.path_list("<path>@<host>[,<path>@<host> ..."])"
```

Creates an object representing a PBS pathname list.

To use a path list object:

```plaintext
pbs.event().job.Shell_Path_List = pbs.path_list(...)
```

Instantiation of these objects requires a formatted input string.

6.9.13.7  User List Objects

```plaintext
pbs.user_list("<user>@<host>[,<user>@<host>..."
```

Creates an object representing a PBS user list.

To use a user list object:

```plaintext
pbs.event().job.User_List = pbs.user_list(...)
```

Instantiation of these objects requires a formatted input string.

6.9.14  Other Objects

6.9.14.1  Argument Objects

```plaintext
pbs.args("<args>")
```

where `<args>` are space-separated PBS arguments to commands such as `qsub` or `qdel`.

Creates an object representing arguments to PBS commands such as `qsub` or `qdel`.

Example:

```plaintext
pbs.args("-Wsuppress_email=N -r y")
```

Instantiation of these objects requires a formatted input string.
6.9.14.2 Message Logging Methods

6.9.14.2.1 Method Taking Log Level and Message as Arguments

\[ pbs.logmsg(log\ event\ class, \ message) \]

where \( message \) is an arbitrary string, and
where \( log\ event\ class \) can be one of the message log event class constants:

- \( pbs.\log\_WARNING \)
- \( pbs.\log\_ERROR \)
- \( pbs.\log\_DEBUG \)

See “Message Log Event Class Objects” on page 505.

This puts a custom string in the PBS Server log.

Example:

```python
for j in pbs.server().jobs():
    pbs.logmsg(pbs.LOG_DEBUG, “found job %s” % (j.id))
```

6.9.14.2.2 Method Taking Job ID and Message as Arguments

\[ pbs.logjobmsg(job\ ID, \ message) \]

where \( job\ ID \) must be an existing or previously existing job ID
where \( message \) is an arbitrary string

This puts a custom string in the PBS Server log. The `tracejob` command can be used to print out the job-related messages logged by a hook script.

Messages are logged at log event class \( pbs.\log\_DEBUG \).

6.9.14.3 PBS Version Objects

\[ pbs.version(<pbs\ version\ string>) \]

Creates an object representing the PBS version string.

Instantiation of these objects requires a formatted input string.
6.9.14.4  Software Resource Objects

`pbs.software("<software info string>")`

Creates an object representing a site-dependent software resource.

Instantiation of these objects requires a formatted input string.

6.9.14.5  Iterator Objects

`iterator`

Returned by Server and queue object methods. Python iterator object, which can be fed to a looping construct (e.g. for, while) to iterate over a sequence type or class. The iterator type has two built-in methods:

`Iterator.__iter__()`
Returns the iterator object itself.

`Iterator.next()`
Returns the next item; raises `StopIteration` exception if no further items exist. Once `next()` raises `StopIteration`, any subsequent calls to `next()` raise the same exception.
6.10 Examples

Example 1: This PBS hook script rejects jobs which do not specify walltime.

Script RequireWalltime.py:

```python
import pbs
import sys

try:
    je = pbs.event()
    j = je.job
    if j.Resource_List["walltime"] == None :
        je.reject("Job has no walltime requested")

except SystemExit:
    pass

except pbs.UnsetResourceNameError:
    je.reject("Job has no walltime requested")
```

Create hook and import script:

```
qmgr -c 'create hook RequireWalltime event="queuejob"
qmgr -c 'import hook RequireWalltime application/x-python default RequireWalltime.py'
```
Example 2: This PBS hook script rejects jobs with CPU requests that are not multiples of 8:

Script Multiple8.py:

```python
import pbs
import sys

je = pbs.event()
j = je.job

mult_limit = 16

if j.Resource_List["ncpus"] != None:
    try:
        je = pbs.event()
j = je.job
        R = j.Resource_List["ncpus"] % mult_limit
        if R != 0:
            je.reject("Ncpus resource is not a multiple of %s." % (mult_limit,))
    except SystemExit:
        pass
    except (pbs.UnsetResourceNameError, TypeError):
        je.reject("Bad Ncpus resource value.")
else:
    R = pbs.event().job.Resource_List
    sel = repr(R["select"])
tot_ncpus = 0
    for chunk in sel.split("+"):
        nchunks = 1
```

544  PBS Professional 11.1 Administrator’s Guide
for c in chunk.split(":"):
    kv = c.split("=")
    if len(kv) == 1:
        nchunks = kv[0]
    elif len(kv) == 2:
        if kv[0] == "ncpus":
            tot_ncpus += (int(nchunks) * int(kv[1]))

try:
    mod = tot_ncpus % mult_limit
    if mod != 0:
        je.reject("Ncpus resource is not a multiple of %s.\n" % (mult_limit,))
except SystemExit:
    pass
except (pbs.UnsetResourceNameError, TypeError):
    je.reject("Bad Ncpus resource value.")

Create hook and import script:

qmgr -c 'create hook Multiple8 event="queuejob"
qmgr -c 'import hook Multiple8 application/x-python
default Multiple8.py'
Example 3: If a user asks for -l ncpus=8:ppn=24, change ncpus to 24

Script ChangeNcpus.py:

```python
import pbs
import sys

try:
    je = pbs.event()
    j = je.job
    j.Resource_List["ncpus"] =
    max(j.Resource_List["ncpus"],
    j.Resource_List["ppn"])
except SystemExit:
    pass
except (pbs.UnsetResourceNameError, pbs.BadResourceValError):
    je.reject("Failed to reset ncpus value")

Create hook and import script:

```qmgr -c 'create hook ChangeNcpus event="queuejob"
qmgr -c 'import hook ChangeNcpus application/x-python
default ChangeNcpus.py'```
Example 4: Calculate \( cph \) (\( cph = \text{total} \ ncpus \ * \ \text{walltime} \) (in hours)) and set a custom \( cph \) job resource to the value. (You must create the \( cph \) resource before using it.)

Script \text{CustCPH.py}:

```python
import pbs

R = pbs.event().job.Resource_List
sel = repr(R["select"])
tot_ncpus = 0
for chunk in sel.split("+")):
    nchunks = 1
    for c in chunk.split(":"):
        kv = c.split("=")
        if len(kv) == 1:
            nchunks = kv[0]
        elif len(kv) == 2:
            if kv[0] == "ncpus":
                tot_ncpus += (int(nchunks) * int(kv[1]))

R["cph"] = tot_ncpus * R["walltime"]
```

Create hook and import script:

```bash
qmgr -c 'create hook CustCPH event="queuejob"
qmgr -c 'import hook CustCPH application/x-python
default CustCPH.py'
```
Example 5: This PBS hook script puts a job in a particular queue (e.g. "interQ") if the job was submitted interactively (i.e. `qsub -I`)

Script `IQueue.py`:

```python
# get the pbs module
import pbs
import sys

try:
    # Get the hook event information and parameters
    # This will be for the 'queuejob' event type.
    je = pbs.event()

    # Get the information for the job being queued
    j = je.job

    if j.interactive:
        # Get the "interQ" queue object
        q = pbs.server().queue("interQ")
        # Reset the job's destination queue
        # parameter for this event
        j.queue = q
        # accept the event
        je.accept()
except SystemExit:
    pass
except:
    je.reject("Failed to route job to queue interQ");
```

Create hook and import script:

```
qmgr -c 'create hook IQueue event="queuejob"
qmgr -c 'import hook IQueue application/x-python
  default IQueue.py'
```
Example 6: Prevent users from using `qalter` to change their jobs in any way. Allow only administrators to change jobs.

Script `NoAlter.py`, on Windows, in a domain:

```python
import os
import pbs

e = pbs.event()
j = e.job
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
isadmin=0
admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "Administrator"]

if who in admin_ulist:
    isadmin=1
else:
    cmd = "net user " + who + " /domain"
    admin_glist = ['Administrators', 'Domain Admins', 'Enterprise
    Admins']
    for line in os.popen(cmd).readlines():
        if line.find("Group") >= 0:
            for li in line.split("*"):
                if li.strip() in admin_glist:
                    isadmin=1
                    break
if e.type == pbs.MODIFYJOB and not isadmin:
    e.reject("Normal users are not allowed to modify their jobs")
```
Script NoAlter.py, on Linux/UNIX:

```python
import pbs

e = pbs.event()

j = e.job

who = e.requestor

pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))

admin_u_list = ["PBS_Server", "Scheduler", "pbs_mom", "root"]

if who not in admin_u_list:
    e.reject("Normal users are not allowed to modify their jobs")
```

Create hook and import script:

```
qmgr -c 'create hook NoAlter event="modifyjob"
qmgr -c 'import hook NoAlter application/x-python
default NoAlter.py'
```
Example 7: Restrict the ability to submit a reservation to PBS administrators only.

Script NoSub.py on Windows:

```python
import pbs
import os
e = pbs.event()
r = e.resv
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))

isadmin=0

admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom",
               "Administrator"]

if who in admin_ulist:
isadmin=1
else:
    cmd = "net user " + who + "/domain"

admin_glist = ['Administrators', 'Domain Admins',
               'Enterprise Admins']

for line in os.popen(cmd).readlines():
    if line.find("Group") >= 0:
        for li in line.split("*"):
            if li.strip() in admin_glist:
                isadmin=1
                break

if e.type == pbs.RESVSUB and not isadmin:
e.reject("Only admins allowed to create reservations!")
```
Script `NoSub.py` on Linux/Unix:

```python
import pbs
import os

e = pbs.event()

r = e.resv

who = e.requestor

pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))

admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "root"]

if e.type == pbs.RESVSUB and who not in admin_ulist:
    e.reject("Only admins allowed to create reservations!")
```

Create hook and import script:

```bash
qmgr -c 'create hook NoSub event="resvsub"'
qmgr -c 'import hook NoSub application/x-python default
        NoSub.py'
```
Example 8: Reject jobs that request a specific queue, e.g. workq2, that do not request memory.

Script queuespec.py:

```python
import pbs
import sys
try:
    e = pbs.event()
    j = e.job
    if j.queue.name == "workq2" and not \\
       j.Resource_List["mem"]:
        e.reject("workq2 requires job to have mem specification")
except SystemExit:
    pass
except:
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))
```

Create hook, import script:

```bash
qmgr -c 'create hook queuespec event="modifyjob"
qmgr -c 'import hook queuespec application/x-python
        default queuespec.py'
```
Example 9: Redirecting newly-submitted jobs

If a job is submitted to a queue other than workq, move it to workq

```python
import pbs
import sys

try:
    # Get the hook event information and parameters
    # This will be for the 'queuejob' event type.
    je = pbs.event()

    # Ignore requests from scheduler or server
    if je.requestor in "["PBS_Server", "Scheduler"]":
        je.accept()

    # Get the information for the job being queued
    j = je.job
    if j.queue in ["long", "short"]:
        j.queue = pbs.server().queue("workq")

    # accept the event
    je.accept()
except SystemExit:
    pass
except:
    je.reject("Failed to route job to queue workq");
```
6.11 Errors, Logging, and Troubleshooting

6.11.1 Error Reporting and Logging

Hook errors are printed to stderr for the command (qsub, qalter, pbs_rsub, or qmove) that triggered the hook. If the hook provides a custom error message, that message is treated the same way.

Hooks can log custom strings to the log file of the daemon from which the hook is executing. When logging a message, a hook uses message logging methods to specify the message, and constant objects to specify the log event class. See section 6.9.14.2 “Message Logging Methods” on page 541, and See “Message Log Event Class Objects” on page 505.

When the PBS server starts, it prints to the server logs both the Python version integrated with the server, and a list of all the hook names registered with the server.

6.11.2 Errors During Creation and Deployment

6.11.2.1 Hook Name Matches Existing Hook

Creating a hook whose name matches that of an existing hook: the following error message is printed in stderr and in the server logs:

   “hook error: hook name <hook_name> already registered, try another name”

6.11.2.2 Using a Hook Name that Starts with “PBS”

Using a hook name that starts with “PBS”: the hook name is rejected with the following error in qmgr’s stderr, as well as in the server logs:

   “hook error: cannot use PBS as a prefix - it is reserved for PBS hooks”
6.11.2.3 Deleting a Non-Existent Hook

Deleting a non-existent hook: the following is returned in qmgr's stderr and server logs:

"qmgr: hook error: nohook does not exist"

6.11.2.4 Specifying a Non-Existent Event Type

Specifying a non-existent event type: an error message is printed to qmgr's stderr and also to the server logs:

Example:

Qmgr: set hook hook1 event="mom_checkpoint"
"hook error: invalid argument to event. Should be one of: queuejob, modifyjob, resvsub, movejob, or "" for no event."

6.11.2.5 Using a Bad Hook Value

Putting in a bad hook value: an error is printed to qmgr's stderr and also to the server logs:

Example:

Qmgr: set hook hook2 order=1025
"qmgr obj=hookA svr=default: order given (1025) is outside the acceptable range of [1, 1000] for type 'site'."
"qmgr: hook error returned from server"

6.11.2.6 Unauthorized User

If qmgr is invoked, and the object being operated on is “hook”, and the executing user at some host does not have access to the target server's private location for hooks data, then the following error is issued to stderr and server logs:

"<user>@<host> is unauthorized to access hooks data from server <hostname>"
6.11.2.7 Setting a Bad Hook Type

Setting a bad type to a hook produces the following error message in qmgr's stderr and also in the server logs:

"hook error: invalid argument to type. Must be site"

6.11.2.8 Setting a Bad Alarm Value

Setting a bad alarm value to a hook produces the following error message in qmgr's stderr and also in the server logs:

"hook error: alarm value of a hook must be > 0"

6.11.2.9 Exporting To Non-Writable File

Exporting a hook's content to a file that is not writable due to ownership or permission problems results in the following error message being printed to stderr:

"qmgr: hook error: <output_file> permission denied"

6.11.2.10 Setting Bad Hook User Attribute

Setting a value for the user attribute of a hook to something other than "pbsadmin" produces the following error message in qmgr's stderr and also in the server logs:

"hook error: user value of a hook must be pbsadmin"

This attribute does not need to be set to the actual name of the PBS service account.
6.11.2.11 Importing From Non-Readable File

Importing a hook where the PBS server is unable to open the input file because the file is non-existent, has permission a problem, or whatever other system-related error causes the following error message to be printed in stderr and in the server logs:

"qmgr: hook error: unable to open <filename> by server
run by <user>@<host>: <error message>"

Examples:

"qmgr: hook error: unable to open hook1.py by server
run by pbsadmin@hostX: permission denied"
"qmgr: hook error: unable to open hook1.py by server
run by pbsadmin@hostY: No such file or directory"

6.11.2.12 Importing/Exporting With Wrong Content-Type

Importing/exporting a hook where the <content-type> is something other than “application/x-python” causes the following error message to be printed in stderr and on the server logs:

"qmgr: hook error: <content_type> must be 'application/x-python'"

Importing/exporting a hook where the <content-encoding> is something other than “default” or “base64” causes the following error message to be printed in stderr and on the server logs:

"qmgr: hook error: <content_encoding> must be
'default' or 'base64'"

An import call on a hook that already has a content script results in the following informational message being printed in stdout and server logs:

"qmgr: hook <hook_name> contents overwritten by file
<hook input file>"

6.11.2.13 Setting Vnode State to Invalid Value

Setting a vnode’s state attribute to an invalid value causes the pbs.BadAttributeValueError exception to be raised.
6.11.3 Errors And Messages During Hook Execution

6.11.3.1 Rejecting an Action

If a hook rejects an action by calling the `pbs.event().reject()` function:

- The following messages are printed to `stderr` of the command that triggered the hook:
  
  
  "<command_name>: Request rejected by filter hook <hook_name>" "<command_name>:<'msg' value passed to pbs.event().reject()>
  
  where 'msg' is the message passed (if any) as input to `pbs.event().reject()`.

- The following messages are printed in the appropriate PBS daemon log, logged at 0x0400 event class:
  
  "<user>@<host>…<request type> request rejected by <hook name> "<user>@<host> …<request type> <'msg' value passed to pbs.event().reject()>

6.11.3.2 Triggering an Alarm

If the alarm was triggered while executing a hook:

- The command that initiated the request gets the following messages in its `stderr`:
  
  "<command_name>: Request rejected by filter hook <hook_name>" "<command_name>: alarm call while running hook <hook_name>"

- The following entry appears in the appropriate PBS daemon log, logged under 0x0100 event class:
  
  "<user>@<host>…<request type> alarm call while running hook <hook_name>, request rejected"
6.11.3.3 Encountering an Unhandled Exception

If a hook encounters an unhandled exception:

- PBS rejects the corresponding action. The command that triggered the hook gets the following message in stderr:
  "<command_name>: request rejected as filter hook <hook_name> encountered an exception. Inform admin."

- The following message appears on the appropriate PBS daemon log, logged under 0x0100 event class:
  "<request type> hook <hook_name> encountered an exception, request rejected"

See section 6.9.6.3.1 “Unhandled Exceptions” on page 503.

6.11.3.4 Starting and Finishing Hook Execution

Whenever hook execution starts or finishes, timestamped, 0x0400 event class log messages appear in the appropriate PBS daemon log:

"11/13/2007 00:00:42 ...<user>@<host>...<request type> running hook named <hook name>"

"11/13/2007 00:01:42<user@><host>...<request type> <hook_name> finished"

See section 6.9.9.1.10 “Event Object Methods” on page 517.

6.11.3.5 Hook Timeout

When a hook timeout is triggered, the hook script gets a Python KeyboardInterrupt from the PBS server. The server logs show:

06/17/2008
17:57:16;0001;Server@fest;Svr;Server@fest;PBS server internal error (15011) in Python script received a KeyboardInterrupt, <type 'exceptions.KeyboardInterrupt'>
6.11.3.6 Hooks Attempting I/O

When the PBS server is running, stdout, stderr, and stdin are closed, so that a hook script containing calls to print to standard output or standard error, or to read input from standard input, gets the following exception:

```
02/24/2008 08:03:34;0086;Server@a-centauri;Svr;Server@a-centauri;Compiling script file:
   </usr/spool/PBS/server_priv/hooks/hook_test.PY>
02/24/2008 08:03:34;0001;Server@a-centauri;Svr;Server@a-centauri;PBS server internal
   error (15011) in Error evaluating Python script,
   <type 'exceptions.IOError'>
```

6.11.3.7 runjob Hook Modifying Accepted Action

runjob hooks can modify only those jobs that are rejected by the hook.

6.11.3.7.1 Modifying Jobs

If a runjob hook accepts an event request, using pbs.event().accept(), but attempts to put the job into a Waiting or Held state, then the hook request is rejected.

If the hook is triggered by a qrun command, the following message is sent to stderr where the qrun command was run. If the hook is triggered when the scheduler tries to run the job, the following message is written to the job’s comment attribute:

```
request rejected by filter hook <hook_name>: cannot modify job after runjob request has been accepted.
```

The following message is written to the PBS server log, at log event class 0x0100:

```
runjob request rejected by <hook name>: cannot modify job after runjob request has been accepted.
```
6.11.3.7.2 Modifying Vnode

If a runjob hook event is accepted via a `pbs.event().accept()` call, and yet an attempt is made to modify a vnode's state, then the hook request is rejected. The following message is sent to the `stderr` of `qrun`, and becomes the job’s comment:

```
request rejected by filter hook <hook_name>: cannot modify vnode after runjob request has been accepted.
```

The following message appears in the PBS server log, logged at event class 0x0100:

```
runjob request rejected by <hook name>: cannot modify a vnode after runjob request has been accepted.
```

6.11.3.8 Runjob Hook Referencing Wrong Parameter

If a `runjob` hook attempts to reference a `pbs.event()` parameter other than `pbs.event().job`, the exception `pbs.EventIncompatibleError` is raised.

6.11.4 Debugging Hook Scripts

The following steps may help you avoid errors in hook scripts:

1. Create a hook, and import its content.

2. Temporarily set the Server's `log_events` to a higher value such as 2047 to see plenty of logging.

3. Do a test run of the hook script, by causing events (e.g. `qsub`, `qalter`, `qmove`, `pbs_rsub`) that invoke the hook script. Check for error messages in the server logs.

4. Correct the hook script, re-import the fixed code, and rerun the test.

5. Once the hook script is running fine, then set the Server’s `log_events` back to default (i.e. 511).
6.12 Advice and Caveats

6.12.1 Choosing What Information to Use

The following are recommendations for writing hooks:

- Use only the documented interfaces. Do not attempt to manipulate the hook stored by PBS.
- Don’t delete attributes.
- Don’t change environment variables set by PBS. See “Environment Variables” on page 265 of the PBS Professional Reference Guide for a list of these environment variables.
- Do not try to access the following (a well-written, portable hook will not depend on any of the following information):
  - Server configuration information other than what is available via qstat (e.g., qmgr, resourcedef and pbs.conf)
  - Scheduling information other than what is available via qstat (e.g., qmgr, sched_config, fairshare, dedicated, holidays)
  - Vnode information other than what is available via qstat (e.g., qmgr, pbsnodes)
- Do not write hooks that depend on the behavior of other hooks.
- Do not make assumptions about the value of PATH; use “import sys” and “modify sys.path”
- Do not make assumptions about the value of the current working directory.
- Do not depend on order of execution of unrelated hooks. For example, do not depend on one job submission’s queuejob hooks running entirely before another job submission’s queuejob hooks. It is not guaranteed that all of one job’s hooks will finish before another job’s
hooks start.

- The `Resource_List` attribute is listed as a Python dictionary. These dictionaries support a restricted set of operations. They can reference values by index. Other features, such as `has_key()`, are not available.

## 6.12.2 Referencing the Object You Want

In order to reference the job, reservation, etc. that you want, you must use `pbs.server()` or `pbs.event()`. See “Server Object Methods” on page 520 of the PBS Professional Reference Guide and “Event Object Methods” on page 517 of the PBS Professional Reference Guide.

### 6.12.2.1 Referencing Job IDs

In order to get information about a particular job with ID `<id>`, use `pbs.server().job('<id>')`. `pbs.event()` can only return the job associated with the current event.

### 6.12.2.2 Referencing Reservations

In order to get information about a reservation being created in a `resvsub` event, use `pbs.event().resv`. `pbs.server` cannot return information about the reservation, because the reservation has not yet been created.

### 6.12.2.3 Referencing Destination Queues

Hooks have access only to the local server. Hooks can allow a job submission to a remote server, but they cannot specify a remote server. See section 6.12.8 “Limitations” on page 575. Hooks can specify the destination queue at a local server for a `queuejob` or `movejob` event.

To specify a destination queue at the local server:

```python
pbs.event().job.queue =
    pbs.server().queue("<local_queue>")
```

Do not specify a queue at a remote server in a hook script.
6.12.3 Recommended Hook Script Structure

Your hook script should catch all exceptions except for SystemExit. It is helpful if it displays a useful error message in the stderr of the command triggering the hook. The error message should show the type of the error and should describe the error. Here is the recommended script structure:

```python
import pbs
import sys

try:
    ...

except SystemExit:
    pass

except:
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))
```
This example shows how a coding error in the hook is caught with the `except` statement, and an appropriate error message is generated. In line 7, the statement `k=5/0` generates a divide-by-zero error. The hook script is designed to reject interactive jobs that are submitted to queue “nointer”.

```python
import pbs
import sys

try:

    batchq = "nointer"
    e = pbs.event()
    j = e.job
    k = 5/0
    if j.queue and j.queue.name == batchq and \
        j.interactive:
        e.reject("Can't submit an interactive job \n        in '%s' queue" % (batchq))

except SystemExit:
    pass

except:
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))
```

The hook is triggered:

```
% qsub job.scr
qsub: c1 hook failed with (<type 'exceptions.ZeroDivisionError'>, ZeroDivisionError('integer division or modulo by zero',)). Please contact Admin
```
6.12.4 Avoiding Interference with Normal Operation

6.12.4.1 Treating SystemExit as a Normal Occurrence

Both `ev.accept()` and `ev.reject()` terminate hook execution by throwing a `SystemExit` exception. A `try...except` clause without arguments will catch all exceptions. If hook content appears in a `try except " clause add the following to treat `SystemExit` as a normal occurrence:

```python
except SystemExit:
    pass
```

Here is an example of an `except` clause that will catch `SystemExit`:

```python
try:
    ....
except:
    ....
```

In the above case, we need to add the `except SystemExit`, so that it will look like this:

```python
try:
    ....
except SystemExit:
    pass
except:
    ....
```

If the existing code has a specified exception, we don't need to add "except `SystemExit:"", since this hook script is only catching one particular exception and will not match `SystemExit`. For example:

```python
try:
    ...
except pbs.BadAttributeValueError:
    ...
```
6.12.4.2 Allowing the Server and Scheduler to Modify Jobs

The Server and Scheduler both use the qalter command during normal operation to modify jobs. Therefore, if you have a modifyjob hook script, make sure you do not interfere with qalter commands issued by the Server or Scheduler. Catch these cases this way:

```python
e = pbs.event()
e.requestor in [ "PBS_Server", "Scheduler" ]:
e.accept()
```

6.12.4.3 Using Sleep in a Windows Hook Script

Under Windows, the PBS server cannot interrupt a hook script executing the Python `time.sleep()` function. The server needs to be able to interrupt the script if the script reaches its timeout. In order to be able to interrupt the script, create a sleep that incrementally sleeps for 1 second. The server can then interrupt the hook script in between the sleeps. For example:

```python
import time
def mysleep(sec):
    for i in range(sec):
        time.sleep(1)
    mysleep(30)  # pseudo sleep for 30 seconds
```

6.12.4.4 Staying Within the Scheduler’s Alarm Time

Consider setting hook `alarm` values so that they do not unduly delay the scheduler. The scheduler will wait for a hook to finish executing. The scheduler’s cycle time has a default value of 20 minutes, and is specified in the scheduler’s sched_cycle_length attribute.
6.12.5 Avoiding Problems

6.12.5.1 Avoid Hook File I/O

When the PBS server is running, `stdout`, `stderr`, and `stdin` are closed. A hook script attempting I/O will get an exception. To avoid this, redirect input and output to a file. See section 6.11.3.6 “Hooks Attempting I/O” on page 561.

6.12.5.2 Avoid Contacting Bad Host

Be careful not to specify a bad host in `<job-id>` in `j.depend`. If it references a non-existent or heavily loaded PBS server, the current PBS server could hang for a few minutes as it tries to contact the bad host. For example:

```python
pbs.event().job.depend =
    pbs.depend("after:23.bad_host")
```

The PBS server could hang while trying to contact "bad_host".

6.12.5.3 Avoid os._exit() Python Function

Do not use the `os._exit()` Python function. It will cause the PBS server to exit.

6.12.5.4 Avoid Attempting to Log Message Using Bad Job ID

If the `pbs.logjobmsg()` method is passed a bad job ID, it raises a Python `ValueError`.

6.12.5.5 Avoid Taking Up Lots of Memory

Certain function calls in PBS Python hooks are expensive to use in terms of memory. If they are called repeatedly in loops, they can use up a lot of memory, potentially causing the server to hang or crash. For example, the
following is expensive since each iterative call to `pbs.server().vnodes()`
causes internal allocation of memory, which won't be freed until after the
hook executes.

In order to avoid this, produce the output only once, save it to memory, and
iterate using the copy. For example:

```python
vnl = []
vni = pbs.server().vnodes()
for vn in vni:
pbs.logmsg(pbs.LOG_DEBUG, "found vn.name=%s" % (vn.name))
vnl.append(vn)

# Now use the saved pbs.server().vnodes() output
for i in range(80):
    for j in pbs.server().vnodes()
```

The following functions in PBS Python hooks return iterators, and should
be used carefully:

- Iterate over a list of jobs:
  ```python
  pbs.server().jobs()
  pbs.server(<queue>).jobs()
  ```
- Iterate over a list of queues:
  ```python
  pbs.server().queues()
  ```
- Iterate over a list of vnodes:
  ```python
  pbs.server().vnodes()
  ```
- Iterate over a list of reservations:
  ```python
  pbs.server().resvs()
  ```
6.12.6 Viewing And Changing Attributes

The job, vnode, or reservation object's attributes appear to the hook as they would be after the event, not before it.

For a movejob event, the only attribute that can be changed is the job's Queue attribute. The job's current queue is shown in the pbs.event().src_queue event attribute. A movejob hook does not run for a pbs_rsub -Wqmove=<job ID>.

For a modifyjob event, since many attributes can be changed, the original job with all its attributes is shown in pbs.event().job_o.

For a runjob event, the only job attributes that can be changed are the job's Hold_Types and Execution_Time attributes, and only if the job is rejected.

For a runjob event, the only vnode attribute that can be changed is the state attribute. The only hook that can change this attribute is a runjob hook.

6.12.6.1 Attribute Values Available to Hooks

Note that for job-related events (movejob, queuejob, modifyjob), the attributes shown to the hook are as they would be if the event were to take place, not as they are before the event. See section 6.9.3 “Using Attributes in Hooks” on page 489.

For a runjob hook, the vnode attributes available to the hook are as they would be after the event were to take place, not before.

6.12.6.2 Testing Vnode State

To see whether a vnode has a particular state set, use the Python bitwise and ("&") operator, for example:

    If v.state & pbs.ND_OFFLINE:
        pbs.logmsg(pbs.LOG_DEBUG, "vnode %s is offline!" % (v.name))
6.12.6.3 Permissible Attribute Changes

6.12.6.3.1 When to Change Reservation Attributes

The only time that a reservation’s attributes can be altered is during the creation of that reservation.

6.12.6.3.2 Caution About Unsetting Reservation Walltime Attribute

The `walltime` resource is used to determine the reservation’s `duration` parameter when the reservation’s `reserve_duration` attribute is not set or is set to `None`. If a `resvsub` hook attempts to unset the `walltime` parameter, for example:

```python
pbs.event().resv.Resource_List["walltime"] = None
```

This will result in the following error:

```bash
% pbs_rsub -R 1800 -l ncpus=1
pbs_rsub: Bad time specification(s)
```

6.12.6.3.3 Changing Job Attributes For a Running Job

When a job is running, only the `cput` and `walltime` attributes can be modified. Attempting to change any other attributes for a running job will cause the corresponding `qalter` action to be rejected. For example, if the job is running, this line in a hook will cause `qalter` to be rejected:

```python
pbs.event().job.Resource_List["mem"] = pbs.size("10mb")
```

To avoid having the `qalter` action rejected, check to see whether the job is running, and follow up accordingly. For example:

```python
e = pbs.event()
if e.job.job_state in [ pbs.JOB_STATE_RUNNING, pbs.JOB_STATE_EXITING, pbs.JOB_STATE_TRANSIT ]:
    e.accept()
```
6.12.6.3.4 Do Not Unset Array Job Indices

Do not unset pbs.event().job.array_indices_submitted for an array job in a modifyjob hook. For example:

```python
pbs.event().job.array_indices_submitted = None
```

If the hook script is executed for a job array, the qalter request will fail with the message:

```
Cannot modify attribute while job running <job_array ID>
```

6.12.6.3.5 Do Not Create Job or Reservation Variable_List

Hooks are not allowed to create job or reservation Variable_List attributes. Hooks can modify the existing Variable_List attribute which is supplied by PBS, by modifying values in the list. The following are disallowed in a hook:

```python
pbs.event().job.Variable_List = dict()
pbs.event().resv.Variable_List = dict()
```

These calls will cause the following exception:

```
04/07/2008 11:22:14;0001;Server@fest;Svr;Server@fest;PBS server internal error (15011) in Error evaluating Python script, attribute 'Variable_List' cannot be directly set.
```

To modify the Variable_List attribute:

```python
pbs.event().job.Variable_List["SIMULATE"] = "HOOK00"
```

6.12.6.3.6 Changing Vnode state Attribute

A vnode's state can be set within a runjob hook only if the runjob hook execution concludes with a pbs.event().reject() call. This means that if a statement that sets a vnode’s state appears in a hook script, it takes effect only if the following is the last line to be executed:

```python
pbs.event().reject()
```
To set a vnode’s state, the syntax is one of the following:

\[
\begin{align*}
& \text{pbs.server().vnode(<vnode-name>).state} = \text{<vnode state constant>} \\
& \text{pbs.server().vnode(<vnode-name>).state} += \text{<vnode state constant>} \\
& \text{pbs.server().vnode(<vnode-name>).state} -= \text{<vnode state constant>}
\end{align*}
\]

where \text{<vnode state constant>} is one of the constant objects listed in section 6.9.7.7 “Vnode State Objects” on page 508.

Examples of changing a vnode’s state attribute:

- To offline a vnode:
  \[
  \text{pbs.server().vnode(vnodeA).state} = \text{pbs.ND_OFFLINE}
  \]
- To add another value to the list of vnode states:
  \[
  \text{pbs.server().vnode(vnodeA).state} += \text{pbs.ND_DOWN}
  \]
- To remove a value from the list of vnode states:
  \[
  \text{pbs.server().vnode(vnodeA).state} -= \text{pbs.ND_OFFLINE}
  \]

When a vnode’s state attribute has no states set, the vnode’s state is equivalent to free. This means that you can remove all values, and the vnode will become free.

When a vnode’s state is successfully set, the following message is displayed and logged at event class 0x0004:

\[
\text{Node;<vnode-name>;attributes set: state - <vnode state constant> by <hook_name>}
\]

### 6.12.7 Required Interfaces

Hooks are required to use the interfaces provided here to access any PBS information or change any PBS state. Hooks which access PBS information or modify PBS in any way except through these interfaces are erroneous and unsupported. For example, a valid hook cannot read pbs.conf or resourcedef, or edit the holidays file directly.

Hooks which execute PBS commands are erroneous and unsupported. The behavior of executing PBS commands inside a hook is undefined (and is likely to cause the hook to hang).
6.12.8 Limitations

Hooks cannot access a server other than the local server. Hooks also cannot specify a non-default server. So for example if a job submission specifies a queue at a server other than the default, the hook can allow that submission, or can change the destination to be at the default server, but cannot change it to another non-default server.

6.12.9 Scheduling Impact of Hooks

6.12.9.1 Effect of runjob Hooks on Preemption

With preemption turned on, the scheduler preempts low-priority jobs to run a high-priority job. If the high-priority job is rejected by a runjob hook, then the scheduler undoes the preemption of the low-priority jobs. Suspended jobs are resumed, and checkpointed jobs are restarted.

6.12.9.2 Effect of runjob Hooks with Strict Ordering

When strict_ordering is set to True and backfill is set to False, a most-deserving job that is repeatedly rejected by a runjob hook will prevent other jobs from being able to run. A well-written hook would put the job on hold or requeue the job at some later time to prevent idling the system.

6.12.9.3 Effect of runjob Hooks with round_robin and by_queue

With round_robin and by_queue set to True, a job continually rejected by a runjob hook may prevent other jobs from the same queue from being run. A well-written hook would put the job on hold or requeue the job at some later time to allow other jobs in the same queue to be run.

A runjob hook's performance directly affects the responsiveness of the PBS scheduler. Consider carefully the trade-off between the work such a hook needs to do and your scheduler's required performance.
6.12.9.4 Peer Scheduling and Hooks

When a job is pulled from one complex to another, the following happens:

- Hooks are applied at the new complex as if the job had been submitted locally.
- Any `movejob` hooks at the furnishing server are run.

6.13 See Also

For a description of the PBS hook APIs, see the PBS Professional Programmer’s Guide. Each PBS object’s attribute’s Python type is listed in its description in “Attributes” on page 361 of the PBS Professional Reference Guide. For example, “Server Attributes” on page 365 of the PBS Professional Reference Guide lists the Python type for the `job_sort_formula` server attribute.

The following man pages and equivalent sections contain useful information:

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Guide Section</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pbs_module(7B)</code></td>
<td>section 9.4 “The pbs Module” on page 125 of the PBS Professional Programmers Guide</td>
</tr>
<tr>
<td><code>pbs_stathook(3B)</code></td>
<td>section 9.5 “The pbs_stathook() API” on page 134 of the PBS Professional Programmers Guide</td>
</tr>
<tr>
<td><code>pbs_hook_attributes(7B)</code></td>
<td>“Hook Attributes” on page 453 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>pbs_job_attributes(7B)</code></td>
<td>“Job Attributes” on page 430 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>pbs_server_attributes(7B)</code></td>
<td>“Server Attributes” on page 365 of the PBS Professional Reference Guide</td>
</tr>
</tbody>
</table>
## Table 6-16: See Also

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Guide Section</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pbs_queue_attributes(7B)</code></td>
<td>“Queue Attributes” on page 407 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>pbs_node_attributes(7B)</code></td>
<td>“Vnode Attributes” on page 420 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>qmgr(1B)</code></td>
<td>“qmgr” on page 178 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>qsub(1B)</code></td>
<td>“qsub” on page 245 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>qmove(1B)</code></td>
<td>“qmove” on page 201 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>qalter(1B)</code></td>
<td>“qalter” on page 152 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>pbs_rsub(1B)</code></td>
<td>“pbs_rsub” on page 95 of the PBS Professional Reference Guide</td>
</tr>
<tr>
<td><code>pbs_manager(3B)</code></td>
<td>section “pbs_manager” on page 49 of the PBS Professional Programmers Guide</td>
</tr>
</tbody>
</table>
Chapter 7

Metascheduling Using HPC Basic Profile

PBS Professional can schedule and manage jobs on one or more HPC Basic Profile Servers using the Grid Forum OGSA HPC Basic Profile web services standard. This chapter describes how to configure PBS Professional to use HPCBP.

7.1 Definitions

HPC Basic Profile (HPCBP)
Proposed standard web services specification for basic job execution capabilities defined by the OGSA High Performance Computing Profile Working Group. See section 7.8.1 “References” on page 597.
Chapter 7  Metascheduling Using HPC Basic Profile

HPC Basic Profile Server
Service that executes jobs from any HPC Basic Profile compliant client.

HPCBP MOM
MOM that sends jobs for execution to an HPC Basic Profile Server. This MOM is a client-side implementation of the HPC Basic Profile Specification, and acts as a proxy for and interface to an HPC Basic Profile compliant server.

Job Submission Description Language (JSDL)
Language for describing the resource requirements of jobs.

7.2  Using HPC Basic Profile

7.2.1  Architecture Restrictions

PBS works with HPCBP only when PBS is running on a subset of platforms that support this. You must run PBS on the platforms that are correct for using HPCBP. PBS manages HPCBP jobs only when the following machines have the supported architectures:

• The PBS server host
• Hosts used to submit HPCBP jobs
• Hosts where the HPCBP MOM runs
• Hosts where HPCBP Servers run

See the PBS Professional release notes for a list of supported architectures for each type of host.

7.2.2  Metascheduling Using HPCBP

PBS can schedule and manage jobs on one or more HPC Basic Profile compliant servers. PBS runs jobs on nodes managed by PBS and/or on nodes managed by an HPC Basic Profile Server, depending on what the HPCBP site policies and resource availability dictate.
These metascheduling capabilities are defined by the Grid Forum OGSA HPC Basic Profile web services standard. See section 7.8.1 “References” on page 597.

Users submit jobs via PBS that are intended to be run either within a PBS complex or on HPCBP Servers outside the complex. The PBS Server sends HPCBP jobs to a special MOM called the HPCBP MOM. This MOM sends HPCBP jobs to the HPCBP Server, which then assigns the jobs according to its policies. The HPC Basic Profile Server manages a set of its own nodes. Each of these is represented as a vnode to PBS.

7.2.3 How HPCBP Works with PBS

You can use HPCBP with PBS in one of two ways:

- Install PBS Professional directly on the Windows HPC nodes, and don't use Microsoft's Job Scheduler.
- If you have a larger PBS Professional installation that you want to expand by adding some Windows HPC nodes, and you want to use Microsoft's HPC tools and Job Scheduler, then use the Metascheduling via HPCBP to Windows HPC feature of PBS Professional.

In this document, we discuss the second method.

You configure one HPCBP MOM per HPCBP Server (Microsoft HPC Server 2008) and add all the HPCBP MOMs to PBS server. Now the PBS server knows about the HPCBP MOMs and can send them jobs. The same PBS server can manage a mix of regular MOMs and HPCBP MOMs. As long as a job does not specify something that restricts where it can run, the PBS scheduler can run it wherever resources are available. The job could run at a regular MOM, or it could be sent to an HPCBP server. You can configure the attributes of the vnodes managed by the HPCBP MOM. For example, you can set the queue attribute on those vnodes so that they accept jobs from a specific queue.

7.2.3.1 The HPCBP MOM

The HPCBP MOM acts as an intermediary between the PBS complex and the HPCBP Server. The HPCBP MOM does the following:

- It converts between the formats used by PBS and HPCBP for informa-
tion going both to and from the HPC Basic Profile Server.

- It takes job requests from the PBS complex, converts them to JSDL, and submits these converted jobs to the HPC Basic Profile Server. This includes selecting hosts managed by the HPCBP Server from the list of hosts specified by a job, and passing this list as part of the JSDL document to the HPCBP Server.

- It retrieves job status information from the HPCBP Server using an HPCBP job identifier and hands this status information to the PBS Server.

- It retrieves node status from the HPCBP Server and returns it to the PBS Server.

### 7.2.3.1.1 MOM Configuration Variables for an HPCBP MOM

Some MOM configuration variables behave differently in the HPCBP MOM. The following table lists MOM configuration variables in the `mom_priv/config` file which function differently in the HPCBP MOM.

<table>
<thead>
<tr>
<th>MOM Configuration Variable</th>
<th>Functionality for HPCBP MOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$action checkpoint</td>
<td>No effect. (Checkpointing is not supported.)</td>
</tr>
<tr>
<td>$action checkpoint_abort</td>
<td>No effect. (Checkpointing is not supported.)</td>
</tr>
<tr>
<td>$action multinodebusy</td>
<td>No effect. (Checkpointing is not supported.)</td>
</tr>
<tr>
<td>$action restart</td>
<td>Checkpointing is not supported. This script must be a requeue script.</td>
</tr>
<tr>
<td>$action terminate</td>
<td>No effect. (Checkpointing is not supported.)</td>
</tr>
<tr>
<td>$checkpoint_path</td>
<td>No effect. (Checkpointing is not supported.)</td>
</tr>
<tr>
<td>$clienthost</td>
<td>List of PBS MOMs allowed to connect to HPCBP MOM.</td>
</tr>
<tr>
<td>$dce_refresh_delta</td>
<td>Unsupported</td>
</tr>
</tbody>
</table>
HPCBP-specific vnode attributes control how the HPCBP MOM works with the HPCBP Server. These attributes are set on the vnode where the HPCBP MOM runs. Instructions for setting these attributes are given in section 7.3 “Configuring PBS for HPCBP” on page 585. The following table lists the HPCBP-specific vnode attributes:

**Table 7-2: HPCBP-specific Vnode Attributes**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Data type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpcbp_webservice_address</td>
<td>String</td>
<td>---</td>
<td>URL to connect to HPC Basic Profile Server</td>
</tr>
<tr>
<td>hpcbp_user_name</td>
<td>String</td>
<td>---</td>
<td>HPCBP service account used by HPCBP MOM for retrieving job and node status from HPC Basic Profile Server</td>
</tr>
</tbody>
</table>
Chapter 7  
**Metascheduling Using HPC Basic Profile**

---

### Table 7-2: HPCBP-specific Vnode Attributes

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Data type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpcbp_stage_protocol</td>
<td>String</td>
<td>scp</td>
<td>Protocol and port for staging files by HPC Basic Profile Server</td>
</tr>
<tr>
<td>hpcbp_enable</td>
<td>Boolean</td>
<td>False</td>
<td>Enables HPCBP features in MOM</td>
</tr>
</tbody>
</table>

---

### 7.2.4 Sending Jobs to HPCBP

PBS runs a HPCBP MOM on a Linux machine and sends jobs to the MOM. This MOM sends its jobs to an HPCBP Server. The HPCBP Server runs the jobs on the nodes it manages.

If a job that will run at the HPCBP Server requests an architecture, that architecture must be supported. See the Release Notes for a list of supported architectures.

### 7.2.5 Enforcing Resource Limits

Some resource limits are enforced by the HPCBP Server. The following table shows which resource limits are enforced by the HPCBP Server:

---

### Table 7-3: Resource Limits Enforced by HPCBP Server

<table>
<thead>
<tr>
<th>Resource Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>cput</td>
</tr>
<tr>
<td>walltime</td>
</tr>
<tr>
<td>mem</td>
</tr>
</tbody>
</table>
7.3 Configuring PBS for HPCBP

7.3.1 List of Steps to Configure PBS for HPCBP

The following is the list of steps to configure PBS for use with HPCBP. Each step is described in detail in the following sections. It is important that you enable the HPCBP MOM last, otherwise you will get errors. Each step involving the HPCBP MOM must be performed as root on the HPCBP MOM host.

1. Become root on the HPCBP MOM host
2. Decide on Number of HPCBP MOMs
3. Create HPCBP Service Account
4. Set HPCBP Service Account Password
5. Set HPCBP Service Account Attribute at HPCBP MOM
6. Specify HPCBP Server’s Web Service Address
7. Specify Staging Protocol for HPCBP Jobs
8. Start Staging Protocol
9. Set User Credentials
10. Install Self-signed Certificate
11. Enable HPCBP MOM

7.3.2 Multiple HPC Basic Profile Servers

You must create one HPCBP MOM for each HPCBP Server that the PBS complex will use.
7.3.3 Create HPCBP Service Account

The HPCBP MOM uses a less-privileged user account as the HPCBP service account. This account is used to retrieve status of nodes and jobs from the HPC Basic Profile Server.

The less-privileged user account is similar to an account which is used for creating system services in Windows. Normally these accounts don't have privileges to create files, log in, etc.

All HPCBP Servers can use the same HPCBP service account.

This account is created in three places:

- Create the HPCBP service account on the HPCBP Server host. Create the HPCBP service account inside Active Directory. This account must be a cluster user in the HPC Cluster Manager, so that it can query for the status of all jobs in the HPC Server.
- Create the same user on the PBS server.
- Create the account on the host where the HPCBP MOM runs.

The username and password for this account must be the same in all three places. The HPCBP MOM passes these to the HPC Server when performing any operation, such as submitting a job, getting job status, terminating a job, etc.

7.3.4 Set HPCBP Service Account Password

You must set the HPCBP service account password. This password must be the same on the HPC Server, the PBS server, and the host where the HPCBP MOM runs. Do the following on the PBS server:

1. Log in under the name in the `hpcbp_user_name` attribute
2. Run the `pbs_password` command to set the password for the HPCBP service account.
7.3.5 Set HPCBP Service Account Name Attribute

Format:

```
Qmgr: set node <vnode name> hpcbp_user_name = <user name>
```

For example, to set the name of the account that the HPCBP MOM on the vnode named MyHPCBPMom will use to pbsnobody:

```
Qmgr: set node MyHPCBPMom hpcbp_user_name = pbsnobody
```

7.3.6 Specify HPCBP Server’s Web Service Address

You must set the `hpcbp_webservice_address` attribute for the vnode running the HPCBP MOM to the address of its HPC Basic Profile Server.

The default port number for an HPC Basic Profile Server is 443. If the HPC Basic Profile Server is running on a port other than the default port, you must specify the port number as well.

To set the HPCBP Server’s web service address:

Format:

```
Qmgr: set node <node name> hpcbp_webservice_address = <URL[:port]>
```

For example, to set the `hpcbp_webservice_address` attribute for the vnode named MyHPCBPMom to the location `host1.site1/location1`, using port 8080:

```
Qmgr: set node MyHPCBPMom hpcbp_webservice_address = https://host1.site1:8080/location1
```

For help finding the HPCBP Server’s web service address, see the Microsoft documentation. Microsoft documentation describes how to “commission the HPC Basic Profile Web Service”, at tech-net.microsoft.com/en-us/library/cc947708(WS.10).aspx.
7.3.7 Specify Staging Protocol for HPCBP Jobs

The HPCBP MOM does not perform file staging for HPCBP jobs; it passes the source file path, target file path, protocol, and optional port number to the HPC Basic Profile Server.

The HPC Basic Profile Server uses the protocol and port number that you specify in the hpcbp-stage-protocol HPCBP MOM vnode attribute for staging files in and out. If the protocol uses a non-standard or unassigned port, you must specify the port number.

The default value of the hpcbp-stage-protocol attribute is scp. If the port number is not specified, the HPCBP MOM passes only the protocol to the HPCBP Server.

Supported protocols:

• https
• ftps
• http
• ssh

Format:

Qmgr: set node <vnode name> hpcbp-stage-protocol = <protocol[:port]>

For example, to set the staging protocol to scp and the port number to 34 for the HPCBP MOM on the vnode named MyHPCBPMom:

Qmgr: set node MyHPCBPMom hpcbp-stage-protocol = scp:34
7.3.8 Start Staging Service

The HPCBP Server must be able to use the staging protocol you have chosen when staging data in from all sources and out to all targets. You must start the selected staging service on any machine that will be involved in staging files for an HPCBP job. This may include:

- Submission host(s)
- Host(s) storing data to be staged
- File server(s)

Run the appropriate daemon or server, depending on the setting of the hpcbp_staging_protocol attribute. So, for example, if you chose “ssh”, then run the sshd daemon. If you chose “http”, then run the http server.

7.3.9 Set User Credentials

Each user submitting jobs to an HPC Basic Profile Server must have a password and a username. PBS stores a username and encrypted password for each user. You must make sure that the passwords at the HPC Basic Profile Server and the PBS Server are the same.

- You or the user must use the pbs_password command to set or update the user’s password.

The PBS Server must pass credential information to the HPCBP MOM at the time of job submission. Therefore, one of the following is required:

- You set the single_signon_password_enable server attribute to True via qmgr. Please read section 9.13.1.1.3 “Caveats for...
single_signon_password_enable” on page 700.

1. Give all jobs a bad password hold:
   
   `qhold -h p <job ID>`

2. Set the single_signon_password_enable attribute to True:
   
   `Qmgr: set server single_signon_password_enable = True`

3. Release the bad password hold:
   
   `qrls -h p <job ID>`

• You set single_signon_password_enable to False. The user must use the -Wpwd option to the qsub command to pass credential information to the PBS Server when submitting a job.
7.3.10 Install Self-signed Certificate

You must retrieve the self-signed certificate from the HPCBP Server, and add the certificate to the list of accepted certificate authorities. The HPCBP MOM uses the value of the `hpcbp_webservice_address` attribute to fetch the server-side certificate from the HPC Basic Profile Server.

1. To retrieve the self-signed certificate, run the following commands on the node running the HPCBP MOM:
   
   ```
   echo | openssl s_client -connect <HPCBP Server host-name>:<port> 2>&1 | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > cert.pem
   ```

2. Run `c_rehash` to make known the location of the certificate store to `openssl`:
   
   ```
   # c_rehash
   /usr/share/ssl/certs
   ```

3. Copy the `cert.pem` to the certificate store:
   
   ```
   # cp cert.pem /usr/share/ssl/certs
   ```

4. Run the `c_rehash` command to add the certificate to the list of accepted certificate authorities:
   
   ```
   # c_rehash
   ```

7.3.11 Enable HPCBP MOM

If the `hpcbp_enable` attribute is set to `False` or is unset, the MOM behaves like a standard MOM. If the attribute is set to `True`, the HPCBP features are enabled in the MOM. The HPCBP MOM features must be enabled after all other HPCBP configuration is done.

Format:

```
Qmgr: set node <vnode name> hpcbp_enable = True
```
For example, to enable the HPCBP features in the MOM on the vnode named MyHPCBPMom:

```
Qmgr: set node MyHPCBPMom hpcbp_enable = True
```

### 7.4 Viewing Status of Nodes

You can use the `pbsnodes` command to view the status of the vnode running the HPCBP MOM and each node under the control of the HPCBP Server. Each node under the control of the HPCBP Server will show up as a separate vnode. The following table lists the information available for the node running the HPCBP MOM and each HPCBP node:

#### Table 7-4: Information Available About HPCBP Vnodes

<table>
<thead>
<tr>
<th>PBS Attribute</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mom</td>
<td>For HPCBP nodes, this is the hostname of HPCBP MOM host</td>
<td></td>
</tr>
<tr>
<td>ntype</td>
<td>Node type</td>
<td>For HPCBP MOM host: PBS</td>
</tr>
<tr>
<td>state</td>
<td>State of this vnode</td>
<td></td>
</tr>
<tr>
<td>resources_available.arch</td>
<td>Architecture of this vnode</td>
<td>For HPCBP MOM host: hpcbp</td>
</tr>
<tr>
<td>resources_available.host</td>
<td>Host name of this vnode</td>
<td></td>
</tr>
<tr>
<td>resources_available.mem</td>
<td>Amount of physical memory available on this vnode</td>
<td>For HPCBP MOM host: 0</td>
</tr>
<tr>
<td>resources_available.ncpus</td>
<td>Number of CPUs available on this vnode</td>
<td>For HPCBP MOM host: 0</td>
</tr>
<tr>
<td>resv_enable</td>
<td>Whether reservations are enabled on this vnode</td>
<td>False</td>
</tr>
</tbody>
</table>
7.5 Errors, Logging, and Troubleshooting

7.5.1 HPCBP Server Failure

If the HPCBP MOM gets a notice of a Failed state from the HPC Basic Profile Server while a job is running under the control of the HPCBP Server, the MOM logs an error message to its log file and changes the state of the job to Exiting.

7.5.2 Error During Retrieval of Status

If any problem occurs while the HPCBP MOM is trying to find the status of jobs running at the HPC Basic Profile Server, the HPCBP MOM marks all vnodes controlled by the HPCBP Server as down. The PBS Server processes all jobs running on the vnodes that are marked down and takes appropriate action depending on the server’s node_fail_requeue attribute. The HPCBP MOM continues to poll the HPCBP server periodically and will mark the vnodes as up when the server comes back.

7.5.3 Errors Encountered While Setting HPCBP-specific Attributes

7.5.3.1 Bad Attribute Values

The following table lists the errors encountered while setting HPCBP-specific attributes under certain conditions, and each resulting error message:

<table>
<thead>
<tr>
<th>Error</th>
<th>Condition</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting hpcbp_enable to True</td>
<td>Jobs are running on HPCBP MOM</td>
<td>Illegal attribute or resource value</td>
</tr>
</tbody>
</table>
### 7.5.3.2 OpenSSL Not Present

If the OpenSSL libraries and header files are not present on the system where the MOM is running when you try to enable the HPCBP MOM, it logs an error message in the MOM log file and functions as a standard MOM.

### 7.6 Restrictions and Requirements

- You can run at most one HPCBP MOM on a host.
- You cannot run any other MOM on a host where an HPCBP MOM is running.

<table>
<thead>
<tr>
<th>Error</th>
<th>Condition</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting <code>hpcbp_enable</code> to <code>False</code>, or unsetting it</td>
<td>Jobs are running on HPCBP MOM</td>
<td>Illegal attribute or resource value</td>
</tr>
<tr>
<td>Setting <code>hpcbp_enable</code> to <code>True</code></td>
<td>Any other HPCBP attribute is unset</td>
<td>Illegal attribute or resource value</td>
</tr>
<tr>
<td>Unsetting <code>hpcbp_enable</code></td>
<td>Any other HPCBP attribute is set</td>
<td>Mutually exclusive values for <code>hpcbp_enable</code></td>
</tr>
<tr>
<td>Setting <code>hpcbp_webservice_address</code> to invalid value</td>
<td>Any</td>
<td>Illegal attribute or resource value</td>
</tr>
<tr>
<td>Setting <code>hpcbp_stage_protocol</code> to invalid value</td>
<td>Any</td>
<td>Illegal attribute or resource value</td>
</tr>
</tbody>
</table>
7.7 Advice and Caveats

7.7.1 Differences Between Standard Linux MOM and HPCBP MOM

- The stime attribute in the PBS accounting logs may not represent the exact start time for an HPCBP job. The stime attribute recorded in the PBS accounting logs for an HPC job is the time the job was accepted by the HPC server.
- The HPCBP MOM does not use the pbs_rCP command for staging operations, regardless of whether the PBS_RCP or PBS_SCP environment variables have been set in the configuration file.
- If the PBS scheduler decides to suspend and resume a job running on the HPCBP Server, the HPCBP MOM rejects the request. PBS handles the job using existing PBS behavior for a failed suspend request.

7.7.2 Unsupported Commands

If the user or administrator runs the pbsdsh command for a job running on the HPCBP Server, the HPCBP MOM logs an error message to the MOM file and rejects the job.

7.7.3 Notes

The HPCBP MOM uses the alt_id job attribute to store the HPCBP Server job identifier for a job. Sometimes this job identifier contains white spaces.
The following commands and their API equivalents are not supported for jobs that end up running on the HPCBP Server:

- `qalter`
- `qsig`
- `qmsg`
- `pbsdsh`
- `pbs_rcp`
- `pbs-report`
- `printjob`
- `tracejob`
- `pbs_rsub`
- `pbs_rstat`
- `pbs_rdel`
- `qhold`
- `qrls`
- `qrerun`
7.8 See Also

7.8.1 References

1. OGSA High Performance Computing Profile Working Group (OGSA-HPCP-WG) of the Open Grid Forum
   https://forge.gridforum.org/sf/projects/ogsa-hpcp-wg
   The HPC Basic Profile specification is GFD.114:

2. OGSA High Performance Computing Profile Working Group (OGSA-HPCP-WG) of the Open Grid Forum
   https://forge.gridforum.org/sf/projects/ogsa-hpcp-wg
   The HPC File Staging Profile Version 1.0:
   http://forge.ogf.org/sf/go/doc15024?nav=1

3. OGSA Job Submission Description Language Working Group (JSDL-WG) of the Open Grid Forum
   The JSDL HPC Profile Application Extension, Version 1.0 is GFD 111:
   http://www.ogf.org/documents/GFD.111.pdf

4. OGSA Usage Record Working Group (UR-WG) of the Open Grid Forum
   The Usage Record - Format Recommendation is GFD.98
   http://www.ogf.org/documents/GFD.98.pdf

5. Network Working Group, Uniform Resource Identifier (URI): Generic Syntax
   http://www.rfc-editor.org/rfc/rfc3986.txt
Chapter 8

Provisioning

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

8.1 Introduction

You can configure vnodes so that PBS will automatically install the OS or application that jobs need in order to run on those vnodes. For example, you can configure a vnode that is usually running RHEL to run SLES instead whenever the Physics group runs a job requiring SLES. If a job requires an application that is not usually installed, PBS can install the application in order for the job to run.
You can use provisioning for booting multi-boot systems into the desired OS, downloading an OS to and rebooting a diskless system, downloading an OS to and rebooting from disk, instantiating a virtual machine, etc. You can also use provisioning to run a configuration script or install an application.

### 8.2 Definitions

**AOE**

The environment on a vnode. This may be one that results from provisioning that vnode, or one that is already in place.

**Master Provisioning Script, Master Script**

The script that makes up the provisioning hook.

**Provision**

To install an OS or application, or to run a script which performs installation and/or setup.

**Provisioning Hook**

The hook which performs the provisioning, either by calling other scripts or running commands.

**Provisioning Tool**

A tool that performs the actual provisioning, e.g. SGI Tempo.

**Provisioned Vnode**

A vnode which, through the process of provisioning, has an OS or application that was installed, or which has had a script run on it.

### 8.3 How Provisioning Can Be Used

- Each application requires specific version of OS

  The site runs multiple applications, and each application may be certified to run on a specific OS. In this situation, a job that will run an
application requiring a specific OS requests the OS, and PBS provisions the required OS.

• The site needs differently configured images of the same OS to be loaded at different times

The site has multiple projects, and each project requires the OS to be configured in a different way on a group of hosts. In this situation, PBS provisions groups of hosts with the correct OS image, for the time period needed by each project. The OS image is configured and supplied by the site administrator.

• The entire site needs different OSes at different times of day

The entire site runs one OS during certain hours, and a different OS at other times. For example, the site runs Windows during the day to support users running interactive Excel sessions, and runs Linux at night for batch jobs. The site does not run any PBS MOMs during the day.

• A user reserves multiple vnodes running the same version of an OS

A user may need a specific version of an OS for a period of time. For example, a user needs 5 nodes running RHEL3 from 5pm Friday until 5am Monday.

• The administrator wants to limit the number of hosts that are being provisioned at any one time, for any of the following reasons:
  - The network can become overwhelmed transferring OS images to execution nodes
  - The hosts can draw excessive power if many are powering up at the same time
  - Some sites notify the administrator whenever an execution node goes down, and when several vnodes are provisioned, the administrator is paged repeatedly
8.4 How Provisioning Works

8.4.1 Overview of Provisioning

PBS allows you to create a provisioning hook, which is a hook that is triggered by a provisioning event. When this hook is triggered, it manages the required provisioning on the vnodes to be provisioned. The hook calls a provisioning mechanism such as SGI's Tempo to accomplish the provisioning.

Provisioning can be the following:

- Directly installing an OS or application
- Running a script which may perform setup or installation

PBS allows you to configure each vnode with a list of available AOEs. This list is specified in the vnode’s `resources_available.aoe` resource. Each vnode’s `current_aoe` attribute shows that vnode’s current AOE. The scheduler queries each vnode’s `aoe` resource and `current_aoe` attribute in order to determine which vnodes to provision for each job.

When users submit jobs, they can request a specific AOE for each job. When the scheduler runs each job, it either finds the vnodes that satisfy the job’s requirements, or provisions the required vnodes.

Users can create reservations that request AOEs. Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE.

8.4.1.1 Rebooting When Provisioning

When provisioning a vnode with some AOEs, the vnode must be rebooted as part of the provisioning process. Some OS installations, for example, require rebooting. In this case, the provisioning script must cause the vnode to be rebooted.

When the installation does not require a reboot, the provisioning script does not need to cause the vnode to be rebooted. For example, provisioning with some applications does not require a reboot.
8.4.2 How Vnodes Are Selected for Provisioning

Each job can request at most one AOE. When scheduling the job, PBS looks for vnodes with the requested AOE, as with any other resource. If there are not enough vnodes with the requested AOE, PBS tries to provision vnodes in order to satisfy the job’s requirements.

8.4.2.1 Provisioning Policy

PBS allows a choice of provisioning policies. You set the scheduler’s provision_policy configuration parameter to be either “avoid_provision” or “aggressive_provision”. The default provisioning policy is “aggressive_provision”.

avoid_provision

PBS first tries to satisfy the job’s request from free vnodes that already have the requested AOE instantiated. PBS uses node_sort_key to sort these vnodes.

If it cannot satisfy the job’s request using vnodes that already have the requested AOE instantiated, it does the following:

- PBS uses node_sort_key to select the free vnodes that must be provisioned in order to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them.
- Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

aggressive_provision

PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses node_sort_key to select the vnodes on which to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provi-
sions any that do not have the requested AOE instantiated on them.

### 8.4.2.2 Examples of Vnode Selection

The following examples show how provisioning policy can affect which vnodes are selected for provisioning.

Example 1: 3 vnodes

In `sched_config`:

```plaintext
node_sort_key: “ncpus HIGH”
```

We have 3 nodes as described in the following table:

<table>
<thead>
<tr>
<th>Vnode/Host</th>
<th>Number of CPUs</th>
<th>Current AOE</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>host1</td>
<td>1</td>
<td>aoe1</td>
<td>free</td>
</tr>
<tr>
<td>host2</td>
<td>2</td>
<td>unset</td>
<td>free</td>
</tr>
<tr>
<td>host3</td>
<td>3</td>
<td>aoe2</td>
<td>free</td>
</tr>
</tbody>
</table>

No jobs are running on any of the vnodes.

Case 1: aggressive provisioning

```plaintext
provision_policy: "aggressive_provision"
```

Job submitted with `-lselect=ncpus=1:aoe=aoe1`

In this case, host3 is used to run the job and host3 is provisioned.

Case 2: avoiding provisioning

```plaintext
provision_policy: "avoid_provision"
```

Job submitted with `-lselect=ncpus=1:aoe=aoe1`

In this case, host1 is used to run the job and host1 is not provisioned.

Example 2: 5 vnodes
The following table shows the example configuration:

**Table 8-2: Example Configuration**

<table>
<thead>
<tr>
<th>Vnode/Host</th>
<th>AOE Available</th>
<th>Current AOE</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>aoe1, aoe2</td>
<td>aoe1</td>
<td>busy</td>
</tr>
<tr>
<td>N2</td>
<td>aoe1, aoe2</td>
<td>aoe2</td>
<td>free</td>
</tr>
<tr>
<td>N3</td>
<td>aoe1, aoe2</td>
<td>NULL</td>
<td>free</td>
</tr>
<tr>
<td>N4</td>
<td>aoe1, aoe2</td>
<td>aoe1</td>
<td>free</td>
</tr>
<tr>
<td>N5</td>
<td>aoe1, aoe2</td>
<td>aoe1</td>
<td>free</td>
</tr>
</tbody>
</table>

The vnodes are sorted in the order N1, N2, N3, N4, N5.

A job is submitted with:

```
qsub -lselect=3:ncpus=1:aoe=aoe1 -lplace=scatter
```

The job needs three vnodes with aoe1. Assume that all other requests except that for the AOE can be satisfied by any vnode.

Case 1: aggressive provisioning

The scheduler selects N2, N3 and N4. It has not considered the AOE instantiated on these vnodes. It then provisions N2 and N3 since N2 has a different AOE instantiated on it and N3 is not provisioned yet. N4 is not provisioned, because it has the requested AOE already instantiated.

Case 2: avoiding provisioning

First, the scheduler selects N4 and N5. It does not choose N2 since it has a different AOE instantiated, and it does not choose N3 since it does not have any AOE instantiated. But N4 and N5 together do not satisfy the job's requirement of 3 vnodes.

Second, the scheduler seeks vnodes that if provisioned can satisfy the job’s request. N2 and N3 can each satisfy the job’s request, so it chooses N2, because it comes first in sorted order.

The job runs on N4, N5 and N2. N2 is provisioned.
8.4.2.3 Rules for Vnode Selection for Provisioning

A vnode is not selected for provisioning for the following reasons:

- It does not have the requested AOE available in its list
- It does not have provisioning enabled on it
- It has other running or suspended jobs
- It already has the requested AOE

8.4.2.4 Triggering Provisioning

When a job requires a vnode, and the vnode’s current_aoe attribute is unset, or is set to a different AOE from the one requested, the vnode is provisioned.

8.4.3 Provisioning And Reservations

8.4.3.1 Creating Reservations that Request AOEs

A reservation can request at most one AOE.

When a user creates a reservation that requests an AOE, the scheduler searches for vnodes that can satisfy the reservation. When searching, the scheduler follows the rule specified in the provision_policy scheduling parameter in PBS_HOME/sched_priv/sched_config. See the pbs_sched(8B) manual page.

The vnodes allocated to a reservation that requests an AOE are put in the resv-exclusive state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

8.4.3.2 Submitting Jobs to a Reservation

If a job that requests an AOE is submitted to a reservation, the reservation must request the same AOE.
8.4.3.3 Running a Job in a Reservation Having a Requested AOE

A job can run in a reservation that has requested an AOE, as long as the job fits the following criteria:

- It requests the same AOE as the reservation

If the job has requested no AOE, or an AOE different from that of the reservation, the job is rejected.

8.4.4 How Provisioning Affects Jobs

8.4.4.1 Preemption and Provisioning

A job that has requested an AOE will not preempt another job, regardless of whether the job’s requested AOE matches an instantiated AOE. Running jobs are not preempted by jobs requesting AOE.

8.4.4.2 Backfilling and Provisioning

If the job being backfilled around or the job doing the backfilling share a vnode, a job that has requested an AOE will not play any part in backfilling:

- It will not be backfilled around by smaller jobs.
- It will not be used as the job that backfills around another job.

8.4.4.3 Walltime and Provisioning

A job’s walltime clock is started after provisioning is over.

8.4.4.4 Using qrun

When a job requesting an AOE is run via qrun -H, the following happens:

- If the requested AOE is available on the specified vnodes, those vnodes
are provisioned with the requested AOE

- If the requested AOE is not available on the specified vnodes, the job is held

### 8.4.5 Vnode States and Provisioning

#### 8.4.5.1 States Associated With Provisioning

The following vnode states are associated with provisioning:

**provisioning**

A vnode is in the provisioning state while it is in the process of being provisioned. No jobs are run on vnodes in the provisioning state.

**wait-provision**

There is a limit on the maximum number of vnodes that can be in the provisioning state. This limit is specified in the Server’s `max_concurrent_provision` attribute. If a vnode is to be provisioned, but cannot because the number of concurrently provisioning vnodes has reached the specified maximum, the vnode goes into the `wait-provisioning` state. No jobs are run on vnodes in the `wait-provisioning` state.

**resv-exclusive**

The vnodes allocated to a reservation that requests an AOE are put in the `resv-exclusive` state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.
8.4.5.2 Provisioning Process

The following table describes how provisioning and vnode state transitions interact:

Table 8-3: Vnode State Transitions and Provisioning

<table>
<thead>
<tr>
<th>Event</th>
<th>Starting Vnode State</th>
<th>Ending Vnode State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vnode is selected for provisioning</td>
<td>free</td>
<td>provisioning</td>
</tr>
<tr>
<td>Provisioning on vnode finishes</td>
<td>provisioning</td>
<td>free</td>
</tr>
<tr>
<td>1. Job running on this vnode leaving some resources available</td>
<td>free</td>
<td>free</td>
</tr>
<tr>
<td>2. No job running on this vnode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Job running on this vnode, using all resources</td>
<td>free</td>
<td>job-busy</td>
</tr>
<tr>
<td>Vnode is selected for provisioning, but other vnodes being provisioned have already reached maximum allowed number of concurrently provisioning vnodes</td>
<td>free</td>
<td>wait-provisioning</td>
</tr>
<tr>
<td>This vnode is waiting to be provisioned for a multi-vnode job, and provisioning fails for another of the job’s vnodes</td>
<td>wait-provisioning</td>
<td>free</td>
</tr>
<tr>
<td>Provisioning fails for this vnode</td>
<td>provisioning</td>
<td>offline</td>
</tr>
<tr>
<td>This vnode is waiting to be provisioned, and another vnode finishes provisioning, bringing the total number of provisioning vnodes below the limit specified in max_concurrent_provision</td>
<td>wait-provisioning</td>
<td>provisioning</td>
</tr>
</tbody>
</table>

8.4.5.3 Vnode State When Provisioning Fails

If provisioning fails on a vnode, that vnode is put into the offline state.
If provisioning for a multi-vnode job fails on one vnode, any vnodes in the wait-provisioning state are put into the free state.

### 8.4.5.4 Using the qmgr Command on Vnodes In Process of Provisioning

The following changes cannot be made to a provisioning vnode (a vnode in the provisioning state):

- Changing value of current_aoe vnode attribute
- Modifying resource resources_available.aoe
- Changing the state of the vnode. The qmgr command returns an error if this is attempted.
- Deleting the vnode from the Server. The qmgr command returns an error if this is attempted.

The following can be modified while a vnode is provisioning:

- The Server's max_concurrent_provision attribute
- A provisioning vnode’s provision_enable attribute

The following cannot be set on the Server’s host:

- current_aoe vnode attribute
- provision_enable vnode attribute
- resources_available.aoe

### 8.4.6 Attributes, Resources, and Parameters Affecting Provisioning

#### 8.4.6.1 Host-level Resources

aoe

The built-in aoe resource is a list of AOEs available on a vnode. Case-sensitive. You specify the list of AOEs that can be requested on a vnode by setting the value of resources_available.aoe to that list. Each job can request at most one AOE.
Automatically added to the “resources” line in PBS_HOME/sched_priv/sched_config.
Cannot be modified while a vnode is provisioning.
Non-consumable. Cannot be set on the Server’s host. Can be set only by a Manager.
Format: string_array.
Default: unset.
Python attribute value type: str

8.4.6.2 Vnode Attributes

current_aoe
The current_aoe vnode attribute shows which AOE is currently instantiated on a vnode. Case-sensitive.
At startup, each vnode’s current_aoe attribute is unset. You must set the attribute to the currently instantiated AOE if you want the scheduler to be able to choose vnodes efficiently.
The value of this attribute is set automatically after a vnode is provisioned.
This attribute cannot be modified while a vnode is provisioning.
Cannot be set on the Server’s host. Settable by Manager only; visible to all.
Format: String.
Default: Unset.

provision_enable
This attribute controls whether the vnode can be provisioned. If set to True, the vnode can be provisioned.
Cannot be set on the Server’s host.
Settable by Manager only; visible to all.
Format: Boolean
Default: Unset
8.4.6.3 Server Attributes

max_concurrent_provision
The maximum number of vnodes allowed to be in the process of being provisioned. Settable by Manager only; readable by all. When unset, default value is used. Cannot be set to zero; previous value is retained.

Format: Integer
Default: 5
Python attribute value type: int

8.4.6.4 Hook Attributes

All attributes of the provisioning hook affect provisioning. See “Hook Attributes” on page 453 of the PBS Professional Reference Guide.

8.4.6.5 Scheduler Configuration Parameters

provision_policy
Specifies the provisioning policy to be used. Valid values: avoid_provision, aggressive_provision.

avoid_provision
PBS first tries to satisfy the job’s request from free vnodes that already have the requested AOE instantiated. PBS uses node_sort_key to sort these vnodes.

If it cannot satisfy the job’s request using vnodes that already have the requested AOE instantiated, it does the following:

PBS uses node_sort_key to select the free vnodes that must be provisioned in order to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them.

Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.
aggressive_provision
PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses node_sort_key to select the vnodes on which to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

Default: “aggressive_provision”.

8.5 Configuring Provisioning

8.5.1 Overview of Configuring Provisioning

The administrator configures provisioning attributes, provides a provisioning tool, and writes a provisioning hook. The administrator configures each vnode to be provisioned with a list of AOE resources, where each resource is an AOE that is available to be run on that vnode. These resources are tags that tell the scheduler what can be run on that vnode. The administrator should also inform the scheduler about the current environment on each vnode, by setting the vnode’s current_aoe attribute. It is also necessary to enable provisioning on each vnode to be provisioned and to set provisioning policy at the Server and Scheduler.
8.5.1.1 Steps in Configuring Provisioning

These are the steps that the administrator must take:

1. Provide a provisioning tool such as SGI Tempo. See section 8.5.2 “Provide a Provisioning Tool” on page 614.

2. Prepare each OS, application, or script that is to be used in provisioning. See section 8.5.3 “Prepare Images” on page 615.

3. Configure each vnode to be provisioned with the appropriate resources. See section 8.5.4 “Define aoe Resources” on page 615.

4. Optional: publish each vnode’s current AOE. See section 8.5.5 “Inform Scheduler of Current AOE” on page 616.

5. Write the provisioning hook’s script. See section 8.5.6 “Write the Provisioning Script” on page 617.

6. Create the empty provisioning hook, import the script, and configure the hook. See section 8.5.7 “Create and Configure the Provisioning Hook” on page 619.

7. Configure provisioning policy. See section 8.5.8 “Configure Provisioning Policy” on page 621.

8. Enable provisioning on vnodes. See section 8.5.9 “Enable Provisioning on Vnodes” on page 622.

9. Enable the provisioning hook. See section 8.5.10 “Enable Provisioning Hook” on page 622.

8.5.2 Provide a Provisioning Tool

For each vnode you wish to provision, there must be a provisioning tool that can be used on that vnode. This provisioning tool can either be written into the provisioning hook script, or be a separate script that is called by the provisioning hook script. You can write the provisioning tool yourself, or you can use something like the SGI Tempo cluster management tool. Your provisioning tool may be able to employ network-accessible power control units.
8.5.3 Prepare Images

You must prepare each image, application, or script you will use. Make sure that each is available to the target vnode. For example, if you use a diskless node server, put your images on the diskless node server.

The values for the ncpus and mem resources must be the same for all OS images that may be instantiated on a given vnode.

8.5.4 Define aoe Resources

The aoe resource is of type string_array, and is used to hold the names of the AOE{s available at each vnode. This resource is not consumable. This resource is unset by default, and by default is added to the resources line in PBS_HOME/sched_priv/sched_config. See “Built-in Resources” on page 350 of the PBS Professional Reference Guide. The aoe resource is visible to all, but settable by the PBS Manager and Operator only.

The scheduler must be able to find out which AOE{s can be run on which vnodes. To tag each vnode with the AOE{s that can run on it, set that vnode’s resources_available.aoe attribute to the list of available AOE{s. For example, if vnode V1 is to run RHEL and SLES, and the hook script will recognize rhel and sles, set the vnode’s resources_available.aoe attribute to show this:

```
Qmgr: set node V1 resources_available.aoe = “rhel, sles”
```

It is recommended that you make a list of all of the AOE{s that may be used in provisioning in your PBS complex. The list is to facilitate script writing and resource configuration. Each entry in this list should contain at least the following information:

- Full description of the AOE
- Resource name of the AOE
- Vnodes that are to run the AOE
- Location where script should look for the AOE
For example, the list might look like the following table:

### Table 8-4: Example AOE List

<table>
<thead>
<tr>
<th>Description</th>
<th>Resource Name</th>
<th>Vnodes</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuSE SLES 9 64-bit</td>
<td>sles9</td>
<td>mars, jupiter, neptune, pluto</td>
<td>imageserver.example.com:/images/sles9-image</td>
</tr>
<tr>
<td>SuSE SLES 10 64-bit</td>
<td>sles10</td>
<td>mars, jupiter, pluto</td>
<td>imageserver.example.com:/images/sles10-image</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux 5 32-bit</td>
<td>rhel5</td>
<td>luna, aitne, io</td>
<td>imageserver.example.com:/images/rhel5-image</td>
</tr>
<tr>
<td>Windows Server 2008 32-bit</td>
<td>winsrv8</td>
<td>luna, aitne, io</td>
<td>\WinServer\C:\images\winsrv8</td>
</tr>
</tbody>
</table>

### 8.5.5 Inform Scheduler of Current AOE

Each vnode has an attribute called `current_aoe` which is used to tell the scheduler what the vnode’s current AOE is. This attribute is unset by default. The attribute is of type `string`. It is visible to all, but settable by the PBS Manager only.

You can set this attribute on each vnode that will be used in provisioning. Set it to the value of the AOE that is currently instantiated on the vnode. So for example, using the table in section 8.5.4 “Define aoe Resources” on page 615, if vnode pluto is running 64-bit SuSE SLES 10, set `current_aoe` to `sles10`:

```omgr
Qmgr: set node pluto current_aoe = sles10
```

When PBS provisions a vnode with a new AOE, the PBS Server sets the value of `current_aoe` to the new AOE.
If PBS cannot provision a vnode with the desired AOE, it marks the vnode offline and unsets the value of current_aoe.

8.5.6 Write the Provisioning Script

You create the provisioning hook using a provisioning script which must manage all provisioning, either directly, or indirectly by calling other scripts. The script in the hook is the master provisioning script.

The script that does the provisioning must have the logic needed to provision the specified vnode with the specified AOE.

There are two types of provisioning. One is when the vnode is rebooted after installing/uninstalling the OS/application or running the script. The other is when the vnode is not rebooted after installing/uninstalling the OS/application or running the script.

The master provisioning script must meet the following requirements:

• Written in Python
• Arguments to the script are the vnode name and the AOE name
• If the vnode must be rebooted for provisioning, the provisioning script must cause the target vnode to be rebooted
• Must indicate success using the correct return value:
  - Return pbs.event.accept(0) if provisioning is successful and the vnode is rebooted
  - Return pbs.event.accept(1) if provisioning is successful and the vnode is not rebooted
• Must indicate failure to PBS by using pbs.event.reject()
• If the master provisioning script calls other scripts, it must wait for them to finish before returning success or failure to PBS
8.5.6.1 Arguments to Master Script

The arguments to the master script are the following:

- Name of vnode to be provisioned
  
  Supplied to the hook via the PBS provision event object, as `pbs.event.vnode`

- Name of AOE to be instantiated on the target vnode

  Supplied to the hook via the PBS provision event object, as `pbs.event.aoe`

These values can be passed to scripts that are called by the master script.

8.5.6.2 Return Values

The master script must indicate to PBS whether it succeeded or failed in a way that PBS can understand.

8.5.6.2.1 Success

By default, `pbs.event.accept()` returns zero. The script must return different values for successful provisioning, depending on whether the vnode is rebooted:

- If provisioning is successful and the vnode is rebooted, the script must return 0 (zero) to PBS via `pbs.event.accept(0)`.
- If provisioning is successful and the vnode is not rebooted, the script must return 1 (one) to PBS via `pbs.event.accept(1)`.

8.5.6.2.2 Failure

If provisioning fails, the script must use `pbs.event.reject()` to indicate failure. By default, `pbs.event.reject()` returns 255. To return another failure code, use the following:

```
pbs.event.reject(error message, error code)
```

where error code is any number between 2 and 255. Returning an error code in `pbs.event.reject()` is optional.
8.5.6.3 Master Script Calls Subscript

Often, the master script (the hook script) calls another script, depending on the provisioning required. The subscript does the actual provisioning of the target vnode with the requested AOE. In this case, the master script must wait for the subscript to return and indicate success or failure. The master script then propagates the result to PBS.

Example of a fragment of a master script calling a subscript:

```
return_value = os.system("/var/vendor/vendor_prov.sh "
  <arguments to vendor_prov.sh>)
```

8.5.7 Create and Configure the Provisioning Hook

The provisioning hook causes any provisioning to happen. The provisioning hook is a Python script which either does the provisioning directly or calls other scripts or tools. Typically the provisioning hook calls other scripts, which do the actual work of provisioning. For complete information on writing hooks, see Chapter 6, "Hooks", on page 461.

You can have at most one provisioning hook. Do not attempt to create more than one provisioning hook.

In the steps that follow, we use as examples a provisioning hook named “Provision_Hook”, and an ASCII script named “master_provision.py”.

8.5.7.1 Create the Hook

To create the provisioning hook:

```
Qmgr: create hook <hook name>
```

For example, to create a provisioning hook called Provision_Hook:

```
Qmgr: create hook Provision_Hook
```
8.5.7.2 Import the Hook Script

If the hook script is called “master_provision.py”, and it is ASCII, and it is located in /root/data/, importing the hook script looks like this:

```
Qmgr: import hook Provision_Hook application/x-python default /root/data/master_provision.py
```

See section 6.6.4 “Importing Hooks” on page 475 for more about importing hooks.

8.5.7.3 Configure the Hook Script

8.5.7.3.1 Set Event Type

The event type for the provisioning hook is called “provision”. To set the event type:

```
Qmgr: set hook Provision_Hook event = provision
```

Do not try to assign more than one event type to the provisioning hook.

8.5.7.3.2 Set Alarm Time

The default alarm time for hooks is 30 seconds. This may be too short for a provisioning hook. You should set the alarm time to a value that is slightly more than the longest time required for provisioning. Test provisioning each AOE, and find the longest time required, then add a small amount of extra time. To set the alarm time:

```
Qmgr: set hook Provision_Hook alarm = <number of seconds required>
```
8.5.8 Configure Provisioning Policy

8.5.8.1 Set Maximum Number of Concurrently Provisioning Vnodes

The value of the Server’s `max_concurrent_provision` attribute specifies the largest number of vnodes that can be in the process of provisioning at any time. The default value of this attribute is 5. Set the value of this attribute to the largest number of vnodes you wish to have concurrently provisioning. See section 8.4.6.3 “Server Attributes” on page 612 for more information on the attribute.

8.5.8.1.1 Considerations

You may wish to limit the number of hosts that can be in the process of provisioning at the same time:

- So that the network isn’t overwhelmed transferring OS images to execution nodes
- So the hosts won't draw excessive power when powering up at the same time

Many sites have tools that notify them when an execution node goes down. You may want to avoid being paged every time an execution node is provisioned with a new AOE.

8.5.8.2 Set Scheduling Policy

When a job is scheduled to be run, and the job requests an AOE, PBS can either try to fit the job on vnodes that already have that AOE instantiated, or it can choose the vnodes regardless of AOE. Choosing regardless of AOE is the default behavior; the assumption is that the chances of finding free vnodes that match all the requirements including that of the requested AOE are not very high.

Provisioning policy is controlled by the `provision_policy` scheduling parameter in `PBS_HOME/sched_priv/sched_config`. This parameter is a string which can take one of two values: `avoid_provision` or `aggressive_provision`. If you want PBS to try first to use vnodes whose
AOEs already match the requested AOE, set `provision_policy` to `avoid_provision`. If you want PBS to choose vnodes regardless of instantiated AOE, set it to `aggressive_provision`.

For details about the `provision_policy` parameter, see section 8.4.6.3 “Server Attributes” on page 612.

For jobs that do not request an AOE, `node_sort_key` is used to choose vnodes.

### 8.5.9 Enable Provisioning on Vnodes

PBS will provision only those vnodes that have provisioning enabled. Provisioning on each vnode is controlled by its `provision_enable` attribute. This attribute is Boolean, with a default value of `False`. You enable provisioning on a vnode by setting its `provision_enable` attribute to `True`.

This attribute cannot be set to `True` on the Server’s host.

See section 8.4.6.2 “Vnode Attributes” on page 611 for details about the `provision_enable` vnode attribute.

It is recommended that you set `provision_enable` on the HPCBP MOM host to `False`.

### 8.5.10 Enable Provisioning Hook

The last step in configuring provisioning is enabling the provisioning hook. The provisioning hook is enabled when its enabled attribute is set to `True`.

To set the enabled attribute to `True` for the provisioning hook named `Provision_Hook`:

```
Qmgr: set hook Provision_Hook enabled = True
```
8.6 Viewing Provisioning Information

8.6.1 Viewing Provisioning Hook Contents

To see the contents of the provisioning hook, export them:

`export hook <hook name> application/x-python default <output-path>/ <output-filename>`

For example, if the provisioning hook is named `Provision_Hook`, and you wish to export the contents to `/usr/user1/hook_contents`:

```
Qmgr: export hook Provision_Hook application/x-python default /usr/user1/hook_contents
```

8.6.2 Viewing Provisioning Hook Attributes

To view the provisioning hook’s attributes, use the `list hook` option to the `qmgr` command:

```
qmgr -c "list hook <hook name>"
```
8.6.3 Printing Provisioning Hook Creation Commands

To print the provisioning hook’s creation commands, use the `print hook` option to the `qmgr` command:

```
qmgr -c "p hook"
#
# Create hooks and set their properties.
#
#
# Create and define hook my_prov_hook
#
create hook my_prov_hook
set hook my_prov_hook type = site
set hook my_prov_hook enabled = True
set hook my_prov_hook event = provision
set hook my_prov_hook user = pbsadmin
set hook my_prov_hook alarm = 30
set hook my_prov_hook order = 1
import hook my_prov_hook application/x-python base64 -c2xzbGwK
```

8.6.4 Viewing Attributes and Resources Affecting Provisioning

8.6.4.1 Server Attributes

To see the Server attributes affecting provisioning, print the Server’s information using the `qmgr` command:

```
qmgr -c “print server”
```
You will see output similar to the following:

```bash
# qmgr
Max open servers: 49
Qmgr: p s
#
# Create queues and set their attributes.
#
# Create and define queue workq
#
create queue workq
set queue workq queue_type = Execution
set queue workq enabled = True
set queue workq started = True
#
# Set server attributes.
#
set server scheduling = True
set server default_queue = workq
set server log_events = 511
set server mail_from = adm
set server resv_enable = True
set server node_fail_requeue = 310
set server pbs_license_min = 0
set server pbs_license_max = 2147483647
set server pbs_license_linger_time = 3600
set server license_count = "Avail_Global:0
    Avail_Local:256 Used:0 High_Use:0"
set server max_concurrent_provision = 5
```
8.6.4.2 Viewing Vnode Attributes and Resources

To see vnode attributes and resources affecting provisioning, use the -a option to the pbsnodes command:

```
pbsnodes -a
host1
    Mom = host1
    ntype = PBS
    state = free
    pcpus = 2
    resources_available.aoe = osimage1, osimage2
    resources_available.arch = linux
    resources_available.host = host1
    resources_available.mem = 2056160kb
    resources_available.ncpus = 2
    resources_available.vnode = host1
    resources_assigned.mem = 0kb
    resources_assigned.ncpus = 0
    resources_assigned.vmem = 0kb
    resv_enable = True
    sharing = default_shared
    provision_enable = True
    current_aoe = osimage2
```
8.7 Requirements and Restrictions

8.7.1 Site Requirements

8.7.1.1 Single-vnode Hosts Only

PBS will provision only single-vnode hosts. Do not attempt to use provisioning on hosts that have more than one vnode.

8.7.1.2 Provisioning Tool Required

For each vnode you wish to provision, there must be a provisioning tool that can be used on that vnode. Examples of provisioning tools are the following:

- The SGI Tempo cluster management tool
- Dual boot system
- Network-accessible power control units

8.7.1.3 Single Provisioning Hook Allowed

The PBS Server allows only one provisioning hook. If you have an existing provisioning hook and you import a provisioning script, that script will become the contents of the hook, whether or not the hook already has a script. The new script will overwrite the existing provisioning hook script.

8.7.1.4 Provisioning Hook Cannot Have Multiple Event Types

The provisioning hook cannot have more than one event type.

8.7.1.5 AOE Names Consistent Across Complex

Make AOE names consistent across the complex. The same AOE should have the same name everywhere.
8.7.2 Usage Requirements

8.7.2.1 Restriction on Concurrent AOE s on Vnode

Only one AOE can be instantiated at a time on a vnode.

Only one kind of aoe resource can be requested in a job. For example, an acceptable job could make the following request:

-1 select=1:ncpus=1:aoe=suse+1:ncpus=2:aoe=suse

8.7.2.2 Vnode Job Restrictions

A vnode with any of the following jobs will not be selected for provisioning:

- One or more running jobs
- A suspended job
- A job being backfilled around

8.7.2.3 Vnode Reservation Restrictions

A vnode will not be selected for provisioning for job MyJob if the vnode has a confirmed reservation, and the start time of the reservation is before job MyJob will end.

A vnode will not be selected for provisioning for a job in reservation R1 if the vnode has a confirmed reservation R2, and an occurrence of R1 and an occurrence of R2 overlap in time and share a vnode for which different AOE s are requested by the two occurrences.

8.7.2.4 Hook Script and AOE Must Be Compatible

The requested AOE must be available to the vnode to be provisioned. The following must be True:

- The AOE must be in the list of available AOE s for the vnode
- Each AOE listed on a vnode must be recognized by the provisioning
hook script.
• The vnode must have provisioning enabled

8.7.2.5 Provisioning Hook Must Be Ready

• The provisioning hook must obey the following rules:
  - It must exist
  - It must have a Python script imported
  - It must be enabled
  - It must be designed to invoke an external script or command for AOE that are to be used

8.7.2.6 Server Host Cannot Be Provisioned

The Server host cannot be provisioned: a MOM can run on the Server host, but that MOM’s vnode cannot be provisioned. The provision_enable vnode attribute, resources_available.aoe, and current_aoe cannot be set on the Server host.

8.7.2.7 PBS Attributes Not Available to Provisioning Hook

The provisioning hook cannot operate on PBS attributes except for the following:
• The name of the vnode to be provisioned: pbs.event.vnode
• The AOE to be instantiated: pbs.event.aoe

8.7.2.8 avoid_provision Incompatible with smp_cluster_dist

The avoid_provision provisioning policy is incompatible with the smp_cluster_dist scheduling scheduler configuration parameter. If a job requests an AOE, the avoid_provision policy overrides the behavior of smp_cluster_dist.
8.8 Defaults and Backward Compatibility

By default, PBS does not provide provisioning. You must configure PBS to provide provisioning.

8.9 Example Scripts

8.9.1 Sample Master Provisioning Hook Script With Explanation

We show a sample provisioning hook script, and an explanation of what the script does. For readability, the sample script is a master script calling two subscripts.

This provisioning hook allows two kinds of provisioning request:

- For the application AOE named “App1”, via the script app_prov.sh
  The app_prov.sh script does not reboot the vnode
- For other provisioning, via the vendor-provided provisioning shell script vendorprov.sh
  The vendorprov.sh script reboots the vnode
8.9.1.1 Sample Master Provisioning Hook Script

```python
import pbs                                     (1)
import os                                      (2)

e = pbs.event()                                (3)
vnode = e.vnode                                (4)
aoe = e.aoe                                    (5)

if (aoe == "App1"):                            (6)
    appret = os.system("/var/user/app_prov.sh "+ vnode + " " + aoe");                  (7)
    if appret != 1:                             (8)
        e.reject("Provisioning without reboot failed", 210)                        (9)
    else:
        e.accept(1)                             (10)

ret = os.system("/var/vendor/vendorprov.sh "+ vnode + " " + aoe");                  (11)

if ret != 0:                                  (12)
    e.reject("Provisioning with reboot failed", 211)                       (13)
else:
    e.accept(0)                                (14)
```

8.9.1.2 Explanation of Sample Provisioning Hook Script

- Lines 1 and 2 import the \texttt{pbs} and \texttt{os} modules.
- Line 3 puts the PBS provisioning event into the local variable named \texttt{"e"}.
- Lines 4 and 5 store the target vnode name and the name of the AOE to
be instantiated on the target vnode in local variables.

- Line 6 checks whether provisioning of the application AOE named “App1” is requested.

- Line 7 is where the actual code to do non-rebooting provisioning could go. In this example, we call a subscript, passing the name of the target vnode and the requested AOE, and storing the return value in “appret”. The non-rebooting provisioning subscript should return 1 on success.

- Line 8 checks whether non-rebooting provisioning via app_prov.sh succeeded.

- Line 9 returns the error code 210 and an error message to PBS if app_prov.sh failed.

- Line 10 returns 1 via pbs.event.accept(1) if non-rebooting provisioning succeeded.

- Line 11 calls the vendor-supplied script that is responsible for doing rebooting provisioning whenever “App1” is not the AOE. The name of the target vnode and the requested AOE are passed to this script. The vendor-supplied script should expect these two arguments. The return value from this script is stored in the variable named “ret”.

- Line 12 checks whether rebooting provisioning via the vendor-supplied script vendorprov.sh was successful.

- Line 13: If the return value is anything but zero (success), the provisioning hook script passes the error code 211 back to PBS, along with an error message.

- Line 14 returns success to PBS via pbs.event.accept(0) and the master script exits.
8.9.2 Sample Master Provisioning Hook Script Calling Tempo

The following is a master provisioning hook script that calls SGI Tempo:

```python
# -*- coding: utf-8 -*-
import pbs
import os

e = pbs.event()
vnode = e.vnode
aoe = e.aoe

if (aoe=="App1"):
    ret = os.system("/root/osprov/application.sh " + vnode + " " + aoe);
    if ret != 0:
        e.reject("Non-reboot provisioning failed",ret)
    else:
        e.accept(1)

ret = os.system("/root/osprov/sgi_provision.sh " + vnode + " " + aoe);
if ret != 0:
    e.reject("Reboot provisioning failed",ret)
else:
    e.accept(0)
```
Here is the provisioning script named sgi_provision.sh:

```sh
#! /bin/sh

scriptdir=`dirname $0`
pid=$$
ERR=255

test -f /etc/profile.d/mgr.sh && . /etc/profile.d/mgr.sh
if [ $? -ne 0 ]; then
    test -f /etc/profile.d/cmgr.sh && . /etc/profile.d/cmgr.sh
fi

# do `rpm -qa |grep agnostic` to check version of installed agnostic
CMAGNOSTIC_PATH_CMIMG="/usr/sbin"
CMAGNOSTIC_PATH_CMNDE="/usr/bin"

if [ ! -x "${CMAGNOSTIC_PATH_CMIMG}/cmimage" ]; then
    myecho "could not find executable ${CMAGNOSTIC_PATH_CMIMG}/cmimage"
    exit $ERR
fi

if [ ! -x "${CMAGNOSTIC_PATH_CMNDE}/cmnodes" ]; then
    myecho "could not find executable ${CMAGNOSTIC_PATH_CMNDE}/cmnodes"
    exit $ERR
fi

function myecho
{
    echo -e "$pid: $*
```
if [ $# -lt 2 ]; then
    myecho "syntax: $0 <machine> <aoe name>"
    exit $ERR
fi

myecho "starting: $0 $*

machine=$1
aoe_name=$2

hostname=`hostname`
if [ "$machine" = "$hostname" ]; then
    myecho "Cannot reboot own machine. Please provide another machine name"
    exit $ERR
fi

# check hostname in the list of compute nodes
vnodes=`${CMAGNOSTIC_PATH_CMNDE}/cmnodes --list --group compute`
if [ $? -ne 0 ]; then
    myecho "${CMAGNOSTIC_PATH_CMNDE}/cmnodes returned error"
    exit $ERR
fi

echo -e "$vnodes" | grep "^${machine}$"
if [ $? -ne 0 ]; then
    myecho "Vnode $machine is not known to me"
    exit $machine
fi
#check aoe name in list of cmimages available
aoes=`${CMAGNOSTIC_PATH_CMIMG}/cmimage --images | awk
    '{print $1}' | awk
-F"image=" '{print $2}'``
if [ $? -ne 0 ]; then
    myecho "${CMAGNOSTIC_PATH_CMIMG}/cmimage returned error"
    exit $ERR
fi
echo -e "$aoes" | grep "^${aoe_name}$"
if [ $? -ne 0 ]; then
    myecho "AOE/image $aoe_name is not known to me"
    exit $ERR
fi

curr_image=`${CMAGNOSTIC_PATH_CMIMG}/cmimage --running
    --node "$machine" | awk
-F"image=" '{print $2}' | awk '{print $1}'``
if [ "$cur_image" = "$aoe_name" ]; then
    myecho "AOE/image $aoe_name already running on $machine"
    exit 0
fi

#okay all set now, kick off provisioning
${CMAGNOSTIC_PATH_CMIMG}/cmimage --set --node
    ${machine} --image ${aoe_name}
if [ $? -ne 0 ]; then
    myecho "setting of image $aoe_name failed"
    exit $ERR
8.9.3 Sample Script Set

This is a set of example Linux scripts designed to work together. They are the following:

provision_hook.py
  This is the script for the provisioning hook. It calls the master provisioning script.

provision_master.py:
This is the master provisioning script. It is responsible for rebooting the machine being provisioned. It calls update_grub.sh to update the current AOE.

update_grub.sh

This shell script updates the linux grub.conf file and sets the value for current_aoe after the reboot.

The update_grub.sh script must be modified according to the grub configuration of the system in question before being run.

8.9.3.1 Provisioning Hook Script

provision_hook.py:

```python
import pbs
import os

e = pbs.event()
vnodae = e.vnode
aoe = e.aoe
#print "vnode:" + vnode
#print "AOE:" + aoe

if (aoe=="App1"):
    print "Provisioning an application"
    e.accept(1)

ret = os.system("python /root/provision_master.py " +
    vnode + " " + aoe + " " + "lin");
#print "Python top level script returned " + str(ret)
if ret != 0:
    e.reject("Provisioning failed",ret)
else:
    e.accept(0)
```
8.9.3.2 Master Provisioning Script

provision_master.py:

```
#!/usr/bin/python

#--------------------
# success : 0
# failure : 1
#--------------------
# win_or_lin == 1 : windows
# win_or_lin == 0 : linux
#--------------------
# 1 is TRUE
# 0 is FALSE
#--------------------

import sys
import os

vnode = sys.argv[1]
aoe = sys.argv[2]
win_or_lin = sys.argv[3]

print vnode, aoe

if not aoe.find('win'):
    print "aoe is win"
    isvnodewin = 1
else:
    print "aoe is *nix"
    isvnodewin = 0
```
print "win_or_lin = [", win_or_lin, "]" 

if (win_or_lin == "win"):
    print "entering window server"
    if isvnodewin:
        #------------ WINDOWS -> WINDOWS
        ret = os.system("pbs-sleep 05")
        #------------ WINDOWS -> WINDOWS

else:
    #------------ WINDOWS -> LINUX
    ret = os.system("pbs-sleep 05")
    #------------ WINDOWS -> LINUX

ret = os.system("pbs-sleep 45")
print "Pinging machine until it is up..."
timeout = 120
ticks = 0

while 1:
    ret = os.system("ping -c 1 -i 5 " + vnode + " -w 10 > /dev/null 2>&1")
    if not ret:
        print "that machine is now up"
        exit(0)

    ticks = ticks + 1
    print "ticks = ", ticks
    if ticks > timeout:
        print "exit ticks = ", ticks
        print "that machine didn’t come up after 2 mins,FAIL"
exit(1)

else:
    print "entering linux server"
    if isvnodewin:
        #------------ LINUX -> WINDOWS
        ret = os.system("sleep 05")
        #------------ LINUX -> WINDOWS

    else:
        #------------ LINUX -> LINUX

        ret = os.system("scp -o StrictHostKeyChecking=no /root/update_grub.sh " + vnode + ":/root > /dev/null 2>&1")
        if ret != 0:
            print "scp failed to copy"
            exit(1)

        ret = os.system("/usr/bin/ssh -o StrictHostKeyChecking=no " + vnode + " /"/root/update_grub.sh " + vnode + " " + aoe + " 1 " + " > /dev/null 2>&1")
        if ret != 0:
            print "failed to run script"
            exit(1)

        ret = os.system("/usr/bin/ssh -o StrictHostKeyChecking=no " + vnode + " /"reboot"" + " > /dev/null 2>&1")
        if ret != 0:
            print "failed to reboot that machine"
            exit(1)
#----------- LINUX -> LINUX

ret = os.system("sleep 45")
print "Pinging machine until it is up..."
timeout = 120
ticks = 0

while 1:
    ret = os.system("ping -c 1 -i 5 " + vnode + " -w 10 > /dev/null 2>&1")
    if not ret:
        print "that machine is now up"
        exit(0)

    print "ticks = ", ticks
ticks = ticks + 1
    if ticks > timeout:
        print "That machine didn't come up after 2 mins. FAIL"
        exit(1)
8.9.3.3 Grub Update Shell Script

update_grub.sh:

```bash
#!/bin/sh

if [ $# -lt 2 ]; then
    echo "syntax: $0 <machine ip> <aoe name>"
    exit 1
fi

machine=$1
aoe_name=$2

menufile="/boot/grub/grub.conf"
if [ ! -f "$menufile" ]; then
    echo "grub.conf file not found. $machine using grub bootloader?"
    exit 1
fi

link=`ls -l $menufile | cut -c1`
if [ "$link" = "l" ]; then
    menufile=`ls -l $menufile | awk -F"-> " '{print $2}'`
    echo "Found link file, original file is $menufile"
fi

titles=`cat $menufile | grep title | awk -F"title" '{print $2}' | sed 's/[^\t]\t//g'`
lines=`echo -e "$titles" | wc -l`

found_aoe_index=-1
count=0
```
while [ $count -lt $lines ]

do
    lineno=`expr $count + 1`
    title=`echo -e "$titles" | head -n $lineno | tail -n 1`
    if [ "$aoe_name" = "$title" ]; then
        found_aoe_index=$count
    fi
    count=`expr $count + 1`
done

if [ $found_aoe_index = -1 ]; then
    echo "Requested AOE $aoe_name is not found on machine $machine"
    exit 2
else
    new_def_line="default=$found_aoe_index"
    def_line=`cat $menufile | grep "^default="`
    echo "new_def_line=$new_def_line"
    echo "def_line=$def_line"
    echo "menufile=$menufile"

    cp $menufile /boot/grub/grub.conf.backup
    cat $menufile | sed "s/^$def_line/$new_def_line/g" > grub.out
    if [ -s grub.out ]; then
        mv grub.out $menufile
    else
        exit 1
    fi
fi
service pbs stop

exit 0

8.10 Advice and Caveats

8.10.1 Using Provisioning Wisely

It is recommended that when using provisioning, you set PBS up so as to prevent things such as the following:

- User jobs not running because vnodes used in a reservation have been provisioned, and provisioning for the reservation job will take too long
- Excessive amounts of time being taken up by provisioning from one AOE to another and back again

In order to avoid problems like the above, you can do the following to keep specific AOE requests together:

- For each AOE, associate a set of vnodes with a queue. Use a hook to move jobs into the right queues.
- Create a reservation requesting each AOE, then use a hook to move jobs requesting AOE into the correct reservation.
8.10.1.1 Preventing Provisioning

You may need to prevent specific users or groups from using provisioning. You can use a job submission, job modification, or reservation creation hook to prevent provisioning. For more about hooks, see Chapter 6, "Hooks", on page 461. The following is an example of a hook script to prevent USER1 from provisioning:

```python
import pbs
import re

#--- deny user access to provisioning

e = pbs.event()
j = e.job   ### Use e.resv to restrict provisioning in resv
who = e.requestor

unallow_ulist = ['USER1']

if who not in unallow_ulist
    e.accept(0)

#User request AOE in select?
if j.Resource_List['select'] != None:
    s = repr(j.Resource_List['select'])
    if re.search('aoe=', s) != None:
        pbs.logmsg(pbs.LOG_DEBUG, 'User %s not allowed to provision' % (who))
        e.reject('User not allowed to provision')

#User request AOE?
if j.Resource_List['aoe'] != None:
    pbs.logmsg(pbs.LOG_DEBUG, 'User %s not allowed to
```
provision" % (who))
    e.reject("User not allowed to provision")
    e.accept(0)

8.10.2 Prevent Provisioning on HPCBP MOM Host

To prevent the HPCBP MOM’s vnode from being provisioned, set the provision_enable attribute to False on this vnode.

8.10.3 Allow Enough Time in Reservations

If a job is submitted to a reservation with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.

8.11 Errors and Logging

8.11.1 Errors

A vnode is marked offline if:

- Provisioning fails for the vnode
- The AOE reported by the vnode does not match the requested AOE after the provisioning script finishes

A vnode is not marked offline if provisioning fails to start due to internal errors in the script.
8.11.2 Logging

8.11.2.1 Accounting Logs

For each job and reservation, an accounting log entry is made whenever provisioning starts and provisioning ends. Each such log entry contains a list of the vnodes that were provisioned, the AOE that was provisioned on these vnodes, and the start and end time of provisioning.

The accounting log entry for the start of provisioning is identified by the header “P”, and the entry for the end of provisioning is identified by the header “p”.

Example:

Printed when job starts provisioning:

```
“01/15/2009 12:34:15;P;108.mars;user=user1 group=group1
  jobname=STDIN queue=workq prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1
  provision_event=START start_time=1231928746”
```

Printed when job stops provisioning:

```
“01/15/2009 12:34:15;p;108.mars;user=user1 group=group1
  jobname=STDIN queue=workq prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1
  provision_event=END status=SUCCESS end_time=1231928812”
```

Printed when provisioning for job failed:

```
“01/15/2009 12:34:15;p;108.mars;user=user1 group=group1
  jobname=STDIN queue=workq prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1
  provision_event=END status=FAILURE end_time=1231928812”
```
8.11.2.2 Server Logs

8.11.2.2.1 Messages Printed at Log Event Class 0x0080

“vnode <vnode name>: Vnode offline since it failed provisioning”
“vnode <vnode name>: Vnode offline since server went down during provisioning”
“Provisioning for Job <job id> succeeded, running job”
“Job failed to start provisioning”
“Provisioning for Job <job id> failed, job held”
“Provisioning for Job <job id> failed, job queued”

8.11.2.2.2 Messages Printed at Log Event Class 0x0100

“Provisioning of Vnode <vnode name> successful”
“Provisioning of <vnode name> with <AOE name> for <job ID> failed, provisioning exit status=<number>”
“Provisioning of <vnode name> with <aoe name> for <job id> timed out”
“Provisioning vnode <vnode> with AOE <AOE> started successfully”
“provisioning error: AOE mis-match”
“provisioning error: vnode offline”

8.11.2.2.3 Messages Printed at Log Event Class 0x0002

“Provisioning hook not found”

8.11.2.2.4 Messages Printed at Log Event Class 0x0001

“Provisioning script recompilation failed”
8.11.2.3 Scheduler Logs

8.11.2.3.1 Messages Printed at Log Event Class 0x0400

Printed when vnode cannot be selected for provisioning because requested AOE is not available on vnode:

“Cannot provision, requested AOE <aoe-name> not available on vnode”

Printed when vnode cannot be selected for provisioning because vnode has running or suspended jobs, or the reservation or job would conflict with an existing reservation:

“Provision conflict with existing job/reservation”

Printed when vnode cannot be selected for provisioning because provision_enable is unset or set False on vnode:

“Cannot provision, provisioning disabled on vnode”

Printed when job cannot run because server is not configured for provisioning:

“Cannot provision, provisioning disabled on server”

Printed when multiple vnodes are running on the host:

“Cannot provision, host has multiple vnodes”

Printed when vnodes are sorted according to avoid_provision policy:

“Re-sorted the nodes on aoe <aoe name>, since aoe was requested”

8.11.2.3.2 Messages Printed at Log Event Class 0x0100

Printed when a vnode is selected for provisioning by a job:

“Vnode <vnode name> selected for provisioning with <AOE name>”
8.11.3 Error Messages

Printed when vnode is provisioning and current_aoe is set or unset or resources_available.aoe is modified via qmgr:

"Cannot modify attribute while vnode is provisioning"

Printed when qmgr is used to change state of vnode which is currently provisioning:

"Cannot change state of provisioning vnode"

Printed when vnode is deleted via 'qmgr > delete node <name>' while it is currently provisioning:

"Cannot delete vnode if vnode is provisioning"

Printed when provision_enable, current_aoe or resources_available.aoe are set on host running PBS server and scheduler:

"Cannot set provisioning attribute on host running PBS server and scheduler"

Printed when current_aoe is set to an AOE name that is not listed in resources_available.aoe of the vnode:

"Current AOE does not match with resources_available.aoe"

Printed when an event of a hook is set to 'provision' and there exists another hook that has event 'provision':

"Another hook already has event 'provision', only one 'provision' hook allowed"

Printed when qsub has -laoe and -lselect=aoe:

"-lresource= cannot be used with "select" or "place", resource is: aoe"

Job comment printed when job fails to start provisioning:

"job held, provisioning failed to start"
Printed when job is submitted or altered so that it does not meet the requirements that all chunks must request same AOE, and this AOE must match that of any reservation to which the job is submitted:

“Invalid provisioning request in chunk(s)”
Chapter 9

Security

This chapter describes the security features of PBS. These instructions are for the PBS administrator and Manager.

9.1 Configurable Features

This section gives an overview of the configurable security mechanisms provided by PBS, and gives links to information on how to configure each mechanism.
Chapter 9  Security

The following table lists configurable PBS security mechanisms and their configuration procedures.

**Table 9-1: Security Mechanisms and their Configuration Procedures**

<table>
<thead>
<tr>
<th>Security Mechanism</th>
<th>Configuration Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access control for Server, queues, reservations</td>
<td>“Using Access Control” on page 659</td>
</tr>
<tr>
<td>Event logging for Server, Scheduler, MOMs</td>
<td>“Event Logging” on page 890</td>
</tr>
<tr>
<td>File copy mechanism</td>
<td>“Setting File Copy Mechanism” on page 688</td>
</tr>
<tr>
<td>Levels of privilege (user roles)</td>
<td>“Setting User Roles” on page 655</td>
</tr>
<tr>
<td>Restricting access to execution hosts via $restrict_user</td>
<td>“Restricting Execution Host Access” on page 683</td>
</tr>
</tbody>
</table>
9.2 Setting User Roles

9.2.1 Root Privilege

Root privilege is required to perform some operations in PBS involving writing to the Server’s private, protected data. Root privilege is required in order to do the following:

- Alter MOM and Scheduler configuration files
- Set scheduler priority formula
- Run the following commands:
  - `pbs_probe`
  - `pbs_mom`
  - `pbs_sched`
  - `pbs_server`
  - `pbsfs`
- Use the `tracejob` command to view accounting log information

There are some operations that root privilege alone does not allow. These operations require Manager privilege but not root privilege. Manager privilege, but not root privilege, is required in order to do the following:

- Set attributes
- Create or delete vnodes using the `qmgr` command, which is the supported method.

9.2.2 User Roles

PBS allows certain privileges based on what role a person has, and whether that person has root privilege. PBS recognizes only three roles, and all those using PBS must be assigned one of these roles. These roles are `Manager`, `Operator`, and `user`. Roles are assigned by PBS Managers only. No roles can be added, and roles cannot be modified; the function of roles is hardcoded in the Server.
In addition to these roles, PBS requires a PBS administrator to perform some downloading, installation, upgrading, configuration, and management functions. PBS does not recognize *PBS administrator* as a PBS role; this term is used in PBS documentation to mean the person who performs these tasks.

PBS roles and PBS administrators are described in the following sections:

### 9.2.2.1 User

#### 9.2.2.1.1 Definition of User

Users are those who submit jobs to PBS.

Users have the lowest level of privilege. Users are referred to in the PBS documentation as “users”. By default, users may operate only on their own jobs. They can do the following:

- Submit jobs
- Alter, delete, and hold their own jobs
- Status their own jobs, and those of others if permission has been given via the `query_other_jobs` Server attribute. The `query_other_jobs` Server attribute controls whether unprivileged users are allowed to select or query the status of jobs owned by other users. This attribute is a Boolean, with default value of `False`, and can be set by a Manager only. See “Server Attributes” on page 365 of the PBS Professional Reference Guide.
- List and print some but not all Server, queue, vnode, Scheduler, and reservation attributes

#### 9.2.2.1.2 Defining List of Users

PBS allows you to define a list of users allowed or denied access to the PBS Server, however this is done using the PBS access control list mechanism. Access control is described in section 9.3 “Using Access Control” on page 659.
9.2.2.2 Operator

9.2.2.2.1 Definition of Operator

A PBS Operator is a person who has an account that has been granted Operator privilege.

Operators have more privilege than users, and less privilege than Managers.

Operators can manage the non-security-related attributes of PBS such as setting and unsetting non-security attributes of vnodes, queues, and the Server. Operators can also set queue ACLs.

Operators can do the following:

• All operations that users can perform
• Set non-security-related Server, queue, and vnode attributes (Operators are not permitted to set Server ACLs)
• Alter some job attributes
• Set or alter most resources on the Server, queues, and vnodes
• Rerun, requeue, delete, and hold all jobs
• Run any command to act on a job

9.2.2.2 Defining List of Operators

To define the list of Operators at a PBS complex, set the Server’s operators attribute to a list of usernames, where each username should be an Operator. See “Server Attributes” on page 365 of the PBS Professional Reference Guide.

It is important to grant Operator privilege to appropriate persons only, since Operators can control how user jobs run.

9.2.2.3 Manager

9.2.2.3.1 Definition of Manager

A Manager is a person who has an account that has been granted PBS Manager privilege.
Managers have more privilege than Operators. Managers can manage the security aspects of PBS such as Server ACLs and assignment of User Roles.

Managers can do the following:

• All operations that Operators can perform
• Create or delete queues or vnodes
• Set all Server, queue, and vnode attributes, including Server ACLs

9.2.2.3.2 Defining List of Managers

To define the list of Managers at a PBS complex, set the Server’s managers attribute to a list of usernames, where each username should be a Manager. See “Server Attributes” on page 365 of the PBS Professional Reference Guide.

If the Server’s managers attribute is not set or is unset, root on the Server’s host is given Manager privilege. If the managers attribute is set to a list of usernames, root on the Server’s host is no longer automatically given Manager privilege.

It is important to grant Manager privilege to appropriate persons only, since Managers control much of PBS.

9.2.2.4 PBS Administrator

9.2.2.4.1 Definition of PBS Administrator

A PBS administrator is a person who has an account with root privilege, and a separate account with Manager privilege.

PBS administrators perform all the functions requiring root privilege, as described in section 9.2.1 “Root Privilege” on page 655.

9.2.2.4.2 Defining PBS Administrators

A PBS administrator is a person with both root privilege and Manager privilege.
9.3 Using Access Control

9.3.1 Access Definitions

In this section we describe the meaning of access for each entity and object where the access of the entity to the object has an access control mechanism.

9.3.1.1 Access to a PBS Object

Below are the definitions of what access to each of the following PBS objects means:

Access to the Server
Being able to run PBS commands to submit jobs and perform operations on them such as altering, selecting, and querying status. It also means being able to get the status of the Server and queues.

Access to a queue
Being able to submit jobs to the queue, move jobs into the queue, being able to perform operations on jobs in the queue, and being able to get the status of the queue.

Access to a reservation
Being able to place jobs in the reservation, whether by submitting jobs to the reservation or moving jobs into the reservation. It also means being able to delete the reservation, and being able to operate on the jobs in the reservation.

9.3.1.2 Access by a PBS Entity

Access can be granted at the Server, queues, and reservations for each of the following entities:

User access
The specified user is allowed access.

Group access
A user in the specified group is allowed access.
Host access

A user is allowed access from the specified host

9.3.2 Requirement for Access

In order to have access to a PBS object such as the Server or a queue, a user must pass all enabled access control tests: the user must be allowed access, the user’s group must be allowed access, and the host where the user is working must be allowed access.

In some cases, Manager or Operator privilege overrides access controls. For some kinds of access, there are no controls. See section 9.3.11 “Operations Controlled by ACLs” on page 677.

9.3.3 Managing Access

PBS uses access control lists (ACLs) to manage access to the Server, queues, and reservations. There is a separate set of ACLs for the Server, each queue, and each reservation. The Server enforces the access control policy for User Roles supported by PBS. The policy is hardcoded within the Server. ACLs can specify which entities are allowed access and which entities are denied access.

Each Server and queue ACL can be individually enabled or disabled by a Manager. If an ACL is enabled, access is allowed or denied based on the contents of the ACL. If the ACL is disabled, access is allowed to all. The contents of each Server or queue ACL can be set or altered by a Manager.

Reservation ACLs are enabled only by the reservation creator. The Server’s resv_enable attribute controls whether reservations can be created. When this attribute is set to False, reservations cannot be created.

No default ACLs are shipped.

9.3.4 ACLs

An ACL, or Access Control List, is a list of zero or more entities (users, groups, or hosts from which users or groups may be attempting to gain access) allowed or denied access to parts of PBS such as the Server,
queues, or reservations. A Server ACL applies to access to the Server, and therefore all of PBS. A queue’s ACL applies only to that particular queue. A reservation’s ACL applies only to that particular reservation. The Server, each queue, and each reservation has its own set of ACLs.

9.3.4.1 Format of ACLs

Entity access is controlled according to the list of entities allowed or denied access as specified in the object’s acl_<entity> attribute. The object’s access control attribute contains a list of entity names, where each entity name is marked with a plus sign (“+”) if the entity is allowed access, and with a minus sign (“-“) if the entity is denied access. For example, to allow User1@host1.example.com, and deny User2@host1.example.com:

+User1@host1.example.com, -User2@host1.example.com

9.3.4.2 Default ACL Behavior

If an entity name is included without either a plus or a minus sign, it is treated as if it has a plus sign, and allowed access.

If an entity name is not in the list, the default behavior is to deny access to the entity. Therefore, if the list is empty but enabled because the object’s acl_<entity>_enable attribute is set to True (see section 9.3.5 “Enabling Access Control” on page 665), all entities are denied access.

9.3.4.3 Modifying ACL Behavior

You can specify how an ACL treats an unmatched entity by including special flags in the ACL itself. These are the plus and minus signs.

To allow access for all unmatched entities (the reverse of the default behavior), put a plus sign (“+”) anywhere by itself in the list. For example:

+User1@host1.example.com, +, -User2@host1.example.com

To deny access for all unmatched entities (the default behavior), put a minus sign (“-“) anywhere by itself in the list. For example:

+User1@host1.example.com, -, -User2@host1.example.com
If there are entries for both a plus and a minus sign, the last entry in the list (closest to the rightmost side of the list) will control the behavior of the ACL.

### 9.3.4.4 Contents of User ACLs

User ACLs contain a username and hostname combination. The subject's username and hostname combination is compared to the entries in the user ACL. Usernames take this form:

```
User1@host.domain.com
User1@host.subdomain.domain.com
```

Usernames can be wildcarded. See section 9.3.4.7 “Wildcards In ACLs” on page 662.

### 9.3.4.5 Contents of Group ACLs

Group ACLs contain names based on the user’s default group, as defined by the operating system where the Server is executing. The subject's default group name on the Server is compared to the entries in the Group ACL. Group names cannot be wildcarded.

### 9.3.4.6 Contents of Host ACLs

Host ACLs contain fully-qualified hostnames. The subject's host name is compared to the entries in the host ACL. To find the fully-qualified name of a host, use the `pbs_hostn` command. See “pbs_hostn” on page 57 of the PBS Professional Reference Guide.

Hostnames can be wildcarded. See the following section.

### 9.3.4.7 Wildcards In ACLs

Usernames and hostnames can be wildcarded. The hostname portion of the username is wildcarded exactly the same way a hostname is wildcarded. The non-hostname portion of a username cannot be wildcarded.
The only character that can be used to wildcard entity names is the asterisk ("*"). Wildcarding must follow these rules:

- The asterisk must be to the right of the at sign ("@")
- There can be at most one asterisk per entity name
- The asterisk must be the leftmost label after the at sign

The following table shows how hostnames are wildcarded:

<table>
<thead>
<tr>
<th>Wildcard Use</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.test.example.com</td>
<td>Any host in the test subdomain in example.com</td>
</tr>
<tr>
<td>*.example.com</td>
<td>Any host in example.com</td>
</tr>
<tr>
<td>*.com</td>
<td>Any host in .com</td>
</tr>
<tr>
<td>*</td>
<td>Any host</td>
</tr>
</tbody>
</table>

The following examples show how wildcarding works in host ACLs:

Example 1: To limit host access to host myhost.test.example.com only:

```
myhost.test.example.com
```

Example 2: To limit host access to any host in the test.example.com subdomain:

```
*.test.example.com
```

Example 3: To limit host access to any host in example.com:

```
*.example.com
```

Example 4: To allow host access for all hosts:

```
*
```

The following examples show how wildcarding works in user ACLs:

Example 1: To limit user access to UserA requesting from host myhost.test.example.com only:

```
UserA@myhost.test.example.com
```
Example 2: To limit user access to UserA on any host in the test.example.com subdomain:

   UserA@*.test.example.com

Example 3: To limit user access to UserA on any host in example.com:

   UserA@*.example.com

Example 4: To limit user access to UserA from anywhere:

   UserA@

   or

   UserA

Listing a username without specifying the host or domain is the equivalent of listing the username followed by “@*”. This means that

   User1

is the same as

   User1@

9.3.4.8  Restrictions on ACL Contents

All access control lists are traversed from left to right, and the first match found is used. It is important to make sure that entries appear in the correct order.

To single out a few, specify those few first, to the left of the other entries.

Example 1: To allow all users in your domain except User1 access, the list should look like this:

   -User1@example.com, ++@example.com

Example 2: To deny access to all users in your domain except User1, the list should look like this:

   +User1@example.com, -*@example.com
9.3.5 Enabling Access Control

Each Server and queue ACL is controlled by a Boolean switch whose default value is *False*, meaning that access control is turned off. When access control is turned off, all entities have access to the Server and to each queue. When access control is turned on, access is allowed only to those entities specifically granted access.

To use access control, first set the contents of the ACL, then enable it by setting its switch to *True*.

Reservation ACLs are enabled when the reservation creator sets their contents. Reservation ACLs do not have switches. See section 9.3.9.1 “Creating and Enabling Reservation Queue ACLs” on page 675.

Reservations use queues, which are regular queues whose ACL values have been copied from the reservation. See section 9.3.9 “How Reservation Access Control Works” on page 675.

9.3.5.1 Table of ACLs and Switches

The following table lists the ACLs and their switches, with defaults, for the Server, queues, and reservations.

<table>
<thead>
<tr>
<th>Table 9-3: ACLs and Their Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>User (Default Value)</td>
</tr>
<tr>
<td>Server</td>
</tr>
<tr>
<td>List</td>
</tr>
</tbody>
</table>
9.3.6 Creating and Modifying ACLs

Server and queue ACLs follow the same rules for creation and modification. Reservation queue ACLs behave the same way regular queue ACLs do. Reservation ACLs can only be created by the reservation creator, and cannot be modified. See section 9.3.8 “Reservation Access” on page 673.

9.3.6.1 Rules for Creating and Modifying Server and Queue ACLs

- Server and queue ACLs are created and modified using the qmgr com-
mand.

- An ACL is a list of entries. When you operate on the list, the first match found, searching from left to right, is used. If there is more than one match for the entity you wish to control, ensure that the first match gives the behavior you want.

- When you create or add to an ACL, you can use the + or - operators to specify whether or not an entity is allowed access. Omitting the operator is equivalent to adding a + operator.

- When you re-create an existing ACL, this is equivalent to unsetting the old ACL and creating a new one.

- When you add to an ACL, the new entry is appended to the end of the ACL, on the right-hand side.

- When you remove an entity from an ACL, you cannot use + or - operators to specify which entity to remove, even if there are multiple entries for an entity and each entry has a different operator preceding it, for example "-bob, +bob".

- When you remove an entity, only the first match found is removed.

### 9.3.6.2 Examples of Creating and Modifying Server and Queue ACLs

The following examples show the Server’s user ACL being set. Queue ACLs work the same way as Server ACLs, and the equivalent `qmgr` command can be used for queues. So, where we use the following for the Server:

```
Qmgr: set server acl_users ...
```

the same effect can be achieved at the queue using this:

```
Qmgr: set queue <queue name> acl_users ...
```

If the queue name is Q1, the `qmgr` command looks like this:

```
Qmgr: set queue Q1 acl_users ...
```
Example 1: To create a Server or queue ACL:

```
Qmgr: set <object> <ACL> = <entity list>
```

Example:

```
Qmgr: set server acl_users =
    "-User1@*.example.com, +User2@*.example.com"
```

ACL looks like this:

-User1@*.example.com, +User2@*.example.com

Example 2: To add to a Server or queue ACL:

```
Qmgr: set <object> <ACL> += <entity list>
```

Example:

```
Qmgr: set server acl_users += -User3@*.example.com
```

ACL looks like this:

-User1@*.example.com, +User2@*.example.com,
  -User3@example.com

Example 3: To remove an entry from an ACL:

```
Qmgr: set <object> <ACL> -= <entity>
```

Example:

```
Qmgr: set server acl_users -= User2@*.example.com
```

ACL looks like this:

-User1@*.example.com, -User3@*.example.com

Example 4: To remove two entries for the same entity from an ACL:

```
Qmgr: set <object> <ACL> -= <entity1, entity1>
```

Example: If ACL contains +A, +B, -C, -A, +D, +A

```
Qmgr: set server acl_users -= "A, A"
```

ACL looks like this:

+B, -C, +D, +A
Example 5: To remove multiple entities from an ACL:

```
Qmgr: set <object> <ACL> -= <entity list>
```

Example: If ACL contains +B, -C, +D, +A

```
Qmgr: set server acl_users -= "B, D"
```

ACL looks like this:

-C, +A

### 9.3.6.3 Who Can Create, Modify, Enable, or Disable ACLs

The following table summarizes who can create, modify, enable, or disable ACLs and their associated switches:

**Table 9-4: Who Can Create, Modify, Enable, Disable ACLs**

<table>
<thead>
<tr>
<th>ACLs and Switches</th>
<th>Manager</th>
<th>Operator</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server ACLs and Switches</td>
<td>Create</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Modify</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Queue ACLs and Switches</td>
<td>Create</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Modify</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Table 9-4: Who Can Create, Modify, Enable, Disable ACLs

<table>
<thead>
<tr>
<th>ACLs and Switches</th>
<th>Manager</th>
<th>Operator</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservation ACLs</td>
<td>Create</td>
<td>Only if reservation creator</td>
<td>Only if reservation creator</td>
</tr>
<tr>
<td></td>
<td>Modify</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Only if reservation creator</td>
<td>Only if reservation creator</td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Reservation Queue ACLs and</td>
<td>Create</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Switches</td>
<td>Modify</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Enable</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Disable</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 9.3.6.4 Who Can Operate on Server ACLs

PBS Managers only can create or modify Server ACLs and the Boolean switches that enable them.

### 9.3.6.5 Who Can Operate on Queue ACLs

PBS Managers and Operators, but not users, can create and modify queue ACLs and their Boolean switches.
9.3.6.6  Who Can Operate on Reservation ACLs

When creating a reservation, the reservation creator cannot disable the user ACL, but can choose to enable or disable the group and host ACLs implicitly via the command line, and can specify the contents of all three ACLs. Reservation ACLs cannot be modified or disabled.

9.3.6.7  Who Can Operate on Reservation Queue ACLs

Unprivileged users cannot directly create, modify, enable, or disable reservation queue ACLs or the associated switches. The reservation creator can indirectly create and enable the reservation queue’s ACLs during reservation creation. If a user wants to modify a reservation queue’s ACLs, they can do so indirectly by deleting the reservation and creating a new one with the desired ACLs.

PBS Managers and Operators can modify, enable, or disable a reservation queue’s ACLs.

A reservation queue’s user ACL is always enabled unless explicitly disabled after creation by a Manager or Operator.

9.3.7  Server and Queue ACLs

Access control for an entity such as a user, group, or host is enabled by setting the attribute enabling that entity’s ACL to True. When this attribute is True, entity access is controlled according to the list of entities allowed or denied access as specified in the ACL for that entity. The default value for each ACL’s switch attribute is False, meaning that entity access is not controlled.

9.3.7.1  Server ACLs

The Server has two ACLs: a host ACL and a user ACL.

Server access is controlled by these attributes:

- User access: acl_user_enable and acl_users
- Host access: acl_host_enable and acl_hosts
9.3.7.2 Queue ACLs

Each queue has three ACLs: a host ACL, a user ACL, and a group ACL. Queue access is controlled by these attributes:

- User access: acl_user_enable and acl_users
- Group access (queue only): acl_group_enable and acl_groups
- Host access: acl_host_enable and acl_hosts

9.3.7.3 Examples of Setting Server and Queue Access

To restrict access to the Server or queue, first set the contents of the ACL, then enable the ACL by setting its switch to True.

Example 1: To allow Server access for all users in your domain except User1, and to allow Server access for User2 in another domain:

Set the Server’s acl_users attribute:

```
Qmgr: set server acl_users = "-User1@example.com, +*@example.com, +User2@otherdomain.com"
```

Enable user access control by setting the Server’s acl_user_enable attribute to True:

```
Qmgr: set server acl_user_enable = True
```

Example 2: To further require that users of the Server be in Group1 only:

Set the Server’s acl_groups attribute:

```
Qmgr: set server acl_groups = +Group1
```

Enable group access control by setting the Server’s acl_group_enable attribute to True:

```
Qmgr: set server acl_group_enable = True
```
Example 3: To allow access to Queue1 from Host1 only:

Set the queue’s acl_hosts attribute:

```
Qmgr: set q Queue1 acl_hosts = +Host1@example.com
```

Enable host access control by setting the queue’s acl_host_enable attribute to True:

```
Qmgr: set q Queue1 acl_host_enable = True
```

### 9.3.8 Reservation Access

Reservations are designed to be created by unprivileged users, although Managers and Operators can create them as well. The Server’s resv_enable attribute controls whether reservations can be created. When this attribute is set to False, reservations cannot be created.

Each reservation has its own access control attributes that can be used to specify which users and groups have access to the reservation, and the hosts from which these users and groups are allowed access. The creator of the reservation sets the list of users, groups and hosts that have access to the reservation. This is done at the time of reservation creation, using options to the pbs_rsub command.

The creator of a reservation creates that reservation’s ACLs. Reservation ACLs are enabled indirectly through the actions of the reservation creator. The reservation’s list of authorized users is always enabled during reservation creation. The reservation’s lists of authorized groups and authorized hosts are only enabled if explicitly set by the reservation creator.

#### 9.3.8.1 Setting Reservation Access

Reservation access is controlled by the following reservation attributes:

- User access: Authorized_Users
- Group access: Authorized_Groups
- Host access: Authorized_Hosts

The pbs_rsub command has three options that allow the user to set the attributes above.
-U <authorized user list>
    If the user does not specify this option, Authorized_Users is set to the username of the person running the command, and the list is enabled.

    If the user specifies a list of authorized users, Authorized_Users is set to this list, and the list is enabled. The creator’s username at the Server is automatically added to this list.

-G <authorized group list>
    If the user does not specify this option, Authorized_Groups is left blank, and the list is not enabled.

    If the user specifies a list of authorized groups, Authorized_Groups is set to this list, and the list is enabled. The creator’s group at the Server is not automatically added to this list.

-H <authorized host list>
    If the user does not specify this option, Authorized_Hosts is left blank, and the list is not enabled.

    If the user specifies a list of authorized hosts, Authorized_Hosts is set to this list, and the list is enabled. The creator’s host is not automatically added to this list.

Reservation ACLs allow or deny access based on group names, usernames, and host names.

9.3.8.2 Reservation Access Defaults

The defaults for each reservation access control attribute are as follows:

- Authorized_Users: The reservation creator only is allowed access
- Authorized_Groups: All groups are allowed access
- Authorized_Hosts: All hosts are allowed access
9.3.8.3 Requirements for Reservation Access

If a user is denied access to a reservation via the reservation’s ACLs, that means that the user cannot do the following:

- Submit a job into the reservation
- Move a job into the reservation

However, that user can still operate on his/her own jobs in the reservation, as long as that user is not denied access at the Server level. The user can do the following:

- Delete the job
- Hold the job
- Move the job out of the reservation

For example, if an Operator qmoves User1’s job into a reservation to which User1 is denied access, User1 can still perform operations on the job such as deleting or holding the job, and User1 can qmove the job out of the reservation.

9.3.9 How Reservation Access Control Works

Each reservation has its own queue, which has its own ACLs, each controlled by a Boolean switch, as described in section 9.3.7.1 “Server ACLs” on page 671. Access to a reservation is actually controlled through the reservation’s queue ACLs. Each reservation access control attribute is copied to the equivalent attribute for the reservation’s queue.

9.3.9.1 Creating and Enabling Reservation Queue ACLs

If the group or host reservation ACL is specified by the reservation creator, the associated Boolean switch for the reservation queue ACL is set to True.

Authorized_Users is always set to the creator and copied to the queue’s acl_users attribute, and acl_user_enable is always set to True.
If **Authorized_Groups** is specified by the creator, it is copied to the queue’s `acl_groups` attribute and `acl_group_enable` is set to `True`. If the reservation creator does not specify a value for **Authorized_Groups**, nothing is copied to the queue’s `acl_groups`, and `acl_group_enable` remains at its default value of `False`.

If **Authorized_Hosts** is specified by the creator, it is copied to the queue’s `acl_hosts` attribute and `acl_host_enable` is set to `True`. If the reservation creator does not specify a value for **Authorized_Hosts**, nothing is copied to the queue’s `acl_hosts`, and `acl_host_enable` remains at its default value of `False`.

The following table shows the relationships between reservation ACLs and reservation queue ACLs:

**Table 9-5: Relationship Between Reservation ACLs and Reservation Queue ACLs**

<table>
<thead>
<tr>
<th>Entity</th>
<th>Reservation ACL</th>
<th>Reservation Queue ACL</th>
<th>Reservation Queue ACL Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>Authorized_Users</td>
<td>acl_users</td>
<td>acl_user_enable</td>
</tr>
<tr>
<td>Groups</td>
<td>Authorized_Groups</td>
<td>acl_groups</td>
<td>acl_group_enable</td>
</tr>
<tr>
<td>Hosts</td>
<td>Authorized_Hosts</td>
<td>acl_hosts</td>
<td>acl_host_enable</td>
</tr>
</tbody>
</table>

**9.3.9.2 Examples of Setting Reservation Access**

Example 1: To disallow access for User1 and allow access for all other users at your domain:

- Set reservation’s **Authorized_Users** attribute using the `-U` option to `pbs_rsub`:

  ```
pbs_rsub ... -U "-User1@example.com, ++@example.com"
  ```

Example 2: To allow access for Group1 and Group2 only:

- Set reservation’s **Authorized_Groups** attribute using the `-G` option to `pbs_rsub`:

  ```
pbs_rsub ... -G "+Group1, +Group2"
  ```

Note that any users in Group1 and Group2 to whom you wish to grant access must be explicitly granted access in the **Authorized_Users** list.
Example 3: To allow access from Host1 and Host2 only:

- Set reservation’s `Authorized_Hosts` attribute using the `-H` option to `pbs_rsub`:

  ```
pbs_rsub ... -H "+Host1.example.com, +Host2.example.com, -*.example.com"
  ```

### 9.3.10 Scope of Access Control

Queue-level ACLs provide different security functionality from that provided by Server-level ACLs. Access to PBS commands is controlled by Server-level ACLs. For example, access to the `qstat` and `qselect` operations are controlled only at the Server level. For unprivileged users, access to a specific queue is controlled through that queue’s ACLs.

The users allowed access to a queue or reservation are a subset of the users allowed access to the Server. Therefore, if you wish to allow a user access to a queue, that user must also be allowed access to the Server. The hosts from which a user may run commands at a queue are a subset of the hosts from which a user may run commands at the Server. See “Server Attributes” on page 365 of the PBS Professional Reference Guide, “Queue Attributes” on page 407 of the PBS Professional Reference Guide, and “Reservation Attributes” on page 393 of the PBS Professional Reference Guide.

### 9.3.11 Operations Controlled by ACLs

ACLs control some operations in PBS, but not others. Manager and Operator privileges override some ACL restrictions.

#### 9.3.11.1 Server Operations Controlled by ACLs

##### 9.3.11.1.1 Server Host ACL

If it is enabled, the Server’s host ACL is checked for and controls all Server operations, and is honored regardless of privilege. Any request coming from a disallowed host is denied.
9.3.11.2 Server User ACL

If it is enabled, the Server’s user ACL is checked for and controls all Server operations, but is overridden by Manager or Operator privilege. This means that the Server’s user ACL applies only to users, not to Managers or Operators. Even if explicitly denied access in the Server’s user ACL, a PBS Manager or Operator is allowed access to the Server. Note that queue access is controlled separately by queue ACLs; even if Managers or Operators are explicitly denied access in the Server’s user ACL, if a queue’s ACLs are not enabled, Managers and Operators have access to the queue. The same is true for reservations.

9.3.11.2 Queue Operations Controlled by ACLs

If enabled, queue ACLs are applied only when an entity is attempting to enqueue a job. Enqueueuing a job can happen in any of three ways:

- Moving a job into the queue
- Submitting a job to the queue
- Routing a job into the queue

Queue ACLs are not applied for non-enqueueing operations, for example:

- Moving a job out of the queue
- Holding a job
- Deleting a job
- Signaling a job
- Getting job status

9.3.11.2.1 Queue Host ACL

If a queue’s host ACL is enabled, it is checked when an entity attempts to enqueue a job. The host ACL is always honored, regardless of privilege.

9.3.11.2.2 Queue User and Group ACLs

If a queue’s user or group ACL is enabled, it is applied when an entity attempts to enqueue a job. Manager and Operator privileges override queue user and group ACLs when an entity attempts to move a job into a queue. This means that a PBS Manager or Operator who is explicitly
Security

9.3.11.3 Reservation Operations Controlled by ACLs

Access to a reservation’s queue is controlled through its queue’s ACLs. A reservation’s queue behaves exactly the same way as a regular queue.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Applied to Users</th>
<th>Applied to Managers/Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moving a job into the queue</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Submitting a job to the queue</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Having a job routed into the queue</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
9.3.11.4 Table of Operations Controlled by ACLs and Overrides

The following table lists which operations are and are not controlled by Server and queue ACLs, and which controls are overridden.

**Table 9-7: Operations Controlled by ACLs, and ACL Overrides**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Server ACLs</th>
<th>Queue ACLs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Host</td>
<td>User</td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moved job into queue</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Moved job out of queue</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Submitted job to queue</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Have job routed into queue</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Delete job</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Hold job</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Release job</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Signal job</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Status job</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Status Server</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Status queue</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>
9.3.12 Avoiding Problems

9.3.12.1 Using Group Lists

When a user specifies a group list, each and every group in which that user might execute a job must have a group name and an entry in the groups database, for example, /etc/group.

9.3.13 Flatuid and Access

The Server’s flatuid attribute affects both when users can operate on jobs and whether users without accounts on the Server host can submit jobs.

9.3.13.1 How flatuid Controls When Users Can Operate On Jobs

This section describes how the Server’s flatuid attribute affects the circumstances under which users can operate on jobs.

This attribute specifies whether, for each user, the username at the submission host must be the same as the one at the Server host. The username at the Server host must always be the same as the username at the execution host. When flatuid is set to True, the Server assumes that UserA@host1 is the same as UserA@host2. Therefore, if flatuid is True, UserA@host2 can operate on UserA@host1’s job.

The value of flatuid also affects whether .rhosts and host.equiv are checked. If flatuid is True, .rhosts and host.equiv are not queried, and for any users at host2, only UserA is treated as UserA@host1. If flatuid is False, .rhosts and host.equiv are queried.

That is, when flatuid is True, even if UserB@host2 is in UserA@host1's .rhosts, UserB@host2 cannot operate on UserA's job(s). If flatuid is False, and UserB@host2 is in UserA@host1’s .rhosts, UserB@host2 is allowed to operate on UserA's job(s).
Example:

UserA@host1 has a job
UserB@host2 is in UserA@host1’s .rhosts

a. flatuid = True: UserB@host2 cannot operate on UserA’s job
b. flatuid = False: UserB@host2 can operate on UserA’s job

The following table shows how access is affected by both the value of the server’s flatuid attribute and whether UserB@host2 is in UserA@host1’s .rhosts:

<table>
<thead>
<tr>
<th>UserB@host2 in UserA@host1’s .rhosts</th>
<th>flatuid = True</th>
<th>flatuid = False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is UserA@host1 treated as UserA@host2?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Is .rhosts queried?</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Can UserB operate on UserA’s jobs?</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### 9.3.13.2 How flatuid Affects Users Without Server Accounts

This section describes how the Server’s flatuid attribute affects users who have no account on the Server host.

- If flatuid is set to False, users who have no account at the Server host cannot submit jobs to PBS.
- If flatuid is set to True, these users can submit jobs. However, the job will only run if it is sent to execution hosts where the user does have an account. If the job is sent to execution hosts where the user does not have an account, the job will not run, and the MOM will log an error message.
9.4 Restricting Execution Host Access

You can configure each PBS execution host so that the only users who have access to the machine are those who are running jobs on the machine. You can specify this by adding the `$restrict_user` parameter to the MOM configuration file `PBS_HOME/mom_priv/config`. This parameter is a Boolean, which if set to `True`, prevents any user not running a job from running any process on the machine for more than 10 seconds. The interval between when PBS applies restrictions depends upon MOM’s other activities, but can be no more than 10 seconds.

You can specify which users are exempt from this restriction by adding the `$restrict_user_exceptions` parameter to the same file. See the description of the parameter in the next section.

You can allow system processes to run by specifying the maximum numeric user ID allowed access to the machine when not running a job. You do this by adding the `$restrict_user_maxsysid` parameter to the MOM configuration file. PBS automatically tries to allow system processes to run: if `$restrict_user` is enabled and `$restrict_user_maxsysid` is unset, PBS looks in `/etc/login.defs` for `SYSTEM_UID_MAX` for the value to use. If there is no maximum ID set there, it looks for `SYSTEM_MIN_UID`, and uses that value minus 1. Otherwise PBS uses the default value of 999. See section 3.6.6 “Restricting User Access to Execution Hosts” on page 68 and “$restrict_user <value>” on page 321 of the PBS Professional Reference Guide.

9.4.1 MOM Access Configuration Parameters

These are the configuration parameters in `PBS_HOME/mom_priv/config` that can be set to restrict and specify access to each execution host. Each execution host has its own configuration file.

```
$restrict_user <value>
```

Controls whether users not submitting jobs have access to this machine. When `True`, only those users running jobs are allowed access.

Format: Boolean
Default: off
$\text{restrict\_user\_exceptions} \ <\text{user\_list}>

List of users who are exempt from access restrictions applied by $\text{restrict\_user}$. Maximum number of names in list is 10.

Format: Comma-separated list of usernames; space allowed after comma

$\text{restrict\_user\_maxsysid} \ <\text{value}>

Allows system processes to run when $\text{restrict\_user}$ is enabled. Any user with a numeric user ID less than or equal to value is exempt from restrictions applied by $\text{restrict\_user}$.

Format: Integer

Default: 999

### 9.4.2 Examples of Restricting Access

To restrict user access to those running jobs, add:

$\text{restrict\_user} \ \text{True}$

To specify the users who are allowed access whether or not they are running jobs, add:

$\text{restrict\_user\_exceptions} \ <\text{user\_list}>

For example:

$\text{restrict\_user\_exceptions} \ \text{User1, User2}$

To allow system processes to run, specify the maximum numeric user ID by adding:

$\text{restrict\_user\_maxsysid} \ <\text{user\_ID}>

For example:

$\text{restrict\_user\_maxsysid} \ 999$
9.5 Logging Security Events

Each PBS daemon logs security-related events, at event class 32 (0x0020) or at event class 128 (0x0080). For information about daemon logfiles, see section 13.4 “Event Logging” on page 890.

9.5.1 Events Logged at Event Class 32 (0x0020)

The following security-related events are logged at decimal event class 32 (0x0020):

- When an execution host has access restrictions in place via the $restrict_user configuration parameter, and MOM detects that a user who is not exempt from access restriction is running a process on the execution host, MOM kills that user's processes and writes a log message:

  01/16/2006 22:50:16;0002;pbs_mom;Svr;restrict_user; killed uid 1001 pid 13397(bash) with log event class PBSE_SYSTEM.

  See section 9.4 “ Restricting Execution Host Access ” on page 683.

- If for some reason the access permissions on the PBS file tree are changed from their default settings, a daemon may detect this as a security violation, refuse to execute, and write an error message in the corresponding log file. The following are examples of each daemon’s log entry:

  Server@<host>: Permission denied (13) in chk_file_sec,
  Security violation "/usr/spool/PBS/server_priv/jobs/"
  resolves to "/usr/spool/PBS"

  pbs_mom: Permission denied (13) in chk_file_sec, Security violation "/usr/spool/PBS/mom_priv/jobs/"
  resolves to "/usr/spool/PBS"

  pbs_sched: Permission denied (13) in chk_file_sec,
  Security violation "/usr/spool/PBS/sched_priv"
  resolves to "/usr/spool/PBS"
A Manager can run `pbs_probe` (on UNIX/Linux) or `pbs_mkdirs` (on Windows) to check and optionally correct any directory permission or ownership problems.

- When a user without a password entry (an account) on the Server attempts to submit a job, the Server logs this event. The following is an example log entry:
  8/21/2009
  15:28:30;0080;Server@capella;Req;req_reject;Reject
  reply code=15023, aux=0, type=1, from
  User1@host1.example.com

- If a daemon detects that a file or directory in the PBS hierarchy is a symbolic link pointing to a non-secure location, this is written to the daemon’s log. The resulting log message is the same as for a permission violation:
  
  ```
  Server@<host>: Permission denied (13) in chk_file_sec,
  Security violation "/usr/spool/PBS/server_priv/jobs/
  " resolves to "/usr/spool/PBS"
  ```

  ```
  pbs_mom: Permission denied (13) in chk_file_sec, Security violation "/usr/spool/PBS/mom_priv/jobs/"
  resolves to "/usr/spool/PBS"
  ```

  ```
  pbs_sched: Permission denied (13) in chk_file_sec,
  Security violation "/usr/spool/PBS/sched_priv"
  resolves to "/usr/spool/PBS"
  ```

- If an `$action` script is to be executed for a job belonging to a user who does not have an account on an execution host, the execution host’s MOM logs this event. The following is an example log entry:
  08/21/2009 16:06:49;0028;pbs_mom;Job;2.host1;No Password Entry for User User1

- When a job triggers an action script for which the environment cannot be set up, perhaps due to a system error, the MOM attempting to run the action script logs the event. The log message contains the following:
  :jobid:failed to setup dependent environment!

- When the scheduler attempts to run a job on an execution host where the job’s owner does not have an account, the MOM on the execution host
logs this event. The following is an example log entry:

08/21/2009 15:51:14;0028;pbs_mom;Job;1.host1;No Pass-
word Entry for User User1

- When the scheduler attempts to run a job on an execution host where the
  job’s owner does not have a home directory, and when the job’s sandbox
  attribute is not set to PRIVATE, the execution host’s MOM logs
  this event. The log message contains the following:

  Access from host not allowed, or unknown host: <numeric
  IP address>

  See “pbs_mom” on page 66 of the PBS Professional Reference Guide.

- If an attempt is made to connect to a host in the PBS complex from an
  unknown host, the PBS daemon logs the information at both levels 32
  and 128 (0x0020 and 0080).

### 9.5.1.1 EventsLogged at Event Class 128 (0x0080)

The following security-related event is logged at event class 128 (0x0080):

- If an attempt is made to connect to a host in the PBS complex from an
  unknown host, the PBS daemon logs the information at both levels 32
  and 128 (0x0020 and 0080).

- If a user or Operator tries to set an attribute that can be set by Managers
  only, or attempts to create or delete vnodes:

  The qmgr command returns this error message:

  qmgr obj=<object> svr=default: Unauthorized Request
  qmgr: Error (15007) returned from server

  The Server logs the following message:

  Req;req_reject;Reject reply code=15007, aux=0, type=9,
  from <username>

- When a user is denied access to the Server because of the contents of the
  acl_users Server attribute, the Server logs the following:

  Req;req_reject;Reject reply code=15007, aux=0, type=21,
  from username@host.domain.com
9.5.1.2 Events Logged at Event Class 1

- When an attempt is made to contact MOM from a non-privileged port for a request requiring a privileged port, MOM logs the following:
  
  `pbs_mom;Svr;pbs_mom;Unknown error: 0 (0) in rm_request, bad attempt to connect message refused from port 61558 addr 127.0.0.1`

9.5.1.3 Events Not Logged

The following events are not logged:

- When an attempt is made to connect to a host in the PBS complex from a disallowed host
- When an ACL check denies an entity access to a PBS object
- A user tries to query other users’ jobs when the Server’s `query_other_jobs` attribute is set to `False`
- When an Operator or Manager overrides the Server’s user ACL

9.6 Setting File Copy Mechanism

When PBS copies files from one location to another, the default file copy mechanism it uses is the `pbs_rcp` command. You can choose to use the `scp` command instead. To do this, set the `PBS_SCP` parameter in `/etc/pbs.conf` to the absolute path of the `scp` command. When the `PBS_SCP` parameter is set, PBS tries `scp` first instead of `rcp`. If `scp` fails, it will try `rcp`, then `scp`, as described in section 9.7 “Security Considerations for Copying Files” on page 689. If the `PBS_SCP` parameter is not set, PBS will not use `scp`.

Set this parameter in `/etc/pbs.conf` on each execution host. If the MOM is already running, stop and restart MOM. Whether jobs are preserved depends on how you stop MOM. For information on how to stop MOM, see section 6.1.7 “Impact of Shutdown / Restart on Running UNIX/Linux Jobs” on page 217 in the PBS Professional Installation & Upgrade

For information on starting MOM, see section 6.1.6 “Stopping & Restarting One MOM” on page 216 in the PBS Professional Installation & Upgrade Guide.

### 9.7 Security Considerations for Copying Files

If using Secure Copy (scp), then PBS will first try to deliver output or stagein/out files using scp. If scp fails, PBS will try again using rcp (assuming that scp might not exist on the remote host). If rcp also fails, the above cycle will be repeated after a delay, in case the problem is caused by a temporary network problem. All failures are logged in MOM’s log, and an email containing the errors is sent to the job owner.

Attempts:

1. scp
2. rcp
3. scp
4. rcp
5. scp
6. rcp
7. scp
8. rcp
9.8 Changing the PBS Service Account Password

Normally, the password for the PBS service account on Windows should not be changed. But if it is necessary to change it, perhaps due to a security breach, then do so using the following steps:

1. Change the PBS service account's password on one machine in a command prompt from an admin-type of account by typing:

   Domain environments:
   
   ```
   net user <name of PBS service account> * /domain
   ```

   Non-domain environment:
   
   ```
   net user <name of PBS service account> *
   ```

2. Provide the Service Control Manager (SCM) with the new password given above. Do this either using the GUI-based Services application which is one of the Administrative Tools, or by unregistering and re-registering the PBS services with the password. See “pbs_account” on page 50 of the PBS Professional Reference Guide.

   To unregister on 32-bit Windows systems:
   
   ```
   pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_server.exe"
   pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_mom.exe"
   pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_sched.exe"
   pbs_account --unreg "\Program Files\PBS Pro\exec\sbin\pbs_rshd.exe"
   ```
To re-register on 32-bit Windows systems:

```
pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_server.exe"
pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_mom.exe"
pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_sched.exe"
pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_rshd.exe"
```

To unregister on 64-bit Windows systems:

```
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_server.exe"
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_mom.exe"
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_sched.exe"
pbs_account --unreg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_rshd.exe"
```

To re-register on 64-bit Windows systems:

```
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_server.exe"
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_mom.exe"
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_sched.exe"
pbs_account --reg "\Program Files (x86)\PBS Pro\exec\sbin\pbs_rshd.exe"
```

When re-registering, you can give an additional -p password argument to the pbs_account command, to specify the password on the command line.
9.9 Paths and Environment Variables

A significant effort has been made to ensure the various PBS components themselves cannot be a target of opportunity in an attack on the system. The two major parts of this effort are the security of files used by PBS and the security of the environment. Any file used by PBS, especially files that specify configuration or other programs to be run, must be secure. The files must be owned by root and in general cannot be writable by anyone other than root.

A corrupted environment is another source of attack on a system. To prevent this type of attack, each component resets its environment when it starts. If it does not already exist, the environment file is created during the install process. As built by the install process, it will contain a very basic path and, if found in root’s environment, the following variables:

- TZ
- LANG
- LC_ALL
- LC_COLLATE
- LC_CTYPE
- LC_MONETARY
- LC_NUMERIC
- LC_TIME

The environment file may be edited to include the other variables required on your system.

**IMPORTANT:**

Note that **PATH** must be included. This value of **PATH** will be passed on to batch jobs. To maintain security, it is important that **PATH** be restricted to known, safe directories. Do NOT include “.” in **PATH**. Another variable which can be dangerous and should not be set is **IFS**.

The entries in the **PBS_ENVIRONMENT** file can take two possible forms:

```
variable_name=value
variable_name
```
In the latter case, the value for the variable is obtained before the environment is reset.

## 9.10 Authentication & Authorization

### 9.10.1 Host Authentication

PBS uses a combination of information to authenticate a host. If a request is made from a client whose socket is bound to a privileged port (less than 1024, which requires root privilege), PBS believes the IP (Internet Protocol) network layer as to whom the host is. If the client request is from a non-privileged port, the name of the host which is making a client request must be included in the credential sent with the request and it must match the IP network layer opinion as to the host’s identity.

### 9.10.2 Host Authorization

Access to the Server from another system may be controlled by an access control list (ACL). Access to `pbs_mom` is controlled through a list of hosts specified in the `pbs_mom`’s configuration file. By default, only “local-host”, the name returned by `gethostname(2)`, and the host named by `PBS_SERVER` from `/etc/pbs.conf` are allowed. See “MOM Parameters” on page 311 of the PBS Professional Reference Guide for more information on the configuration file. Access to `pbs_sched` is not limited other than it must be from a privileged port.

### 9.10.3 User Authentication

The PBS Server authenticates the user name included in a request using the supplied PBS credential. This credential is supplied by `pbs_iff`.
Hostbased authentication will allow users within your complex to execute commands on or transfer files to remote machines. This can be accomplished for both the r-commands (e.g., rsh, rcp), and secure-commands (e.g., ssh, scp). The following procedure does not enable root to execute any r-commands or secure-commands without a password. Further configuration of the root account would be required.

Correct name resolution is important. Using fully qualified domain names on one machine and short names on another will not work. Name resolution must be consistent across all machines.

9.11.1 RSH/RCP

- Verify that the rsh-server and rsh-client packages are installed on each host within the complex.
- Verify that the rsh and rlogin services are on on each host within the complex. Example:

```
chkconfig --list | grep -e rsh -e rlogin
rlogin: on
rsh: on
```

On the headnode (for simplicity) add the hostname of each host within the complex to `/etc/hosts.equiv`, and distribute it to each host within the complex. Example file (filename: `/etc/hosts.equiv`):

```
headnode
dnode01
dnode02
dnode03
dnode04
dnode05
```
9.11.2 SSH/SCP

- Verify that the openSSH package is installed on each host within the complex.
- Verify that the openSSH service is on on each host within the complex. Example:
  
  ```
  chkconfig --list | grep ssh
  sshd 0:off 1:off 2:on 3:on 4:on 5:on 6:off
  ```

- Modify the following ssh config files on each host within the complex to enable the hostbased authentication. These options may be commented out, and so must be uncommented and set.

  a. `/etc/ssh/sshd_config`

  ```
  HostbasedAuthentication yes
  ```

  b. `/etc/ssh/ssh_config`

  ```
  HostbasedAuthentication yes
  ```

- Stop and start the openSSH service on each host within the complex.
  
  ```
  /etc/init.d/sshd stop
  /etc/init.d/sshd start
  ```

- On the headnode (for simplicity) create a file which contains the hostname and IP address of each host within the complex, where the hostname and IP address are comma delimited. Each entry should have all of the information from the line in `/etc/hosts`. Example file (file-
name: ssh_hosts):
  headnode,headnode.company.com,192.168.1.100
  node01,node01.company.com,192.168.1.1
  node02,node02.company.com,192.168.1.2
  node03,node03.company.com,192.168.1.3
  node04,node04.company.com,192.168.1.4
  node05,node05.company.com,192.168.1.5

So that if your /etc/hosts file has:
192.168.1.7 host05.company.com host05

the line in ssh_hosts would be:
node05,node05.company.com,192.168.1.7

• Gather each host’s public ssh host key within the complex by executing
  ssh-keyscan against the ssh_hosts file created in Step 5, and
  distribute the output to each host within the complex.
  
  `ssh-keyscan -t rsa -f ssh_hosts > /etc/ssh/ssh_known_hosts2`

• Create the /etc/ssh/hosts.equiv file for all of the machines in
  the complex. This must list the first name given in each line in the /etc/hosts file. Using the example from step 5:

  Your /etc/hosts file has:
  192.168.1.7 host05.company.com host05

  The hosts.equiv file should have:
  node05.company.com

• Every machine in the complex will need to have ssh_config and
  sshd_config updated. These files can be copied out to each
  machine.

### 9.11.3 Special Notes

The configurations of OpenSSH change (frequently). Therefore, it is important to understand what you need to set up. Here are some tips on some versions.
9.11.3.1 OpenSSH_3.5p1

Procedure above should work.

9.11.3.2 OpenSSH_3.6.1p2

Procedure above should work with the following additional step:

Define “EnableSSHKeysign yes” in the /etc/ssh/ssh_config file

9.11.3.3 OpenSSH_3.9p1

Procedure above should work with the following two additional steps:

1. Define “EnableSSHKeysign yes” in the /etc/ssh/ssh_config file
2. Change permissions from 0755 to 4755:

   chmod 4755 /usr/lib/ssh/ssh-keysign

This file is required to be setuid to work.

9.11.3.4 LAM

Use “ssh -x” instead of “ssh”.

If you want to use SSH you should enable ‘PermitUserEnvironment yes' so that the user's environment will be passed to the other hosts within the complex. Otherwise, you will see an issue with tkill not being in the user's PATH when executing across the hosts.

9.11.3.5 SuSe

Be sure that hostname resolution works correctly. Double-check that the server machine resolves the client’s IP address to the same name that is in your known hosts and hosts.equiv. You can have comma-delimited names, including IP addresses, ahead of the key, as in the following:

   example,example.domain,192.168.100.1 ssh-rsa thekey
Performing host based authentication as root may require that you set up a .shosts file in the home directory of root.

See http://www.ssh.com/support/documentation/online/ssh/adminguide/32/Host-Based_Authentication.html for more information.

9.12 Root-owned Jobs

The Server will reject any job which would execute under the UID of zero unless the owner of the job, typically root/Administrator, is listed in the Server attribute acl_roots.

The Windows version of PBS considers as a “root” account the following:

• Local SYSTEM account
• Account that is a member of the local Administrators group on the local host
• Account that is a member of the Domain Admins group on the domain
• Account that is a member of the Administrators group on the domain controller
• Account that is a member of the Enterprise Admins group on the domain
• Account that is a member of the Schema Admins group on the domain

In order to submit a job from this “root” account on the local host, be sure to set acl_roots. For instance, if user foo is a member of the Administrators group, then you need to set:

Omgr: set server acl_roots += foo

in order to submit jobs and not get a “bad uid for job execution” message.

IMPORTANT:

Allowing “root” jobs means that they can run on a configured host under the same account which could also be a privileged account on that host.
9.13   User Passwords

PBS has different password requirements dictated by the UNIX, Linux and
Windows operating systems, and by HPC Basic Profile. Jobs submitted on
UNIX and Linux systems do not require passwords. Jobs on HPC Basic
Profile and Windows systems require passwords. Jobs that will run at an
HPC Basic Profile server must be submitted from an HPCBP submission
host, which must have the supported architecture. See the PBS Profes-
sional 11.1 release notes for a list of supported architectures.

PBS provides two systems for handling user passwords, but these two sys-
tems are incompatible. These two systems are per-user/per-server pass-
words and per-job passwords.

9.13.1   Windows and HPC Basic Profile

Windows systems and HPC Basic Profile servers require a password for
PBS to run a process as the user, so users on these systems must supply a
password. User passwords can be managed using one of the following:

• Per-user/per-server passwords, where each user submits a password
  once for each server. Note that this method cannot be used when the
  PBS complex contains machines that are neither Windows nor HPCBP
  submission hosts.
• Per-job passwords, where each user submits a password once for each
  job

You can choose one or the other, but not both.

9.13.1.1   Per-user/per-server Passwords

You can configure the PBS server to store user passwords, so that each user
only has to submit a password once, and that password is used for all of the
user’s jobs. To enable the server to store user passwords, set the server’s
single_signon_password_enable attribute to True. The
pbs_password command is then run for each user, either by the user or
by you, before the user submits any jobs.
9.13.1.1.1 The single_signon_password_enable Server Attribute

single_signon_password_enable
If enabled, this option allows users to specify their passwords only once, and PBS will remember them for future job executions. Incompatible with qsub -Wpwd.
Format: Boolean
Default: False (UNIX), True (Windows)
Python attribute value type: bool

9.13.1.1.2 Examples

To enable the server to store passwords:

Qmgr: set server single_signon_password_enable = True

To specify a user’s password:

pbs_password

9.13.1.1.3 Caveats for single_signon_password_enable

• If your PBS complex contains UNIX or Linux machines whose platform is not the supported architecture for HPCBP submission hosts, jobs cannot be submitted from those machines and cannot be run on those machines when the server’s single_signon_password_enable attribute is set to True. This is because the pbs_password command is not available on those machines.

• When the server’s single_signon_password_enable attribute is set to True, users cannot use the -Wpwd option to the qsub command.

• The server’s single_signon_password_enable attribute can only be set when there are no jobs, or all jobs have a “bad password” hold, and can only be disabled when no jobs exist.

• A user must set a password at any server at which the user wishes to run jobs, if that server has single_signon_password_enable set to True. For example, if a user wishes to move a job to another server via the qmove command, and the receiving server has single_signon_password_enable set to True, the user’s password
must first be stored at the receiving server via the `pbs_password` command. This applies to jobs moved for peer scheduling as well.

### 9.13.1.1.4 Single Signon and Invalid Passwords

If a job's originating Server has `single_signon_password_enable` set to `True`, and the job fails to run due to a bad password, the Server will place a hold on the job of type “p” (bad password hold), update the job’s comment with the reason for the hold, and email the user with possible remedy actions. The user (or a manager) can release this hold type via:

```
qrls -h p <jobid>
```

### 9.13.1.2 Per-job Passwords

If your PBS complex contains UNIX or Linux machines whose platform is not the supported architecture for HPCBP submission hosts, use this method.

Users can submit a password for each job that requires a password by using the `-Wpwd` option to the `qsub` command. For example:

```
qsub -lselect = 2:mem=2GB -Wpwd
```

Users cannot use this method when the server’s `single_signon_password_enable` attribute is set to `True`.

### 9.13.2 UNIX and Linux Only

If your PBS complex contains only UNIX and Linux machines, and does not send jobs to an HPC Basic Profile Server, you do not need user passwords. Make sure that the server’s `single_signon_password_enable` attribute is set to `False` or is unset, otherwise some jobs will not run.
9.14 File and Directory Permissions

Each parent directory above PBS_HOME must be owned by root and writable by root only. All files and directories used by PBS should be writable by root only. Permissions should allow read access for all files and directories except those that are private to the daemons. The following should not be writable by any but root:

- PBS_HOME/mom_priv
- PBS_HOME/sched_priv
- PBS_HOME/server_priv

The PBS_HOME directory must be readable and writable from server hosts by root (Administrator) on UNIX/Linux.

On Windows, PBS_HOME must have Full Control permissions for the local "Administrators" group on the local host.

9.15 Windows Caveats

The PBS installer installs the Microsoft 2005 redistributable pack. Please refer to the Microsoft documentation for further details on this package.
Chapter 10

Making Your Site More Robust

This chapter describes how to configure PBS to make your site more robust. If PBS is not already installed, install it according to the PBS Professional Installation & Upgrade Guide.

10.1 Robustness

PBS provides the following mechanisms that support site robustness and flexibility:

Failover

The PBS complex can run a backup server. If the primary server fails, the secondary takes over without an interruption in service.
10.2 Failover

10.2.1 Terminology

Primary Server
The PBS Professional server daemon which is running during normal operation.

Secondary Server
The PBS Professional server daemon which takes over when the primary server fails.

Primary Scheduler
The PBS Professional scheduler daemon which is running during normal operation.

Secondary Scheduler
The PBS Professional scheduler daemon which takes over when the primary scheduler is not available.

Active
A server daemon is active when it is managing user requests and communicating with the scheduler and MOMs.
Idle

A server daemon is idle when it is running, but only accepting handshake messages, not performing workload management.

10.2.2 How Failover Works

During normal operation, the primary server is active and the secondary server is idle. If the primary server fails for any reason, the secondary server becomes active and takes over server functions for the complex. No work is lost during the transition between servers. PBS functions the same during failover as it does during normal operation. The PBS data service is considered to be part of the PBS server; if it fails, this triggers failover.

10.2.2.1 Primary and Secondary Schedulers

Each server is paired with its own scheduler. The primary server can use only the primary scheduler, but the secondary server can use either scheduler. The secondary server will use the primary scheduler if possible, but will start its own scheduler if necessary.

10.2.2.1.1 Starting Secondary Scheduler Under Windows

The following information is also included in the configuration instructions later; it is here for explanation only. Under Windows, the secondary server only starts a scheduler if it needs to when it takes over from the primary server, and not when the host is booted. To achieve this, the pbs.conf file on the secondary server host contains the line, “PBS_START_SCHED=1”, which is different from UNIX/Linux, and the startup type for the scheduler is “Manual”. The server starts the scheduler manually using “net start pbs_sched”.

10.2.2.2 Primary and Secondary Data Services

Each server uses a data service. The primary server uses only its own data service. The secondary server first attempts to use the primary data service, but will start its own if necessary.
10.2.2.3 Normal Post-configuration Behavior

After you have configured PBS for failover, and started both servers, the secondary server periodically attempts to connect to the primary server until it succeeds and registers itself with the primary server. The secondary server must register itself in order to take over upon failure of the primary server.

10.2.2.4 Behavior During Failover

When both server daemons are running, the primary server sends periodic handshake messages to the secondary. If the secondary server stops receiving handshake messages from the primary server, the following happens:

- The secondary server waits for a specified delay period before taking over. This delay is specified using the `pbs_server -F` option. The default period is 30 seconds.
  - The secondary server records the timestamp of the `PBS_HOME/server_priv/svrlive` file
  - The secondary waits for the specified delay, then checks the time stamp again
  - If the time stamp has changed, the secondary server remains idle
  - If the time stamp has not changed, the secondary attempts to open a new TCP connection to the primary
  - If the secondary server cannot open a TCP connection to the primary, the secondary becomes active
- The secondary server logs a message saying that failover has occurred.
- An email is sent to and from the account defined in the server’s `mail_from` attribute, saying that failover has occurred.
- The secondary server checks whether the primary scheduler is still running.
  - If the primary scheduler is still running, the secondary server uses it.
  - If the primary scheduler is not running, the secondary server starts
the secondary scheduler.

- The secondary server attempts to connect to the data service:
  - The secondary server tries for 20 seconds to connect to the data service on the primary server’s host.
  - If it cannot, it tries for 20 seconds to connect to a data service on the secondary server’s host.
  - If the secondary server cannot connect to a data service on either host, it attempts to start a data service on the secondary server’s host.

- The secondary server notifies all of the MOMs that it is the active server.

- The secondary server begins responding to network connections and accepting requests from client commands such as `qstat` and `qsub`.

### 10.2.2.5 Behavior When Primary Resumes Control

When the primary server starts back up, it takes control from the secondary server, becoming the active server. The secondary server becomes idle and resumes listening for the regular handshake messages from the primary server.

The primary server may have been stopped for any of several reasons. The restart method will vary accordingly. If the host was stopped, the PBS server is restarted automatically when the host is started. If the host is still up but the server was stopped, restart the server. See section 6.1.4.4 “Manually Starting the Server” on page 212 in the PBS Professional Installation & Upgrade Guide.

The primary server uses only its own scheduler and data service. When the primary server resumes control, it starts a data service, and if the secondary scheduler and/or data service is running, they are stopped. No data is lost in the transition.

When the primary has taken control, the secondary logs a message saying so:

```plaintext
received takeover message from primary, going inactive
```
10.2.2.6 Server Name and Job IDs During Failover

The server name and job IDs do not change when the secondary server is active. For example, the primary server is on a host named Primary-Host.example.com, and the secondary server is on a host named SecondaryHost.example.com. When the primary server is active, the server name is PrimaryHost, jobs are given job IDs of the form NNNN.PrimaryHost, and the value of the server_host server attribute is Primary-Host.example.com. When the secondary server is active, the server name is still PrimaryHost, jobs are still given job IDs of the form NNNN.PrimaryHost, but the value of server_host is Secondary-Host.example.com.

The table below summarizes the server name, value of server_host and the IDs given to jobs, when either the primary or secondary server is active.

Table 10-1: Server Name, Job ID and Value of server_host Depending on Which Server is Active

<table>
<thead>
<tr>
<th>Active Server</th>
<th>Primary</th>
<th>Secondary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname</td>
<td>PrimaryHost.example.com</td>
<td>SecondaryHost.example.com</td>
</tr>
<tr>
<td>Server Name</td>
<td>PrimaryHost</td>
<td>PrimaryHost</td>
</tr>
<tr>
<td>Value of server_host</td>
<td>PrimaryHost.example.com</td>
<td>SecondaryHost.example.com</td>
</tr>
<tr>
<td>Job Name</td>
<td>NNNN.PrimaryHost</td>
<td>NNNN.PrimaryHost</td>
</tr>
</tbody>
</table>

10.2.2.7 Information Used by Primary and Secondary Servers

The primary and secondary servers share a single source for attribute information, so anything set via the qmgr command need only be set once. PBS_HOME is in a shared location. License information is shared and needs to be set at only one server.

Each server, execution and client host uses its own pbs.conf file, so these must be set for each host in the complex.
10.2.2.8 Impact on Users

Users will not notice when a failover occurs. When a user uses a PBS command such as qstat, the command tries to connect to the primary server first. If that fails, the command tries the secondary server.

If the secondary server responds to the command, the command creates a local file so that this process is not repeated for every PBS command.

Under UNIX, the file is named:

/tmp/.pbsrc.UID

where UID is the user ID.

Under Windows, the file is named:

%TEMP\.pbsrc.USERNAME

where USERNAME is the user’s login name.

When this file exists, commands try the secondary server first, eliminating the delay in attempting to connect to the down server. If a command cannot connect to the secondary server, and can connect to the primary server, the command removes the file.

The file is removed when the primary server takes over.

10.2.2.9 Determining Which Server Is Active

The server attribute server_host contains the name of the host on which the active server is running. Use the qstat -bf command to see the value of server_host.

10.2.2.10 Delay Between Primary Failure and Secondary Becoming Active

The default delay time from detection of possible primary server failure until the secondary server takes over is 30 seconds. A secondary server on a very reliable network can use a shorter delay. A secondary server on an unreliable network may need to use a longer delay. The delay is specified via the -F option to the pbs_server command.
### 10.2.2.11 Communication with MOMs

The secondary server is automatically added to the list of hosts allowed to connect to MOMs, in the \$clienthost MOM configuration parameter.

### 10.2.3 Windows Locations

Since PBS is installed on 32-bit Windows systems in `\Program Files\PBS Pro\`, and on 64-bit Windows systems in `\Program Files (x86)\PBS Pro\`, we use INSTALLATION_BASE to indicate either `\Program Files\PBS Pro\` or `\Program Files (x86)\PBS Pro\`, whichever is appropriate.

### 10.2.4 Prerequisites for Failover

#### 10.2.4.1 Checklist of Prerequisites for Failover

The following table contains a checklist of the prerequisites for failover. Each entry has a link to more detailed information about the entry.

<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identical server hosts</td>
<td>See section 10.2.4.2 “Server Host Requirements” on page 711</td>
</tr>
<tr>
<td>MOMs on server hosts don’t share a <code>mom_priv</code> directory</td>
<td>See section 10.2.4.3 “Requirements for MOMs on Server Hosts” on page 712</td>
</tr>
<tr>
<td>All hosts must be able to communicate over the network</td>
<td>See section 10.2.4.4 “Communication Between Hosts” on page 713</td>
</tr>
<tr>
<td>All hosts must be able resolve hostnames of other hosts in complex</td>
<td>See section 10.2.4.5 “Hostname Resolution” on page 713</td>
</tr>
</tbody>
</table>
Table 10-2: Prerequisites for Failover

<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filesystem must be shared, on a separate host from either server host, and provide features required for failover</td>
<td>See section 10.2.4.6 “Shared Filesystem” on page 713</td>
</tr>
<tr>
<td>Administrator must have access to filesystem from both server hosts</td>
<td>See section 10.2.4.7 “Permission Requirements” on page 716</td>
</tr>
<tr>
<td>Same version of PBS for all components</td>
<td>See section 10.2.4.8 “Same PBS Versions Everywhere” on page 717</td>
</tr>
<tr>
<td>Primary server’s scheduler must be able to run when primary server runs</td>
<td>See section 10.2.4.9 “Requirement for Primary Server’s Scheduler” on page 717</td>
</tr>
<tr>
<td>Data service user account must be the same on both primary and secondary server hosts</td>
<td>See section 10.2.4.10 “Same Data Service Account on Both Server Hosts” on page 717</td>
</tr>
<tr>
<td>Data service host must be default</td>
<td>See section 10.2.4.11 “Data Service Host Configuration Requirement” on page 717</td>
</tr>
<tr>
<td>User names must be consistent across primary &amp; secondary servers hosts</td>
<td>See section 10.2.4.12 “Consistent User Names” on page 718</td>
</tr>
<tr>
<td>The mail_from server attribute specifies an email address that is monitored. Not required, but recommended.</td>
<td>See section 10.2.4.13 “Server Mail is Monitored” on page 718</td>
</tr>
</tbody>
</table>

10.2.4.2 Server Host Requirements

The primary and secondary servers must run on two separate host machines. Both host machines must have the same architecture. They must be binary compatible, including word length, byte order, and padding within structures. There must be exactly one primary and one secondary server.
10.2.4.3 Requirements for MOMs on Server Hosts

You can run a MOM on both the primary and secondary server hosts, but this is not recommended.

If a MOM is to run on both server hosts, the two MOMs must not share the same PBS_HOME/mom_priv directory. In addition, it is strongly recommended that the following be true:

- The mom_priv directory structure be replicated on a local, non-shared, filesystem. On Windows, MOM already has a local directory on each server host. On UNIX/Linux, you must create these.

  To replicate the mom_priv directory structure on the primary server’s host if it doesn’t exist there already, do the following on the primary server host:

  ```
  scp -r <existing PBS_HOME/mom_priv> <local PBS_HOME/mom_priv>
  ```

  To replicate the mom_priv directory structure on the secondary server’s host if it doesn’t exist there already, do the following on the secondary server host:

  ```
  scp -r <existing PBS_HOME/mom_priv> <local PBS_HOME/mom_priv>
  ```

- Each MOM use its own, local, mom_priv directory structure

  The PBS_MOM_HOME entry in pbs.conf specifies the location for the mom_priv directory. If PBS_MOM_HOME is specified in pbs.conf, pbs_mom uses that location instead of PBS_HOME.

  To prevent the MOMs from automatically using the same directory, do one of the following:

  - Recommended: Specify the separate, local PBS_MOM_HOME entry in each server host’s pbs.conf file (each pbs_mom will use the location for mom_priv specified in its PBS_MOM_HOME). Give the location of the local PBS_HOME/mom_priv that you replicated on each host. You can perform this step now, or later, when editing pbs.conf on each server host, in

On an SGI ICE, use two different service nodes to run the primary and secondary servers.
section 10.2.5.3 “Host Configuration for Failover on UNIX/Linux” on page 720, or section 10.2.5.4 “Host Configuration for Failover on Windows” on page 724.

- Use the -d option when starting at least one pbs_mom to specify that they use the local, non-default locations for mom_priv

10.2.4.4 Communication Between Hosts

Both the primary and secondary server hosts must be able to communicate over the network with each other and all execution hosts.

Beware of dependencies on remote file systems: The $PBS_CONF environment variable must point to pbs.conf. PBS depends on the paths in pbs.conf being available when its startup script is executed. PBS will hang if a remote file access hangs, and normal privileges don’t necessarily carry over for access to remote file systems. For example, a FAT filesystem mounted via NFS won’t support permissions.

10.2.4.5 Hostname Resolution

Hostname resolution must work between each host in the PBS complex. Make sure that all hosts in the complex (the primary and secondary server hosts, the file server host, and all execution and client hosts) are set up so that they can resolve the names of all other hosts in the complex. If you are not sure whether hostname resolution is working, run the pbs_hostname command at each host, testing the hostnames of the other hosts. The pbs_hostname command will return the canonical hostname of the specified host.

10.2.4.6 Shared Filesystem

The filesystem you use for the machines managed by PBS should be highly reliable. We recommend, in this order, the following filesystems:

- HA DAS
- DAS, such as xfs or gfs
- HA NFS
- NFS
PBS_HOME is the top directory used by the PBS server. The primary and secondary servers share the same PBS_HOME directory. The PBS_HOME directory must conform to the following:

- The PBS_HOME directory must be available under the same name to both the primary and secondary server hosts.
- The PBS_HOME directory must be on a file system which meets the following requirements:
  - It should reside on a different machine from either of the server hosts.
  - It must be shared by the primary and secondary server hosts.
  - It must be reliable. The file system must be always available to both the primary and secondary servers. A failure of the file system will stop PBS from working.
  - The file system protocol must provide file locking support.
  - The file locking daemons must be running.
  - For UNIX/Linux, the filesystem must support POSIX (Open Group) file semantics.
  - It must support concurrent read and write access from two hosts.
  - It must support multiple export/mounting.

If your filesystem does not conform to the specifications above, follow the steps in the next sections.

10.2.4.6.1 Using NFS Filesystems

When using NFS for PBS_EXEC, NFS must be configured to allow root access and to allow setuid-root programs to execute from it.

If possible, mount NFS file systems synchronously (without caching) to avoid reliability problems.

NFS filesystems should be hard mounted.
10.2.4.6.2 Setting Up the Shared Filesystem on UNIX/Linux

You can use NFS or another filesystem protocol to set up the shared file system on which PBS_HOME resides. Examples are SGI CXFS, IBM GPFS, and Red Hat GFS. Make sure your protocol supports:

- Multiple export/mounting
- Simultaneous read/write from two hosts
- File locking support

To set up your file system:

1. Choose a machine for the file server host. This machine must not be either of the server hosts.
2. Make sure the file system is mounted by both the primary and secondary server hosts. For NFS, make sure the file system is hard mounted by both hosts.
3. Make sure the file system can provide file locking. For NFS, the lock daemon, lockd, must be running.
4. Make sure that PBS_HOME is available under the same name to both the primary and secondary server hosts.

10.2.4.6.3 Setting up a Shared PBS_HOME on Windows

Under Windows, you can use the network share facility to set up the shared PBS_HOME. The primary and secondary servers will share a PBS_HOME directory that is located on a network share file system on a non-server host.
To set up your file system:

1. Choose a machine for the file system host. This machine must not be either of the server hosts. Log on to this machine.

2. On the file system host, create a folder named, for example, C:\pbs_home.

3. Using Windows Explorer, select the C:\pbs_home folder by right-clicking it, and choose “Properties”.

4. Select the Sharing tab, and click the checkbox that says "Share this folder"; specify "Full Control" permissions for the local Administrators group on the local computer.

5. Make sure that PBS_HOME is available under the same name to both the primary and secondary server hosts.

6. Make sure the file system can provide file locking.

7. Copy the files from the local PBS home directory (where it was put during installation) onto the shared PBS_HOME directory.

   On Windows XP:
   
   xcopy /o /e "\Program Files\PBS Pro\home" "\<shared filesystem host>\pbs_home"

   On Vista, 32-bit:
   
   robocopy "\Program Files\PBS Pro\home" "\<shared filesystem host>\pbs_home" /R

   On Vista, 64-bit:
   
   robocopy "\Program Files (x86)\PBS Pro\home" "\<shared filesystem host>\pbs_home" /R

   Make sure that the copy of each file is successful. For any files that failed to be copied, perform the copy by hand.

### 10.2.4.7 Permission Requirements

The PBS_HOME directory must meet the security requirements of PBS. Each parent directory above PBS_HOME must be owned by root and writable by root only.
The **PBS_HOME** directory must be readable and writable from both server hosts by root (Administrator) on UNIX/Linux.

On Windows, **PBS_HOME** must have Full Control permissions for the local "Administrators" group on the local host. Example:

```
Z:\ BUILTIN\Administrators:F
<domain>\<PBS service account>:F
BUILTIN\Administrators:F
Everyone:R
```

### 10.2.4.8 Same PBS Versions Everywhere

Both server hosts, all the execution hosts, and all the client hosts must run the same version of PBS Professional.

### 10.2.4.9 Requirement for Primary Server’s Scheduler

The primary scheduler must be able to run whenever the primary server is running. The primary server can use only the primary scheduler. If the primary server becomes active but cannot use its own scheduler, PBS will not be able to schedule jobs.

### 10.2.4.10 Same Data Service Account on Both Server Hosts

The data service account must be the same on both server hosts. The UID of the data service account must be identical on both the primary and secondary server hosts. We recommend that the data service account is called `pbsdata`.

If you change either service account, both must be changed at the same time and both servers must be restarted.

### 10.2.4.11 Data Service Host Configuration Requirement

The **DATA_SERVICE_HOST** parameter must not be set in `pbs.conf`. If this parameter is set, failover cannot take place.
10.2.4.12 Consistent User Names

User names must be consistent across the primary and secondary server hosts. If user names are not consistent, jobs are killed.

10.2.4.13 Server Mail is Monitored

Use the `qmgr` command to set the `mail_from` server attribute to an address that is monitored regularly:

```
Qmgr: s server mail_from=<address>
```

10.2.5 Configuring Failover

10.2.5.1 Overview of Configuring Failover

If PBS is not already installed, install it according to the PBS Professional Installation & Upgrade Guide.

Please make sure that you have satisfied all of the prerequisites under section 10.2.4 “Prerequisites for Failover” on page 710.

The following table contains a guide to the steps in configuring PBS for failover for both UNIX/Linux and Windows. The table contains a link to the description of each step.

<table>
<thead>
<tr>
<th>Step</th>
<th>UNIX/Linux</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure <code>/etc/pbs.conf</code> on each host in the complex</td>
<td>See section 10.2.5.2 “Configuring the pbs.conf File” on page 719</td>
<td>See section 10.2.5.4.1 “Configuring Failover for the Primary Server on Windows” on page 726</td>
</tr>
<tr>
<td>Configure the primary server</td>
<td>See section 10.2.5.3.1 “Configuring Failover For the Primary Server on UNIX/Linux” on page 721</td>
<td></td>
</tr>
</tbody>
</table>
Table 10-3: Overview of Configuring Failover

<table>
<thead>
<tr>
<th>Step</th>
<th>UNIX/Linux</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure the secondary server</td>
<td>See section 10.2.5.3.2 “Configuring Failover For the Secondary Server on UNIX/Linux” on page 723</td>
<td>See section 10.2.5.4.2 “Configuring Failover for the Secondary Server on Windows” on page 728</td>
</tr>
<tr>
<td>Configure execution and client hosts</td>
<td>See section 10.2.5.3.3 “Configuring Failover For Execution and Client Hosts on UNIX/Linux” on page 724</td>
<td>See section 10.2.5.4.3 “Configuring Failover for Execution and Client Hosts on Windows” on page 730</td>
</tr>
<tr>
<td>Configure failover with peer scheduling</td>
<td>See section 10.2.6.2 “Configuring Failover to Work With Peer Scheduling” on page 731</td>
<td></td>
</tr>
<tr>
<td>Configure failover with routing queues</td>
<td>See section 10.2.6.1 “Configuring Failover to Work with Routing Queues” on page 730</td>
<td></td>
</tr>
<tr>
<td>Configure failover with access control</td>
<td>See section 10.2.6.3 “Configuring Failover to Work With Access Controls” on page 731</td>
<td></td>
</tr>
</tbody>
</table>

### 10.2.5.2 Configuring the pbs.conf File

The `$PBS_CONF` environment variable contains the path to the `pbs.conf` file. Each host in the complex must have a properly configured `/etc/pbs.conf` file. This file specifies the hostnames of the primary and secondary servers, the location of `PBS_HOME` and `PBS_MOM_HOME`, and whether to start a server, a scheduler, or a MOM on this host.

The name used for the server in the `PBS_SERVER` variable in the `pbs.conf` file must not be longer than 64 characters. If the short name for the server resolves to the correct host, you can use this in `pbs.conf`
as the value of PBS_SERVER. However, if the fully-qualified domain name is required in order to resolve to the correct host, then this must be the value of the PBS_SERVER variable.

**Table 10-4: Parameters in pbs.conf for Failover**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_EXEC</td>
<td>Path</td>
<td>Location of PBS bin and sbin directories</td>
</tr>
<tr>
<td>PBS_HOME</td>
<td>Path</td>
<td>Location of PBS working directories in shared filesystem; use specific path on that host</td>
</tr>
<tr>
<td>PBS_MOM_HOME</td>
<td>Path</td>
<td>Location of mom_priv on each host; overrides PBS_HOME for mom_priv</td>
</tr>
<tr>
<td>PBS_PRIMARY</td>
<td>FQDN of hostname</td>
<td>Hostname of primary server host</td>
</tr>
<tr>
<td>PBS_SECONDARY</td>
<td>FQDN of hostname</td>
<td>Hostname of secondary server host</td>
</tr>
<tr>
<td>PBS_SERVER</td>
<td>Hostname</td>
<td>Fully-qualified domain name of primary server host</td>
</tr>
<tr>
<td>PBS_START_MOM</td>
<td>0 or 1</td>
<td>Specifies whether a MOM is to run on this host</td>
</tr>
<tr>
<td>PBS_START_SCHED</td>
<td>0 or 1</td>
<td>Specifies whether scheduler is to run on this host</td>
</tr>
<tr>
<td>PBS_START_SERVER</td>
<td>0 or 1</td>
<td>Specifies whether server is to run on this host</td>
</tr>
</tbody>
</table>

**10.2.5.3 Host Configuration for Failover on UNIX/Linux**

- Make sure that you have satisfied all of the prerequisites under section 10.2.4 “Prerequisites for Failover” on page 710.
- PBS should already be installed in the default location on the primary and secondary server hosts and on the execution hosts. The client com-
mands should already be installed on the client hosts.

- If the primary server and scheduler are running, shut them down. See “qterm” on page 268 of the PBS Professional Reference Guide.

10.2.5.3.1 Configuring Failover For the Primary Server on UNIX/Linux

1. Make sure that you have satisfied all of the prerequisites under section 10.2.4 “Prerequisites for Failover” on page 710.

2. Stop PBS on both the primary and secondary server hosts:

   On the primary server’s host:
   
   `<path to init.d>/init.d/pbs stop`

   On the secondary server’s host:
   
   `<path to init.d>/init.d/pbs stop`

3. On the primary server’s host, edit the `/etc/pbs.conf` file so that it DOES NOT include failover settings. It should look like this:

   ```
   PBS_SERVER=<short name for primary host>
   PBS_HOME=<shared location of PBS_HOME>
   PBS_START_SCHED=1
   ```

   It is recommended not to run a MOM on both server hosts. The following setting in `pbs.conf` will prevent a MOM from running:

   ```
   PBS_START_MOM=0
   ```

   If you will run a MOM on the server hosts, specify this:

   ```
   PBS_START_MOM=1
   ```

   If you will run a MOM on both server hosts, specify `PBS_MOM_HOME` on this host. The location you specify is the directory that you repli-
cated in section 10.2.4.3 “Requirements for MOMs on Server Hosts” on page 712:

\[
\text{PBS\_MOM\_HOME}=\text{<location of local, replicated mom\_priv>}
\]

4. On the primary server’s host, start the primary PBS server and scheduler daemons:

\[
\text{<path to init.d>/init.d/pbs start}
\]

5. Stop the PBS server on the primary server’s host:

\[
\text{<path to init.d>/init.d/pbs stop}
\]

6. On the primary server’s host, edit the /etc/pbs.conf file to include the failover settings for PBS\_PRIMARY and PBS\_SECONDARY. It should look like this:

\[
\begin{align*}
\text{PBS\_PRIMARY}= & \text{<primary_host>} \\
\text{PBS\_SECONDARY}= & \text{<secondary_host>} \\
\text{PBS\_SERVER}= & \text{<short name for primary host>} \\
\text{PBS\_HOME}= & \text{<shared location of PBS\_HOME>}
\end{align*}
\]

The primary scheduler will start automatically:

\[
\text{PBS\_START\_SCHED}=1
\]

It is recommended not to run a MOM on both server hosts. The following setting in pbs.conf will prevent a MOM from running:

\[
\text{PBS\_START\_MOM}=0
\]

If you will run a MOM on the server hosts, specify this:

\[
\text{PBS\_START\_MOM}=1
\]

If you will run a MOM on both server hosts, specify PBS\_MOM\_HOME on this host. The location you specify is the directory that you replicated in section 10.2.4.3 “Requirements for MOMs on Server Hosts” on page 712:

\[
\text{PBS\_MOM\_HOME}=\text{<location of local, replicated mom\_priv>}
\]

7. On the primary server’s host, start the primary PBS server and scheduler daemons:

\[
\text{<path to init.d>/init.d/pbs start}
\]
10.2.5.3.2 Configuring Failover For the Secondary Server on UNIX/Linux

1. Make sure that you have satisfied all of the prerequisites under section 10.2.4 “Prerequisites for Failover” on page 710.

2. On the secondary server’s host, edit the /etc/pbs.conf file to include the following settings:

   PBS_PRIMARY=<primary_host>
   PBS_SECONDARY=<secondary_host>
   PBS_SERVER=<short name for primary host>
   PBS_HOME=<shared location of PBS_HOME>

   The secondary server will start its own scheduler if it needs to; a scheduler should not automatically start on the secondary server host. Include the following so that a scheduler does not automatically start on this host:

   PBS_START_SCHED=0

   It is recommended not to run a MOM on both server hosts. The following setting in pbs.conf will prevent a MOM from running:

   PBS_START_MOM=0

   If you will run a MOM on the server hosts, specify this:

   PBS_START_MOM=1

   If you will run a MOM on both server hosts, specify PBS_MOM_HOME on this host. The location you specify is the directory that you replicated in section 10.2.4.3 “Requirements for MOMs on Server Hosts” on page 712:

   PBS_MOM_HOME=<location of local, replicated mom_priv>

3. On the secondary server’s host, to change the delay time between failure of the primary server and activation of the secondary server from its default of 30 seconds, use the -F <delay> option on the secondary server's command line in the PBS start script on the secondary server’s
host. Edit the init.d/pbs script so that the server is invoked with the -F <delay> option:

```
pbs_server -F <delay>
```

See “pbs_server” on page 108 of the PBS Professional Reference Guide.

4. On the secondary server’s host, start the secondary PBS server daemon:

```
<path to init.d>/init.d/pbs start
```

### 10.2.5.3 Configuring Failover For Execution and Client Hosts on UNIX/Linux

1. Make sure that you have satisfied all of the prerequisites under section 10.2.4 “Prerequisites for Failover” on page 710.

2. On each execution or client host, configure the /etc/pbs.conf file to include the following parameters:

```
PBS_PRIMARY=<primary_host>
PBS_SECONDARY=<secondary_host>
PBS_SERVER=<short name for primary host>
PBS_HOME=<location of PBS_HOME>
```

The pbs.conf files on execution hosts are already configured to start the MOM daemon only. Similarly, the pbs.conf files on client hosts are already configured to start no daemons.

3. On each execution host, restart the MOM:

```
<path to init.d>/init.d/pbs start
```

### 10.2.5.4 Host Configuration for Failover on Windows

- PBS should already be installed in the default location on the primary and secondary server hosts and on the execution and client hosts. See the PBS Professional Installation and Upgrade Guide for instructions.
- Please make sure that you have satisfied all of the prerequisites under section 10.2.4 “Prerequisites for Failover” on page 710.
- Configure server failover from the console of each host or through VNC. Do not use Remote Desktop. Setting up the server failover fea-
ture from a Remote Desktop will cause problems. In particular, starting
the server on either the primary or secondary server host will lead to the
error:

```
error 1056: Service already running
```

This will happen even though

```
PBS_HOME\server_priv\server.lock and
PBS_HOME\server_priv\server.lock.secondary files do
not exist.
```

- Stop all the PBS services on the primary and secondary server hosts. 
  On each host:

```
net stop pbs_server
net stop pbs_mom
net stop pbs_sched
net stop pbs_rshd
```
10.2.5.4.1 Configuring Failover for the Primary Server on Windows

1. Make sure that you have satisfied all of the prerequisites under section 10.2.4 “Prerequisites for Failover” on page 710.

2. On the primary server host, specify the location of PBS_HOME for the primary server:

   `pbs-config-add "PBS_HOME=\\<shared filesystem host>\pbs_home"`

3. On the primary server host, specify the primary and secondary server names in the `pbs.conf` file by running the following commands:

   `pbs-config-add "PBS_SERVER=<short name of primary server host>"`
   
   `pbs-config-add "PBS_PRIMARY=<FQDN of primary server host>"`
   
   `pbs-config-add "PBS_SECONDARY=<FQDN of secondary server host>"`

4. On the primary server host, specify that the primary server host will run a server:

   `pbs-config-add "PBS_START_SERVER=1"`

5. On the primary server host, specify that the primary scheduler will automatically start:

   `pbs-config-add "PBS_START_SCHED=1"`

6. It is recommended not to run a MOM on both server hosts. The following setting in `pbs.conf` will prevent a MOM from running. On the primary server’s host:

   `pbs-config-add "PBS_START_MOM=0"`

   If you will run a MOM on the primary server’s host, indicate that a MOM should run, and specify the local PBS_MOM_HOME by running the following commands:

   `pbs-config-add "PBS_START_MOM=1"`
   `pbs-config-add "PBS_MOM_HOME=C:INSTALLATION_BASE\home"`

7. If you are going to run a MOM on this host, start it:
net start pbs_mom

8. Now start the PBS server, scheduler and rshd on this host:

    net start pbs_server
    net start pbs_sched
    net start pbs_rshd

9. Set the managers attribute on the primary server so that when the secondary server takes over, you can still run privileged tasks under the Administrator account or from a peer pbs_server:

    Qmgr: set server managers="<account that installed PBS>@*,<name of PBS service account>@*"
10.2.5.4.2 Configuring Failover for the Secondary Server on Windows

1. Make sure that you have satisfied all of the prerequisites under section 10.2.4 “Prerequisites for Failover” on page 710.

2. On the secondary server host, specify the location of PBS_HOME for the secondary server:

   ```
   pbs-config-add "PBS_HOME=\<shared filesystem host>\pbs_home"
   ```

3. On the secondary server host, specify the primary and secondary server names in the pbs.conf file by running the following commands:

   ```
   pbs-config-add "PBS_SERVER=<short name of primary server host>"
   pbs-config-add "PBSPRIMARY=<FQDN of primary server host>"
   pbs-config-add "PBSSECONDARY=<FQDN of secondary server host>"
   ```

4. Specify that the secondary server host will run a server:

   ```
   pbs-config-add "PBS_START_SERVER=1"
   ```

5. On the secondary server’s host, modify pbs.conf to specify how the secondary server will start its scheduler:

   ```
   pbs-config-add "PBS_START_SCHED=1"
   ```

6. On the secondary server’s host, specify the startup type for the scheduler:

   Go to Control Panel->Administrative Tools->Services, and bring up the PBS_SCHED service dialog, select the General tab, and specify "Manual" for startup type.

7. It is recommended not to run a MOM on both server hosts. The following setting in pbs.conf will prevent a MOM from running. On the secondary server’s host:

   ```
   pbs-config-add "PBS_START_MOM=0"
   ```

   If you will run a MOM on the secondary server’s host, indicate that a MOM should run, and specify the local PBS_MOM_HOME by running
the following commands:

```
pbs-config-add "PBS_START_MOM=1"
pbs-config-add "PBS_MOM_HOME=C:INSTALLATION_BASE\home"
```

**8.** If you are going to run a MOM on this host, start it:

```
net start pbs_mom
```

**9.** Now start the secondary server on this host:

```
net start pbs_server
```

You will see the following message:

"PBS_SERVER could not be started"

This message appears because the secondary server is inactive, waiting for the primary server to fail.

**10.** On the secondary server’s host, to change the delay time between failure of the primary server and activation of the secondary server from its default of 30 seconds, use Start Menu->Control Panel->Administrative Tools->Services, choosing PBS_SERVER, and specify in the “Start Parameters” entry box:

```
-F <delay>
```

Then restart the secondary server.

The Services dialog does not save the “Start Parameters” value for future restarts. The default delay value specified in the PBS start script, or the default if there is none, will be in effect on subsequent restarts.
10.2.5.4.3 Configuring Failover for Execution and Client Hosts on Windows

1. Make sure that you have satisfied all of the prerequisites under section 10.2.4 “Prerequisites for Failover” on page 710.

2. On each execution or client host, specify the location of PBS_HOME for the primary server:

```
pbs-config-add "PBS_HOME=\<shared filesystem host>\pbs_home"
```

3. On each execution or client host, specify the primary and secondary server names in the `pbs.conf` file by running the following commands:

```
pbs-config-add "PBS_SERVER=<short name of primary server host>"
pbs-config-add "PBS_PRIMARY=<FQDN of primary server host>"
pbs-config-add "PBS_SECONDARY=<FQDN of secondary server host>"
```

4. If this is an execution host, restart the MOM:

```
net start pbs_mom
```

10.2.6 Configuring Failover with Other PBS Features

10.2.6.1 Configuring Failover to Work with Routing Queues

You must configure failover to work with routing queues which have destinations in another complex. No additional configuration is required for routing queues which have destinations in the same complex.

For a routing queue in one complex which points to a queue Q1 in another PBS complex that is set up for failover, it is a good idea to specify both `Q1@primary.example.com` and `Q1@secondary.example.com` as destinations.
For example, if a routing queue has a destination queue at another complex’s primary server:

```
Qmgr: set queue r66
    route_destinations=workq@primary.example.com
```
you need to add the same queue at the other complex’s secondary server:

```
Qmgr: set queue r66
    route_destinations+=workq@secondary.example.com
```
See section 2.2.6 “Routing Queues” on page 26.

### 10.2.6.2 Configuring Failover to Work With Peer Scheduling

For peer queueing where the furnishing complex is set up for failover:

- You must list the furnishing queue at both primary and secondary servers. If the furnishing queue is Q1, the `peer_queue` line in the pulling complex’s `sched_config` file must list `Q1@primary.example.com` and `Q1@secondary.example.com`

For peer queueing where the pulling complex is set up for failover:

- You must add `<manager>@primary.example.com` and `<manager>@secondary.example.com` to the list of managers at the furnishing server.

See section 4.8.31 “Peer Scheduling” on page 229.

### 10.2.6.3 Configuring Failover to Work With Access Controls

If you are using access control on the server (the `acl_host_enable` server attribute is set to `True` and the `acl_hosts` server attribute is specified), add the secondary server to the host list in `acl_hosts`:

```
Qmgr: s server acl_hosts+=<secondary server host>
```
See section 9.3.4 “ACLs” on page 660.
10.2.7 Using PBS with Failover Configured

10.2.7.1 Stopping Servers

To stop both servers when the primary server is active, and the secondary server is running and idle, do the following:

```
qterm -f
```

To stop the primary server and leave the secondary server idle:

```
qterm -i
```

To stop the secondary server only:

```
qterm -F
```

10.2.7.2 Starting Servers

After configuring the servers, you can start them in any order.

If you want to start the primary server when the secondary server is the active server, you do not need to stop the secondary. When the primary server starts, it informs the secondary that the secondary can become idle.

However, if there is a network outage while the primary starts and the secondary cannot contact it, the secondary will assume the primary is still down, and remain active, resulting in two active servers. In this case, stop the secondary server, and restart it when the network is working:

UNIX/Linux:

```
qterm -F
pbs_server
```

Windows:

```
qterm -F
net start pbs_server
```
To restart the secondary server while it is the active server:

```
pbs_server -F -1
```

The secondary server makes one attempt to contact the primary server, and becomes active immediately if it cannot.


### 10.2.8 Recommendations and Caveats

- If you do not wish for the secondary server to take over, use the -i option to the `qterm` command when stopping the primary server.
- When the primary server is active, and the secondary server is running and idle, the `pbs start/stop` script stops the active server, but leaves the idle server running. This means that the idle server becomes the active server.
- `PBS_HOME` should not be on either server host
- Neither PBS server should be the NFS fileserver
- The primary scheduler must be able to run when the primary server is started, otherwise no jobs will be scheduled; the primary server cannot use the secondary scheduler.
- Just because servers are redundant, that doesn't mean that your complex is. Look for single points of failure.
- If the “take over” delay time specified with the `pbs_server -F` option is too long, there may be a period, up to that amount of time, when clients cannot connect to either server.
- If the “take over” delay time specified with the `pbs_server -F` option is too short and there are transient network failures, then the secondary server may attempt to take over while the primary server is still active.
- While the primary server is active and the secondary server is inactive, the secondary server will not respond to any network connection attempts. Therefore, you cannot status the secondary server to determine whether it is running.
- If you start the secondary server manually, it will not start its own
scheduler. You must start the secondary server’s scheduler manually, if required.

- If the secondary server is running, and the primary server cannot contact the secondary server when the primary server is restarted, the primary assumes the secondary is not running and takes over. This can result in two servers running at once.

- Do not run the pbs_ds_password command if you have configured PBS for failover operation. It is important not to inadvertently start two separate instances of the data service on two machines, thus potentially corrupting the database.

### 10.2.9 Troubleshooting Failover

#### 10.2.9.1 PBS Won’t Start

- If you see the following error:

  "Failover is configured. Temporarily disable failover before running pbs_ds_password"

  This means that PBS was started for the first time with failover configured. PBS cannot be started for the first time with failover configured. Remove definitions for PBS_PRIMARY and PBS_SECONDARY from pbs.conf on the primary server’s host, start PBS, stop PBS, replace the definitions, and start PBS again.

#### 10.2.9.2 Primary and Secondary Servers Both Running

If both servers are running, this may be because the primary server was stopped and then restarted, and while the primary was stopped, the secondary began to take over. While the secondary server was coming up, it was not able to receive the message from the primary server indicating that it should go idle, or it couldn’t register with the primary.

To avoid this problem, use the -i option to the qterm command, which tells the secondary server to remain idle.
10.2.9.3 Primary or Secondary Server Fails to Start

It does not matter in which order the primary and secondary servers are started.

If the primary or secondary server fails to start with the error:

```
another server running
```

then check for the following conditions:

1. There may be lock files left in \texttt{PBS\_HOME/server\_priv} that need to be removed.

   The primary and secondary servers use different lock files:
   - primary: \texttt{server.lock}
   - secondary: \texttt{server.lock.secondary}

2. On UNIX, the RPC \texttt{lockd} daemon may not be running. You can manually start this daemon by running as root:

   ```
   <path to daemon>/rpc.lockd
   ```

   Check that all daemons required by your NFS are running.

10.2.9.4 Windows: Primary Server Fails to Resume Control

If the primary server fails to take over for the secondary server when it should, reboot the primary server’s host.

10.3 Checkpoint and Restart

PBS Professional allows you to configure MOM to checkpoint jobs using your scripts and checkpoint tools. In addition, users may manage their own checkpointing from within their application.
10.3.1 Terminology

Application Checkpoint
The application performs its own checkpointing when it receives the appropriate signal etc.

Checkpoint and Abort, checkpoint_abort
The checkpoint script or tool writes a restart file, then PBS kills and requeues the job. The job uses the restart file when it resumes execution.

Restart
A job that was stopped after being checkpointed while previously executing is executed again, starting from the point where it was checkpointed.

Restart File
The job-specific file that is written by the checkpoint script or tool. This file contains any information needed to restart the job from where it was when it was checkpointed.

Restart Script
The script that MOM runs to restart a job. This script is common to all jobs, and so must use the information in a job’s restart file to restart the job.

Snapshot Checkpoint
The checkpoint script or tool writes a restart file, and the job continues to execute. The job resumes based on this restart file if the system experiences a problem during the job’s subsequent execution.

10.3.2 How Checkpointing Works

When a job is checkpointed, MOM executes a checkpoint script. The checkpoint script saves all of the information necessary to checkpoint the job. If the checkpoint is for a snapshot, the job continues to run. If the job is checkpointed and aborted, PBS kills and requeues the job after checkpointing it.
When a job is restarted, MOM executes a restart script. The restart script uses the saved information to restore the job. The manner of restarting the job depends on how it was checkpointed:

- If the job was checkpointed during shutdown, the job becomes eligible to run when PBS is restarted, and will start from where it was checkpointed.
- If the job was checkpointed by the scheduler because it was preempted, the scheduler briefly applies a hold, but releases the hold immediately after checkpointing the job, and runs the restart script when the job is scheduled to run.
- If the job was checkpointed and held via the `qhold` command, the hold must be released via the `qrls` command for the job to be eligible to run. Then when the scheduler next runs the job, the restart script is executed, and the job runs from where it was checkpointed.

A restarted job always runs on the same machine where it was running when it was checkpointed. On Cray systems, a job will run on the original login node, but job processes are not guaranteed to run on the same compute nodes as before the checkpoint.

You can configure PBS to requeue jobs that were snapshot checkpointed while they ran, if the epilogue exits with a special value. These jobs are then restarted from the restart file.

You can provide checkpointing for jobs using any combination of scripts that you write and third-party checkpointing tools such as Meiosys Checkpoint and BLCR (Berkeley Lab Checkpoint/Restart). You can configure PBS to trigger the scripts or tools, so that the scripts and/or tools create a job’s restart file.

You can configure one behavior for snapshots, and another behavior for checkpoint and abort.

Some applications provide their own checkpointing, which is triggered, for example, when the application receives a signal or detects a change in a file.
10.3.2.1 Types of Checkpointing

10.3.2.1.1 Checkpoint and Abort

Checkpoint and abort is used when a job is checkpointed before being killed. When the job is checkpointed, the following takes place:

- MOM runs the checkpoint_abort script; the checkpoint script or tool writes a restart file specific to that job
- The checkpoint_abort script terminates the job
- PBS requeues the job
- If the job was held via the qhold command, PBS applies a hold to the job (puts it in the Held state)

The job resumes execution based on the information in the restart file.

Checkpoint and abort is applied when:

- The qhold command is used on a job
- The server is shut down via qterm -t immediate or qterm -t delay
- The scheduler preempts a job using the checkpoint method

10.3.2.1.2 Snapshot Checkpoint

Snapshot checkpointing is used for checkpointing a job at regular intervals. The job continues to run. When the job is checkpointed, the following takes place:

- MOM runs the snapshot checkpoint script; the checkpoint script or tool writes a restart file specific to that job
- The job continues to execute

The job resumes execution based on this restart file if the system crashes or if the epilogue returns -2. See section 10.3.7.3 “Requeueing via Epilogue” on page 757.
The interval can be specified by the user via `qsub -c <checkpoint spec>`. You can specify a default interval, in the `checkpoint_min` queue attribute, or in the `Checkpoint` job attribute. See “qsub” on page 245 of the PBS Professional Reference Guide and “Job Attributes” on page 430 of the PBS Professional Reference Guide.

### 10.3.2.1.3 Application Checkpoint

Application checkpointing is when an application checkpoints itself. PBS can be used to trigger application checkpointing, but does not manage the checkpoint files or process. Application checkpointing can be triggered when the application receives a signal or detects a change in a file.

### 10.3.2.2 Events That Trigger Checkpointing

The following table lists the events that can trigger checkpointing, and the kind of checkpointing that is used.

<table>
<thead>
<tr>
<th>Event</th>
<th>Type of Checkpointing Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The <code>qhold</code> command is used on a job</td>
<td><code>checkpoint_abort</code></td>
<td>See section 10.3.7.6 “Holding a Job” on page 759</td>
</tr>
<tr>
<td>Server shut down via <code>qterm -t immediate</code> or <code>qterm -t delay</code></td>
<td><code>checkpoint_abort</code></td>
<td>See section 10.3.7.2 “Checkpointing During Shutdown” on page 756</td>
</tr>
<tr>
<td>Scheduler preempts a job using the checkpoint method</td>
<td><code>checkpoint_abort</code></td>
<td>See section 10.3.7.5 “Preemption Using Checkpoint” on page 758</td>
</tr>
</tbody>
</table>
Table 10-5: Events Triggering Checkpointing

<table>
<thead>
<tr>
<th>Event</th>
<th>Type of Checkpointing Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic checkpointing of a job, as specified by <code>qsub -c &lt;checkpoint spec&gt;</code>, or the queue’s checkpoint_min attribute</td>
<td>Snapshot</td>
<td>See section 10.3.7.1 “Periodic Job Checkpointing” on page 756</td>
</tr>
<tr>
<td>Periodic checkpoint of an application, where checkpoint script triggers application checkpoint</td>
<td>Snapshot and application checkpoint</td>
<td>See section 10.3.7.7 “Periodic Application Checkpoint” on page 760</td>
</tr>
<tr>
<td>User sends application checkpoint signal, or user creates checkpoint trigger file</td>
<td>Application checkpoint</td>
<td>See section 10.3.7.8 “Manual Application Checkpoint” on page 760</td>
</tr>
</tbody>
</table>

10.3.2.3 Effect of Checkpointing on Jobs

When a job is checkpointed and aborted (requeued), its accumulated queue waiting time depends on how that time is calculated:

- If you are using eligible time, the accumulated waiting time is preserved
- If you are not using eligible time, the accumulated waiting time is lost

The job exit code for being checkpointed and aborted is -12, named `JOB_EXEC_CHK`.  

When a job is restarted, it runs on the same machine as it did when it was checkpointed. On Cray machines, the job runs on the original login node, but may run on different compute nodes.

10.3.2.4 Effect of Checkpointing on Job Resources

When a job is checkpointed and aborted, all of its resources are freed.
A snapshot checkpoint does not affect a job’s resources.

10.3.2.5 Restarting a Job

When a job is restarted, MOM runs the restart script specified in the $action restart MOM parameter. This script looks in the checkpoint directory (see section 10.3.6.5 “Specifying Checkpoint Path” on page 755) for the restart file for that job. It uses the information in that file to restart the job.

For a job that was checkpointed and aborted because it was held, the job has had a hold placed on it so that it will not be eligible for execution until the hold is released. In order for a checkpointed and held job to be eligible for execution, the hold must be removed using the qr1s command. The job’s owner can remove a User hold, but other holds must be removed by a Manager or Operator. See “qr1s” on page 208 of the PBS Professional Reference Guide.

If the job was preempted via checkpointing, the scheduler releases the hold on the job immediately after checkpointing the job. This will show up in the scheduler’s log file, but the job will not appear to be held because the hold duration is very short.

A job that was checkpointed and requeued during shutdown is not held. This job is eligible for execution as soon as the necessary daemons are back up. See section 10.3.7.4 “Checkpointed Jobs and Server Restart” on page 758.

A job that was snapshot checkpointed and later requeued because the epilogue returned a special exit status is requeued in the Q state, and is eligible to be restarted when the scheduler selects it for execution.

When a checkpointed and aborted job is restarted, MOM resumes tracking the job. She tracks either the original PID of the job, or the PID of the restart script, depending on the setting of the $restart_transmogrify MOM parameter. See section 10.3.4.3 “Setting $restart_transmogrify MOM Parameter” on page 745.
10.3.3 Prerequisites for Checkpointing Jobs

The following are the prerequisites for checkpointing jobs:

- The MOM must be configured for checkpointing
  - Specified checkpoint directories must correspond to available directories (see section 10.3.6.5 “Specifying Checkpoint Path” on page 755)
  - Checkpoint and restart MOM configuration parameters must be specified (see section 10.3.4.2 “Specifying Checkpoint and Restart Parameters” on page 743)
- A checkpointing script or tool must be available for each type of checkpointing to be used

10.3.3.1 Restrictions on Checkpointing

- Checkpointing is not supported for job arrays.
- PBS does not directly support OS-level checkpointing.
- Checkpointing is not supported by the HPCBP MOM.
- You can configure only one snapshot script, so if more than one kind of snapshot checkpointing is required, the script must distinguish which kind of snapshot to perform.
- You can configure only one checkpoint_abort script, so if more than one kind of checkpoint_abort is required, the script must also distinguish which kind of checkpoint_abort to perform.
- You can configure only one restart script. The restart script is run once for each of the job’s tasks, so if some restarts are for application checkpointing, the script must handle those restarts correctly (application restarts may require only one iteration.)
- A restarted job must run on the same machine where it was running when it was checkpointed. Note that on Cray systems, the job runs on the original login node, but may run on different compute nodes.
- Checkpointing cannot be used for interactive jobs. See section 10.3.8 “Advice and Caveats” on page 760.
10.3.4 Configuring Checkpointing

10.3.4.1 Overview of Configuring Checkpointing

You configure checkpointing by editing the MOM configuration file, PBS_HOME/mom_priv/config. You edit MOM configuration parameters to do the following:

- Specify script paths
  - Specify path to checkpoint_abort script, if needed
  - Specify path to snapshot script, if needed
  - Specify path to restart script
- Set $restart_transmogrify MOM parameter to fit your restart script
- Make the checkpoint path match that specified in the restart script

10.3.4.2 Specifying Checkpoint and Restart Parameters

To configure checkpointing, you specify a path to a script that MOM executes when checkpointing is called for. You can specify a separate path/script for each of checkpoint_abort, snapshot, and restart using the following MOM configuration parameters:

- $action checkpoint timeout !path/script script-args
  Specifies snapshot behavior.
- $action checkpoint_abort timeout !path/script script-args
  Specifies checkpoint_abort behavior.
- $action restart timeout !path/script script-args
  Specifies restart behavior.

where

$action
  Specifies that MOM perform the indicated action.

checkpoint
  MOM executes the script specified in path/script once for each of the job’s tasks when a snapshot is called for.
checkpoint_abort
MOM executes the script specified in path/script once for each of the job’s tasks when a checkpoint_abort is called for.

restart
MOM executes the script specified in path/script once for each of the job’s tasks when a restart is called for.

timeout
The number of seconds allowed for the script or tool to execute. The value of the $restart_transmogrify MOM parameter determines whether this limit is applied. Values for $restart_transmogrify, and resulting behavior:

False
If the script/tool does not finish running during this time, it is killed and handled as if it had returned failure.

True
No timeout limit is applied.

path/script
The path to the script, including the name of the script. The path can be absolute or relative. If the path is relative, it is relative to PBS_HOME/mom_priv.

Examples of absolute paths and script names:
/usr/bin/checkpoint/snapshot
/usr/bin/checkpoint/checkpt-abort
/usr/bin/checkpoint/restart

script-args
These are the arguments to the script, if any.
PBS automatically expands some arguments to checkpoint and restart scripts. The following table lists the arguments that are expanded by PBS:

Table 10-6: Checkpoint Script Arguments Expanded by PBS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%globid</td>
<td>Global ID (no longer used)</td>
</tr>
<tr>
<td>%jobid</td>
<td>Job ID</td>
</tr>
<tr>
<td>%sid</td>
<td>Session ID</td>
</tr>
<tr>
<td>%taskid</td>
<td>Task ID</td>
</tr>
<tr>
<td>%path</td>
<td>File or directory name to contain restart files</td>
</tr>
</tbody>
</table>

10.3.4.2.1 Examples of Checkpoint and Restart Parameters

The following are examples of snapshot, checkpoint_abort, and restart MOM parameters:

$\texttt{action checkpoint 60 !/usr/bin/checkpoint/snapshot}$
  \texttt{ %jobid %sid %taskid %path}

$\texttt{action checkpoint\_abort 60 !/usr/bin/checkpoint/}$
  \texttt{ checkpt\_abort %jobid %sid %taskid %path}

$\texttt{action restart 30 !/usr/bin/checkpoint/restart %jobid}$
  \texttt{ %sid %taskid %path}

10.3.4.3 Setting $\texttt{restart\_transmogrify}$ MOM Parameter

The $\texttt{restart\_transmogrify}$ MOM parameter controls how MOM runs the restart script, and whether she expects to resume tracking the job’s original PID or a new PID. When she runs a restart script, MOM forks a child process, which \texttt{exec()}s the start script. If $\texttt{restart\_transmogrify}$ is \texttt{True}, the start script becomes the top task of the job. If $\texttt{restart\_transmogrify}$ is \texttt{False}, the start script does not become the top task of the job.
If your restart script preserves the job’s original PID, set $restart_transmogrify to False. This way, the script does not become the top task of the job, and MOM continues to track the job’s original PID.

If your restart script results in a new PID for the job, set $restart_transmogrify to True. This way, the restart script becomes the top task of the job, and MOM tracks the PID of the new top process, which is the script.

10.3.4.4 Checkpointing on Cray Systems

Checkpointing on the Cray XT uses BLCR. Use the PBS generic checkpoint facility for checkpoint and restart. Write checkpoint and restart scripts that use BLCR commands.

10.3.4.4.1 Cray Checkpointing Requirements

- CPR jobs must be linked with the Cray MPT 3.0.1 or later libraries, which have integrated BLCR support. The application must be an MPT application (either MPI or shmem) and linked with the BLCR libcr. Most users will simply need to load the BLCR module via the following:

  module load blcr

  and compile their application in the usual way for MPI or shmem applications.

- The Cray XT system must have the BLCR kernel module installed and loaded.

- For the Lustre file system, ensure the directory to which the context files will be written has a stripe count of 1. For example, if the checkpoint path is /scratch/BLCR_checkpoint_dir, do the following:

  # cd /scratch
  # lfs setstripe -c 1 BLCR_checkpoint_dir

- Ensure that the context file's destination is in the shared filesystem. The context file is written in the directory where the cr_checkpoint command is performed.

- Cray CPR must be enabled on the login nodes. As root, use xtopview
from the boot node, and do the following:

```bash
boot# xtopview -x /etc/node_classes -c login
login> chkconfig blcr on
```

- Environment variables must be set correctly on the login node. The file `PBS_HOME/pbs_environment`, which is available to user job scripts, must contain the following:

  ```
  LD_PRELOAD=/usr/lib64/libcr_run.so.0
  LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/lib:/usr/lib64
  ```

- Make sure that the ALPS checkpoint-restart rpm is installed on the root that is being used by those running jobs, or set the `CRAY_ROOTFS` environment variable to point to the root where the checkpoint-restart rpm is installed:

  On a CNL system the rpm might be called "cray-alpscpr-cn1".

  If users are using the DSL root and the *alpscpr* rpm is installed in the DSL root, checkpoint-restart works without having to set the `CRAY_ROOTFS` environment variable.

  When the *alpscpr* rpm is installed in a different root from the one being used by users, set the `CRAY_ROOTFS` environment variable to point to the root where it is installed. For example, if users are using the DSL root and the *alpscpr* rpm is installed on the INITRAMFS root, set it as shown:

  ```
  CRAY_ROOTFS=INITRAMFS
  ```

### 10.3.4.4.2 Cray Checkpointing Caveats

- On Cray systems, a job will run on the original login node, but job processes are not guaranteed to run on the same compute nodes as before the checkpoint.

- Do not use periodic checkpointing based on CPU time (via `qsub -c c`) for MPT jobs on a Cray. While the MOM runs and tracks CPU usage on the login nodes, the application is run on one or more compute nodes, where CPU usage by the compute node cannot be tracked by the MOM.
Chapter 10  Making Your Site More Robust

10.3.5  Parameters and Attributes Affecting Checkpointing

10.3.5.1  MOM Configuration Parameters Affecting Checkpointing

$\text{action checkpoint } <\text{timeout}> !<\text{script-path}> <\text{args}>
Checkpoints the job, allowing the job to continue running.

$\text{action checkpoint_abort } <\text{timeout}> !<\text{script-path}> <\text{args}>
Checkpoints, kills, and requeues the job.

$\text{action restart } <\text{timeout}> !<\text{script-path}> <\text{args}>
Restarts checkpointed job.

The $<\text{timeout}>$ is the time allowed for checkpoint or restart script to run.

Not supported in the HPCBP MOM.

$\text{checkpoint_path } <\text{path}>
MOM passes this parameter to the checkpoint and restart scripts. This path can be absolute or relative to PBS_HOME/mom_priv. Overrides default. Overridden by path specified in the pbs_mom -C option and by PBS_CHECKPOINT_PATH environment variable. Not supported in the HPCBP MOM.

$\text{restart_background } <\text{True|False}>
Specifies whether MOM runs the restart script in the background (MOM doesn’t wait) or foreground (MOM waits). When set to True, MOM runs the restart script in the background.

Automatically set by MOM; Controlled by value of $\text{restart_transmogrify}$. When $\text{restart_transmogrify}$ is True, $\text{restart_background}$ is set to False. When $\text{restart_transmogrify}$ is False, $\text{restart_background}$ is set to True.

Format: Boolean
Default: False
$restart_transmogrify <True|False>
Specifies which PID MOM tracks for a job that has been checkpointed and restarted.

When this parameter is set to True, MOM tracks the PID of the restart script. When this parameter is set to False, MOM tracks the PID of the original job.

The value of $restart_transmogrify controls the value of $restart_background.

Not supported in the HPCBP MOM.
Format: Boolean
Default: False

10.3.5.2 Options to pbs_mom Affecting Checkpointing

-C checkpoint_directory
Specifies the path to the directory where MOM creates job-specific subdirectories used to hold each job’s restart files. MOM passes this path to checkpoint and restart scripts. Overrides other checkpoint path specification methods.
Any directory specified with the -C option must be owned, readable, writable, and executable by root only (rwx,---,---, or 0700), to protect the security of the restart files. See the -d option to pbs_mom.

Format: String
Default: PBS_HOME/checkpoint

10.3.5.3 Job Attribute Affecting Checkpointing

Checkpoint
Determines when the job will be checkpointed. Can take on one of the following values:
c
Checkpoint at intervals, measured in CPU time, set on the job’s execution queue. If there is no interval set on the queue, the job is not checkpointed.
c=<minutes of CPU time>
Checkpoint at intervals of the specified number of minutes of job CPU time. This value must be greater than zero. If the interval specified is less than that set on the job’s execution queue, the queue's interval is used.
Format: Integer

w
Checkpoint at intervals, measured in walltime, set on the job’s execution queue. If there is no interval set at the queue, the job is not checkpointed.

w=<minutes of walltime>
Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less that that set on the execution queue in which the job resides, the queue's interval is used.
Format: Integer

n
No checkpointing.

s
Checkpoint only when the server is shut down.

u
Unset. Defaults to behavior when interval argument is set to s.
Default: u.
Format: String

10.3.5.4 Queue Attribute Affecting Checkpointing

cHECKPOINT_MIN
Specifies the minimum number of minutes of CPU time or walltime allowed between checkpoints of a job. If a user specifies a time less than this value, this value is used instead. The value given in checkpoint_min is used for both CPU minutes and walltime minutes. See the Checkpoint job attribute.
10.3.5.5 Environment Variable Affecting Checkpointing

PBS_CHECKPOINT_PATH
MOM passes this path to the checkpoint and restart scripts. Overridden by -C option to pbs_mom; overrides $checkpoint_path MOM parameter and default. See section 10.3.6.5 “Specifying Checkpoint Path” on page 755.

10.3.5.6 The Epilogue

PBS will requeue a job which was snapshot checkpointed, if the epilogue returns the value 2. See section 10.3.7.3 “Requeueing via Epilogue” on page 757.

10.3.6 Checkpoint and Restart Scripts

The restart script is run by the same MOM that ran the checkpoint script. The checkpoint and restart scripts are run for each task of the job. When MOM executes a checkpoint or restart script, she forks a child process, which exec( )s the script. The restart script looks for the restart file in the job-specific subdirectory created by MOM, under the specified path. See section 10.3.6.5 “Specifying Checkpoint Path” on page 755.

10.3.6.1 Environment Variables for Scripts

PBS sets the following variables in the checkpoint and restart scripts’ environments before running the scripts:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Value of Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GID</td>
<td>Job owner’s group ID</td>
</tr>
</tbody>
</table>
### Table 10-7: Checkpoint/Restart Script Environment Variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Value of Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td>Job owner’s PBS home directory</td>
</tr>
<tr>
<td>LOGNAME</td>
<td>Job owner’s login name</td>
</tr>
<tr>
<td>PBS_JOBCOOKIE</td>
<td>128-bit random number used as token to authenticate job processes</td>
</tr>
<tr>
<td>PBS_JOBID</td>
<td>The job’s ID</td>
</tr>
<tr>
<td>PBS_JOBNAME</td>
<td>The job’s name</td>
</tr>
<tr>
<td>PBS_MOMPORT</td>
<td>Port number on which MOM listens for resource manager requests</td>
</tr>
<tr>
<td>PBS_NODEFILE</td>
<td>Path and filename of this job’s nodes file</td>
</tr>
<tr>
<td>PBS_NODENUM</td>
<td>Index into the nodes file; index of this vnode; starts at 0</td>
</tr>
<tr>
<td>PBS_QUEUE</td>
<td>Name of the job’s execution queue</td>
</tr>
<tr>
<td>PBS_SID</td>
<td>Session ID of task for which script is being called</td>
</tr>
<tr>
<td>PBS_TASKNUM</td>
<td>Index into task table for this job; index of task for which script is being called</td>
</tr>
<tr>
<td>SHELL</td>
<td>Job owner’s login shell</td>
</tr>
<tr>
<td>UID</td>
<td>Job owner’s execution ID</td>
</tr>
<tr>
<td>USER</td>
<td>Job owner’s username</td>
</tr>
<tr>
<td>USERPROFILE</td>
<td>(Windows only) Job owner’s Windows home directory</td>
</tr>
<tr>
<td>USERNAME</td>
<td>(Windows only) Job owner’s Windows username</td>
</tr>
</tbody>
</table>
10.3.6.2 The Checkpoint Script

The checkpoint script writes a restart file that is specific to the job being checkpointed. The checkpoint script must save all of the information needed to restart the job. This is the information that will be used by the restart script to restart the job. PBS runs the script for each running job task, on each vnode where a task is running.

10.3.6.2.1 Requirements for Checkpoint Script

- The first line of the script must specify the shell to be used, for example:
  
  ```bash
  #!/bin/sh
  ```

- The script should return the following error codes:
  - **Zero** for success
  - **Non-zero** for failure

- The script should block until the checkpoint process is finished.

- The restart file and its directory should be owned by root, and writable by root only, with permission 0755.

- Under UNIX/Linux, the checkpoint script should be owned by root, and writable by root only, with permission 0755.

- Under Windows, the checkpoint script must have at least Full Control permission for the local Administrators group.

- The checkpoint script must write the restart file(s) in the location expected by the restart script. You don’t have to use the `%path` parameter passed by MOM.

- If the script is for checkpoint-abort, the script must ensure that all processes are killed, whether directly or indirectly, for example by touching a file. All job processes must exit.

10.3.6.3 The Restart Script

The restart script does only one of the following:

- Reinstates the job’s original PID, so that MOM tracks the original PID
- Becomes the new top process of the job, so that MOM tracks the PID of the script
If $restart_transmogrify$ is set to $True$, the restart script becomes the new top task for the job, and MOM begins tracking its process ID, where she was tracking the job’s original process ID. If $restart_transmogrify$ is set to $False$, MOM continues to track the original job PID.

The restart script can use $pbs_attach$ to attach job processes to the original job PID, or to the script’s PID. See “$pbs_attach$” on page 52 of the PBS Professional Reference Guide.

10.3.6.3.1 Caveats for Restart Script

The $pbs_attach$ command is not supported under Windows.

10.3.6.3.2 Requirements for Restart Script

The restart script must handle everything required to restart the job from the information saved by the checkpoint script.

The restart script must block until the restart process is finished.

Under UNIX/Linux, the restart script should be owned by root, and writable by root only, with permission 0755.

Under Windows, the restart script must have at least Full Control permission for the local Administrators group.

10.3.6.3.3 Return Values for Restart Script

The restart script must inform PBS of success or failure. It must return one of the following:

- **Zero** for success
- **Non-zero** for failure

10.3.6.4 Scripts for Application Checkpointing

If a user’s application can be checkpointed periodically according to wall-time or CPU time, you can use the PBS snapshot checkpoint facility to trigger snapshot checkpointing by the application.

If a user’s application can be checkpointed, you can use the PBS $checkpoint_abort$ facility before shutting down PBS to avoid losing intermediate results.
Some applications produce a restart file when they are sent a specific signal, or when a specific file is affected. A checkpoint script for this purpose sends the application the correct signal, or makes the correct change to the file.

Some applications only need the checkpoint and restart scripts to be run once each. In this case, the checkpoint and restart scripts should handle this requirement.

### 10.3.6.5 Specifying Checkpoint Path

When a job is checkpointed, information about the job is saved into a file. The location for this file can be any directory accessible to MOM.

The path to the checkpoint directory is composed of two parts. The first part is common to all jobs; this part can specified. The second part is a job-specific subdirectory, created by MOM for each job, under the common directory. The job’s restart file is written in this job-specific subdirectory.

The default common directory, `PBS_HOME/checkpoint`, is provided for convenience.

You can specify the filename and the path for the common directory using any of the following methods. If the first is specified, PBS uses it. If not, and the second is specified, PBS uses the second, and so on.

- The `-C` path option to the `pbs_mom` command
- The `PBS_CHECKPOINT_PATH` environment variable
- The `$checkpoint_path` MOM configuration option in `PBS_HOME/mom_priv/config`
- The default value of `PBS_HOME/checkpoint`

The job-specific subdirectory is named with the following format:

```
<job ID>.CK
```

For example, if you specify `/usr/bin/checkpoint` for the common directory, and the job’s ID is `1234.host1`, the job’s restart file is written under `/usr/bin/checkpoint/1234.host1.CK`.

The restart file and its directory should be owned by root, and writable by root only.
10.3.6.5.1 Checkpoint Path Caveats

If the checkpoint file is in PBS_HOME/checkpoint/<job ID>.CK/, and MOM thinks that a checkpoint failed (the checkpoint script returned non-zero), she will remove the checkpoint file. If the checkpoint script puts the checkpoint file in another location, MOM does not remove the checkpoint file.

10.3.7 Using Checkpointing

10.3.7.1 Periodic Job Checkpointing

If a job’s Checkpoint attribute is set to c, c=<minutes>, w, or w=<minutes>, the job is periodically checkpointed. The checkpoint interval is specified either in the job’s Checkpoint attribute or in the queue’s checkpoint_min attribute. See “Job Attributes” on page 430 of the PBS Professional Reference Guide. The job’s Checkpoint attribute is set using the -c <interval> option to the qsub command. See “qsub” on page 245 of the PBS Professional Reference Guide.

When this attribute is set, at every <interval> the job is checkpointed and a restart file is written, but the job keeps running.

10.3.7.2 Checkpointing During Shutdown

The effect on jobs of shutting down PBS depends on the method used to shut PBS down. When a job is checkpointed during shutdown, MOM runs the checkpoint_abort script, and PBS kills and requeues the job. PBS does not hold the job, so the job is eligible to be run again as soon as the server starts up.

If you use the qterm command, there are three different suboptions to the -t option to control whether jobs are checkpointed, requeued, or allowed to continue running.

If you use the PBS start/stop script, the script affects only the host where the script is run. Any jobs running completely or partly on that host are killed and requeued, but not checkpointed. Any jobs not running on that host are left running.
The effect of each shutdown method is described here:

### Table 10-8: Effect of Shutdown on Jobs

<table>
<thead>
<tr>
<th>Shutdown Method</th>
<th>Effect on Checkpointable Jobs</th>
<th>Effect on Non-checkpointable Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>qterm -t quick</td>
<td>Continue to run</td>
<td>Continue to run</td>
</tr>
<tr>
<td>qterm -t delay</td>
<td>Checkpointed, killed, requeued, held</td>
<td>Requeued if rerunnable; continue to run if not rerunnable</td>
</tr>
<tr>
<td>qterm -t immediate</td>
<td>Checkpointed, killed, requeued, held</td>
<td>Requeued if rerunnable; deleted if not rerunnable</td>
</tr>
<tr>
<td>init.d/ pbs stop</td>
<td>Any jobs running completely or partly on host where stop script is run are killed and requeued</td>
<td>Any jobs running completely or partly on host where stop script is run are killed and requeued</td>
</tr>
</tbody>
</table>

Any running subjobs of a job array are always killed and requeued when the server is shut down.

#### 10.3.7.3 Requeueing via Epilogue

You can configure MOM to requeue a failed job that was snapshot checkpointed during its execution. For example, if a job terminates, but had a hardware failure during execution, PBS can requeue the job, and MOM will run the start script, which can restart the job from its restart file.

When the job is requeued via the epilogue mechanism, it is in the Q state.
10.3.7.3.1 Requirements for Requeueing via Epilogue

The following requirements must be met in order for a job to be requeued via the epilogue mechanism:

- The epilogue must return a value of 2
- The job must have been checkpointed under the control of PBS
- The MOM must be configured with a restart script in the $action restart MOM configuration parameter
- The MOM must be configured to snapshot checkpoint jobs in the $action checkpoint MOM configuration parameter
- The jobs must request checkpointing via their Checkpoint attribute. See section 10.3.7.1 “Periodic Job Checkpointing” on page 756
- The epilogue script in PBS_HOME/mom_priv/epilogue must return the following:
  - Zero (0) for successful termination (requeue is not required)
  - Two (2) for failure (requeue is required)

10.3.7.4 Checkpointed Jobs and Server Restart

When the server is restarted using the pbs_server -t warm command or the init.d/pbs start script, jobs that were checkpointed and aborted upon shutdown are waiting in their queues, and are eligible to be run according to the scheduler’s algorithm.

When the server is restarted using the pbs_server -t hot command, jobs that were checkpointed and aborted upon shutdown are immediately rerun, before the scheduler selects which jobs to run.

10.3.7.5 Preemption Using Checkpoint

When a job is preempted via checkpointing, MOM runs the checkpoint_abort script, and PBS kills and requeues the job. When the scheduler elects to run the job again, the scheduler runs the restart script to restart the job from where it was checkpointed. For a description of using preemption, see section 4.8.33 “Using Preemption” on page 258.
10.3.7.6 Holding a Job

When anyone uses the qhold command to hold a checkpointable job, MOM runs the checkpoint_abort script, which kills all job processes, and PBS requeues, and holds the job.

A job with a hold on it must have the hold released via the qrls command in order to be eligible to run.

The following is the sequence of events when a job is held:

- MOM runs the checkpoint_abort script
- The job’s execution is halted
- The resources assigned to the job are released
- The job is placed in the Held state in the execution queue
- The job’s Hold_Types attribute is set appropriately

A held job is waiting in its queue. The following is the sequence of events when a held job is restarted:

- The hold is released by means of the qrls command; the job is now in the Queued state
- The job continues to wait in its queue until the scheduler schedules it for execution
- The scheduler selects the job for execution
- The job is sent to its original MOM for execution
- The MOM runs the restart script

10.3.7.6.1 Restrictions on Holding a Job

A job in the process of provisioning cannot be held.

The qhold command can be used on job arrays, but not on subjobs or ranges of subjobs.

If the job cannot be checkpointed and aborted, qhold simply sets the job's Hold_Types attribute. The job continues to execute.

The checkpoint-abort script must terminate all job processes, or the qhold command will appear to hang.
10.3.7.7 Periodic Application Checkpoint

The snapshot checkpoint script can trigger checkpoint by a job’s application, if the application is written to support checkpointing itself. Note that an application may be designed to be checkpointed at specific stages in its execution, rather than at specific points in time. If an application can be usefully checkpointed at specific points in time, then snapshot checkpointing may be useful. See section 10.3.7.1 “Periodic Job Checkpointing” on page 756.

10.3.7.8 Manual Application Checkpoint

When an application is checkpointed manually, the user triggers checkpointing by the application by sending the application a specific signal, or by creating a file.

10.3.8 Advice and Caveats

Multi-vnode jobs may cause network sockets to be opened between submission and execution hosts, and open sockets may cause a checkpointing script or tool to fail. The following use sockets:

- An interactive job, i.e. a job submitted using `qsub -I`, opens unprivileged sockets. `qsub` binds a socket to a port, then waits to accept a connection from MOM on that socket. Data from standard in is written to the socket and data from the socket is written to standard out.
- The `pbs_demux` process collects stdio streams from all tasks
- The `pbsdsh` program spawns tasks. The `-o` option to this command prevents it from waiting for spawned tasks to finish, so that no socket is left open to the MOM to receive task manager events. When the `-o` option is used, the shell must use some other method to wait for the tasks to finish. See “pbsdsh” on page 118 of the PBS Professional Reference Guide.
10.3.9 Accounting

If a job is checkpointed and requeued, the exit status passed to the epilogue and recorded in the accounting record is the following:

-12, meaning that the job was checkpointed and aborted

A checkpoint (“C”) record is written in the accounting log when the job is checkpointed and requeued, as when the `qhold` command is used, or the job is checkpointed and aborted.

10.4 Preventing Communication and Timing Problems

10.4.1 Introduction

PBS communicates with remote execution hosts in order to track their availability and manage the jobs running on them. PBS is dependent upon your network for this communication. If there are network outages, or if the execution node becomes too busy for MOM to be able to respond to the server's queries, PBS will not be able to function properly. You can configure PBS to be better able to withstand these types of communication issues.

The following attributes and parameters control how PBS handles communication timing:

Table 10-9: Attributes and Parameters For Communication Timing

<table>
<thead>
<tr>
<th>Attribute or Parameter</th>
<th>Description</th>
<th>Cross Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Attributes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>job_requeue_timeout</td>
<td>Controls how long the process of requeueing a job is allowed to take</td>
<td>See section 10.4.3 “Setting Job Requeue Timeout” on page 767</td>
</tr>
</tbody>
</table>
### Table 10-9: Attributes and Parameters For Communication Timing

<table>
<thead>
<tr>
<th>Attribute or Parameter</th>
<th>Description</th>
<th>Cross Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_fail_requeue</td>
<td>Controls how long the server waits before requeueing or deleting a job when it loses contact with the MOM on the job’s primary execution host</td>
<td>See section 10.4.2 “Node Fail Requeue: Jobs on Failed Vnodes” on page 763</td>
</tr>
<tr>
<td>rpp_retry</td>
<td>Maximum number of times the RPP library will try to re-send a packet before giving up.</td>
<td>See section 10.4.4 “Inter-daemon Communication” on page 767</td>
</tr>
<tr>
<td>rpp_highwater</td>
<td>Maximum number of RPP packets allowed in transit at any one time.</td>
<td>See section 10.4.4 “Inter-daemon Communication” on page 767</td>
</tr>
</tbody>
</table>

### MOM Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Cross Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$max_load</td>
<td>Vnode is considered to be <em>busy</em> if it is above this load.</td>
<td>See section 10.4.5 “Managing Load Levels on Vnodes” on page 768</td>
</tr>
<tr>
<td>$ideal_load</td>
<td>Vnode is considered to be not <em>busy</em> if it is below this load.</td>
<td>See section 10.4.5 “Managing Load Levels on Vnodes” on page 768</td>
</tr>
<tr>
<td>$prologalarm</td>
<td>Maximum number of seconds the prologue and epilogue may run before timing out</td>
<td>See section 10.4.6 “Prologue &amp; Epilogue Running Time” on page 772</td>
</tr>
</tbody>
</table>

### Queue Attributes
10.4.2 Node Fail Requeue: Jobs on Failed Vnodes

The `node_fail_requeue` server attribute controls how long the server waits before requeuing or deleting a job when it loses contact with the MOM on the job’s primary execution host.

10.4.2.1 How Node Fail Requeue Works

You can specify how long the server waits after it loses contact with Mother Superior before deleting or requeuing her jobs. This behavior is controlled by the server’s `node_fail_requeue` attribute.

This attribute’s value is the delay between the time the server determines that the primary execution host cannot be contacted and the time it requeues the job, and does not include the time it takes to determine that the host is out of contact.

If this attribute is set to a value other than zero, and the server loses contact with an execution host, all jobs for which this is the primary execution host are requeued or deleted at the same time.

If `node_fail_requeue` is unset, and the host where Mother Superior is running fails, the server assumes that the job is still running until one of the following happens:

- Mother Superior comes back up and tells the server to requeue the job
- The job is manually rerun

---

Table 10-9: Attributes and Parameters For Communication Timing

<table>
<thead>
<tr>
<th>Attribute or Parameter</th>
<th>Description</th>
<th>Cross Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>route_retry_time</td>
<td>Interval between retries at routing a job</td>
<td>See section 10.4.7 “Time Between Routing Retries” on page 773</td>
</tr>
</tbody>
</table>
10.4.2.1.1 Allowable Values

*Greater than zero*

The server waits for the specified number of seconds after losing contact with a Mother Superior, then attempts to contact the primary execution host, and if it cannot, requeues any jobs that can be rerun and deletes any jobs that cannot be rerun.

*Unset or zero*

Jobs are left in the *Running* state whether or not the server has contact with their Mother Superior.

*Less than zero*

The attribute is treated as if it were set to 1, and jobs are deleted or requeued after the server has been out of contact with Mother Superior for 1 second.

10.4.2.1.2 Default Value

The default value for this attribute is 310, meaning that when the server loses contact with an execution host, it waits for 310 seconds after losing contact with Mother Superior before requeueing or deleting jobs.

10.4.2.2 Effect Of Requeueing On Jobs

When a job is thus requeued, it retains its original place in its execution queue with its former priority. The job is usually the next job to be considered during scheduling, unless the relative priorities of the jobs in the queue have changed. This can happen when the job sorting formula assigns higher priority to another job, another higher-priority job is submitted after the requeued job started, this job’s owner has gone over their fair-share limit, etc.

Any resources that were being used by a job are freed when the job is requeued.

10.4.2.3 The node_fail_requeue Server Attribute

The node_fail_requeue attribute can take these values:
Unset or zero
Jobs are not requeued; they are left in the Running state until the execution vnode is recovered.

Greater than zero
Jobs are requeued if they are rerunnable when the node has been down for the specified number of seconds. Jobs that cannot be requeued are deleted.

Less than zero
This is the same as setting the value to 1.

Format: Integer
Default: 310 seconds

10.4.2.4 Where node_fail_requeue Applies

The server’s node_fail_requeue attribute applies only in the case where the server loses contact with Mother Superior.

When Mother Superior loses contact with a sister MOM, the job is immediately deleted or requeued.

10.4.2.5 Jobs Eligible to be Requeued

Jobs are eligible to be requeued if they meet either of the following criteria:

• The job’s Rerunnable attribute is set to y
• The job did not begin execution, for example:
  - a multi-host job did not start on one or more vnodes
  - provisioning failed for the job

Jobs are ineligible to be requeued if their Rerunnable attribute is set to n and they have started execution.

10.4.2.6 Using node_fail_requeue

The number of seconds selected should be long enough to exceed any transient non-vnode failures, but short enough to requeue the job in a timely fashion. Transient non-vnode failures can prevent MOM from reporting back to the server before the server marks the vnode **down**. These include:

- Network outages
- Vnode is too busy to respond, perhaps due to heavy swapping

Using this feature requires that you take the following into account:

- If the host where Mother Superior is running fails, and `node_fail_requeue` is unset, the server assumes that the job is still running until one of the following happens:

  - Mother Superior comes back up and tells the server to requeue the job
  - The job is manually rerun

If your site has hosts that fail and are not monitored, failed jobs may go unnoticed for a long time.

- If your network has temporary failures, and `node_fail_requeue` is set to a duration shorter than the outage, jobs will be unnecessarily requeued. This can be especially annoying when the job has been running for days.

10.4.2.7 Advice and Caveats

If your site experiences frequent network failures or your execution hosts are often too busy to respond to the server, it is recommended that you either set `node_fail_requeue` to a value greater than the time MOM is unavailable, or set it to **zero**. This way jobs won’t be requeued just because the network had a temporary outage or the vnode was too busy. Choose a value greater than both the longest likely network outage time and the time MOM is unavailable. For example, one site has set the value to 10 minutes, and another has set it to 15 minutes (900 seconds) to avoid problems due to swapping.
### 10.4.3 Setting Job Requeue Timeout

When jobs are preempted via requeueing, the requeue can fail if the job being preempted takes longer than the allowed timeout. The time for requeueing includes post-processing such as staging files out, deleting files, and changing the job’s state from \( R \) to \( Q \). See section 4.8.33 “Using Preemption” on page 258. The time allowed for a job to be requeued is controlled by the `job_requeue_timeout` server attribute.

You can use `qmgr` to set the `job_requeue_timeout` server attribute to a value that works for the jobs at your site. This attribute is of type `Duration`, with a minimum allowed value of 1 second and a maximum allowed value of 3 hours. The default timeout is 45 seconds. See “Server Attributes” on page 365 of the PBS Professional Reference Guide.

### 10.4.4 Inter-daemon Communication

The Reliable Packet Protocol (RPP) is a library of functions used by PBS for reliable, flow-controlled, two-way data transmission. RPP is built on top of UDP. For more about RPP, see “RPP Library” on page 89 of the PBS Professional Programmer’s Guide.

#### 10.4.4.1 Factors Affecting RPP Performance

Usually, RPP performance is affected at sites with many vnodes. In this case, the network is overwhelmed with all the UDP traffic, and packets are dropped.

#### 10.4.4.2 When to Adjust RPP Settings

It can help to adjust `rpp_retry` when there are too many retries or failures showing up in log messages from any daemon. The log messages have this form:

```
/var/spool/PBS/server_logs/20100107:01/07/2010
21:41:39;0080;Server@surly;Node;rpp;total
(pkts=154,retries=120,fails=12) last 3601 secs
(pkts=154,retries=120,fails=12)
```
10.4.4.3 Finding Useful Values for RPP Settings

It is recommended that you start by setting `rpp_retry` to 30, and raise the value if that is insufficient. It is not recommended to set the value of `rpp_retry` to more than 100.

10.4.4.4 Server RPP Attributes

rpp_retry
The maximum number of times the RPP network library will try to send a UDP packet again before giving up. The number of retries is added to the original try, so if `rpp_retry` is set to 2, the total number of tries will be 3.

Format: *Integer*

Valid values: *Greater than or equal to zero*

Default: 10

Python type: int

rpp_highwater
The maximum number of RPP packets that can be in transit at any time.

Format: *Integer*

Valid values: *Greater than or equal to one*

Default: 64

Python type: int

10.4.5 Managing Load Levels on Vnodes

An overloaded execution host may end up too busy for MOM to respond to the server’s queries, and causing the server to mark the MOM as *down*.

PBS can track the state of each execution host, running new jobs on the host according to whether the host is marked *busy* or not.

This behavior is somewhat different from load balancing, described in section 4.8.27 “Using Load Balancing” on page 216. In load balancing, the scheduler estimates how much load a job would produce, and will not place
a job where doing so would put the load above the limit. When managing load levels on vnodes as described here, the scheduler uses the state of the vnode to determine whether to place a job on that vnode.

The state of the vnode is set by MOM, according to its load. You can set two load levels using the $max_load and $ideal_load MOM configuration parameters. When the load goes above $max_load, the vnode is marked as busy. When the load drops below $ideal_load, the vnode is marked free.

PBS does not run new jobs on vnodes under the following conditions:

- Vnodes that are marked busy
- Vnodes whose resources, such as ncpus, are already fully allocated
- Vnodes where the load is above $max_load, when load balancing is turned on. See section 4.8.27 “Using Load Balancing” on page 216.
- Vnodes where running the job would cause the load to go above $max_load, when load balancing is turned on. See section 4.8.27 “Using Load Balancing” on page 216.

The load used by MOM is the following:

- On UNIX/Linux, it is the raw one-minute averaged “loadave” returned by the operating system
- On Windows, it is based on the processor queue length

The $max_load and $ideal_load MOM configuration parameters are also used for cycle harvesting (see section 4.8.9.5 “Cycle Harvesting Based on Load Average” on page 164) and load balancing (see section 4.8.27 “Using Load Balancing” on page 216.)

MOM checks the load average on her host every 10 seconds.

When a vnode’s state changes, for example from free to busy, MOM informs the server.

### 10.4.5.1 Techniques for Managing Load

Whether or not you set $max_load, PBS will not run jobs requesting a total of more than the available number of CPUs, which is set in resources_available.ncpus. So for example if
resources_available.ncpus is set to 4, and a job running on the vnode has requested 2 CPUs, PBS will not run jobs requesting a total of more than 2 CPUs.

10.4.5.1.1 Types of Workload

How you manage load depends on your workload. Some jobs do not lend themselves to sharing CPUs, but some jobs can share CPUs without being hindered. Most MPI jobs would be hindered if some processes had to wait because others were slowed by sharing a CPU. If you need a job to have reproducible timing, it cannot share a CPU. Certain single-vnode jobs that alternate between CPU usage and I/O can share a CPU without being slowed significantly, thereby increasing throughput.

10.4.5.1.2 How Not To Share CPUs

For vnodes primarily running jobs that would be slowed or invalidated by sharing a CPU, have PBS assign jobs according to the number of available CPUs, so that there is no sharing of CPUs. Set resources_available.ncpus to the number of available CPUs. Do not set $max_load or $ideal_load.

10.4.5.1.3 How To Share CPUs

For vnodes running only jobs that can share CPUs, you can have PBS manage jobs according to the load on the vnodes, not the number of CPUs. This is called oversubscribing the CPUs. Set resources_available.ncpus to a value greater than the actual number of CPUs, such as two or three times the actual number. Set $max_load to a reasonable value so that PBS will run new jobs until $max_load is reached. Set $ideal_load to the minimum load that you want on the vnode.

10.4.5.1.4 Suspending Jobs on Overloaded Vnodes

You can specify that MOM should suspend jobs when the load goes above $max_load, by adding the suspend argument to the $max_load parameter. See section “$max_load <load> [suspend]” on page 772. In this case, MOM suspends all jobs on the vnode until the load drops below $ideal_load, then resumes them. This option is useful only when the source of the load includes work other than PBS jobs. This option is not recommended when the load is due solely to PBS jobs, because it can lead
to the vnode cycling back and forth between being overloaded, being marked busy, suspending all jobs, being marked free, then starting all jobs, being overloaded, and so on.

10.4.5.2  Caveats and Recommendations

- Load level tracking is not supported in the HPCBP MOM.
- It is recommended that the value for $ideal_load be lower than the value for $max_load. The value for $ideal_load should be low enough that new jobs are not run before existing jobs are done using the vnode’s spare load.
- If you set only one of $max_load and $ideal_load, for example you set $max_load, but not $ideal_load, PBS sets the other to the same value.
- Do not allow reservations on hosts where $max_load and $ideal_load are configured. Set the resv_enable vnode attribute on these hosts to False.
- If you are using cycle harvesting via load balancing, be careful with the settings for $ideal_load and $max_load. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MOM to report being busy, and that PBS does not start any new jobs while the user is working. See section 4.8.9.5 “Cycle Harvesting Based on Load Average” on page 164.

10.4.5.2.1  Allowing Non-job Processes on Execution Host

If you wish to run non-PBS processes on a host, you can prevent PBS from using more than you want on that host. Set the $ideal_load and $max_load MOM configuration parameters to values that are low enough to allow other processes to use some of the host.

10.4.5.3  Load Configuration Parameters

$ideal_load <load>  
MOM parameter. Defines the load below which the vnode is not considered to be busy. Used with the $max_load parameter. Not supported in the HPCBP MOM.
Example:

$ideal_load 1.8

Format: *Float*

No default

$max_load <load> [suspend]

MOM parameter. Defines the load above which the vnode is considered to be busy. Used with the $ideal_load parameter.

If the optional suspend argument is specified, PBS suspends jobs running on the vnode when the load average exceeds $max_load, regardless of the source of the load (PBS and/or logged-in users).

Not supported in the HPCBP MOM.

Example:

$max_load 3.5

Format: *Float*

Default: number of CPUs

### 10.4.6 Prologue & Epilogue Running Time

Each time the scheduler runs a job, it waits for the prologue to finish before it runs another job. In order to prevent a hung prologue from halting job execution, prologues and epilogues are only allowed to run for a specified amount of time before PBS kills them. The running time is specified in the $prologalarm MOM configuration parameter. The default value for this parameter is *30 seconds*.

### 10.4.6.1 Prologue Timeout Configuration Parameter

$prologalarm <timeout>

Defines the maximum number of seconds the prologue and epilogue may run before timing out.

Example:

$prologalarm 30
10.4.7 Time Between Routing Retries

If the network is flaky, PBS may not be able to route a job from a routing queue to the destination queue. If all destination queues for a routing queue are at capacity, a job in a routing queue remains where it is. The time between routing retries is controlled by the `route_retry_time` queue attribute.

If the network experiences long outages, you may wish to set the time between retries to a sufficiently long time that PBS is not wasting cycles attempting to route jobs.

If jobs in a routing queue are not being routed because the destination queues are full, and most jobs are long-running jobs, you may wish to set the time between retries so that attempts are infrequent. It is recommended that the time between retries be no longer than the longest time acceptable to have an open slot in an execution queue.

10.4.7.1 Routing Retry Attribute

```
route_retry_time
Time delay between routing retries. Typically used when the network between servers is down. Used only with routing queues.
```

Format: `Integer seconds`
Default: `30 seconds`
Python type: `pbs.duration`
10.5 Reservation Fault Tolerance

If the vnodes associated with an advance reservation or the soonest occurrence of a standing reservation become unavailable, PBS marks the advance or standing reservation as *degraded*. PBS attempts to reconfirm degraded reservations.

States of available vnodes:

- *free*
- *busy*
- *job-exclusive*
- *job-sharing*
- *job-busy*

States of unavailable vnodes:

- *down*
- *offline*
- *stale*
- *state-unknown, down*

The reservation’s state becomes *RESV_DEGRADED*, abbreviated DG, and its substate becomes *RESV_DEGRADED*.

If vnodes associated with an occurrence later than the soonest occurrence of a standing reservation become unavailable, the reservation stays in state *RESV_CONFIRMED*, but its substate becomes *RESV_DEGRADED*.

During the time that a degraded advance reservation or the soonest occurrence of a degraded standing reservation is running, its state is *RESV_RUNNING*, and its substate is *RESV_DEGRADED*. 
The following table shows states and substates for degraded reservations:

**Table 10-10: Degraded Reservation States and Substates**

<table>
<thead>
<tr>
<th>Advance Reservation</th>
<th>Reservation Time in Future</th>
<th>Reservation Time Is Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>RESV_DEGRADED</td>
<td>RESV_RUNNING</td>
</tr>
<tr>
<td>Substate</td>
<td>RESV_DEGRADED</td>
<td>RESV_DEGRADED</td>
</tr>
<tr>
<td>Soonest Occurrence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>RESV_DEGRADED</td>
<td>RESV_RUNNING</td>
</tr>
<tr>
<td>Substate</td>
<td>RESV_DEGRADED</td>
<td>RESV_DEGRADED</td>
</tr>
<tr>
<td>Non-soonest Occurrence Only</td>
<td>State</td>
<td>RESV_CONFIRMED</td>
</tr>
<tr>
<td>Substate</td>
<td>RESV_DEGRADED</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The following table lists the numeric representations of reservation states and substates:

**Table 10-11: Numeric Reservation States and Substates**

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Substate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RESV_NONE</td>
</tr>
<tr>
<td>1</td>
<td>RESV_UNCONFIRMED</td>
</tr>
<tr>
<td>2</td>
<td>RESV_CONFIRMED</td>
</tr>
<tr>
<td>3</td>
<td>RESV_WAIT</td>
</tr>
<tr>
<td>4</td>
<td>RESV_TIME_TO_RUN</td>
</tr>
<tr>
<td>5</td>
<td>RESV_RUNNING</td>
</tr>
<tr>
<td>6</td>
<td>RESV_FINISHED</td>
</tr>
<tr>
<td>7</td>
<td>RESV_RESV_BEING_DELETED</td>
</tr>
<tr>
<td>8</td>
<td>RESV_DELETED</td>
</tr>
<tr>
<td>9</td>
<td>RESV_DELETING_JOBS</td>
</tr>
</tbody>
</table>
10.5.1 Reconfirming Reservations

PBS attempts to reconfirm degraded reservations. This means that PBS checks whether there are alternate available vnodes on which the advance reservation or the soonest occurrence can run. PBS starts periodically trying to reconfirm the reservation after the reservation becomes degraded, and stops trying before the reservation’s start time. For standing reservations, PBS tries to reconfirm subsequent occurrences if the soonest occurrence could not be reconfirmed.

PBS periodically tries to reconfirm a degraded reservation until either:

- The reservation is confirmed
- The cutoff time is reached

A degraded reservation has a read-only reservation attribute called 
reserve_retry, whose value is the next time at which the reservation is due to be reconfirmed.

10.5.1.1 Attributes Affecting Reservation Reconfirmation

**reserve_retry_cutoff**

Server attribute. The time period before the reservation start time during which PBS does not attempt to reconfirm a degraded reservation. When this value is changed, all degraded reservations use the new value.

Cannot be read by user or Operator.

Format: Integer (seconds)

Values: Must be greater than zero

Default: 7200 (2 hours)

Python attribute value type: int
reserve_retry_init
Server attribute. The amount of time after a reservation becomes degraded that PBS waits before attempting to reconfirm the reservation. When this value is changed, only reservations that become degraded after the change use the new value.
Cannot be read by user or Operator.
Format: Integer seconds
Values: Must be greater than zero
Default: 7200 (2 hours)
Python attribute value type: int

10.5.1.2 Configuring Reservation Reconfirmation

You can configure the amount of time between when a reservation becomes degraded and when PBS first tries to reconfirm it, using the server attribute reserve_retry_init. If you unset this attribute, it reverts to its default value. If you change this attribute, previously degraded reservations are unaffected; only reservations that become degraded after the change have the new attribute value applied to them.

PBS stops trying to reconfirm a reservation for a configurable amount of time before that reservation’s start time. The amount of time between when PBS stops trying to reconfirm a reservation and that reservation’s start time is specified in the server attribute reserve_retry_cutoff. If you unset this attribute, it reverts to its default value. If you change this attribute, all degraded reservations use the new value.

10.5.1.3 Reconfirming Standing Reservations

PBS attempts to reconfirm each soonest occurrence of a standing reservation. This means that if the first occurrence of a standing reservation becomes degraded, and PBS is unable to reconfirm it, then after its reservation period has passed, PBS will try to reconfirm the second occurrence of the standing reservation, and so on. PBS will try to reconfirm each occurrence of the standing reservation until either the reservation is confirmed or there are no more occurrences.
10.5.2 Allocating New Vnodes

Once new vnodes are allocated for a reservation:

- The reservation has been confirmed
- The state and substate of the reservation are RESV_CONFIRMED
- The reservation’s resv_nodes attribute lists the new vnodes

10.5.3 Restarting the Server

When the server is restarted, reservations are assumed confirmed until associated vnodes are recognized as unavailable. If any reservations become degraded after a server restart, PBS sets the time when the reservation becomes degraded to the time of the restart. If a vnode is set offline before the restart, it is considered unavailable after the restart, so all its associated reservations become degraded.

10.5.4 Logging Degraded Reservation Information

The server logs a message when a vnode associated with a confirmed reservation becomes unavailable, at event class 0x0100:

An attempt to reconfirm reservation will be made on
<ctime timestamp>

The server logs a message for each attempt at reconfirming a reservation, at event class 0x0100:

Next attempt to reconfirm reservation will be made on
<ctime timestamp>

The scheduler logs degraded reservations at event class 0x0200:

Reservation is in degraded mode, <x> out of <y> vnodes are unavailable; <list of vnodes down>
Chapter 11

Integrations

This chapter covers information on integrations, and is intended for the PBS administrator.

11.1 Support for MPI

PBS Professional is tightly integrated with several implementations of MPI. PBS can track resource usage for all of the tasks run under these MPIs. Some of the MPI integrations use `pbs_attach`, which means MOM polls for usage information like CPU time. The amount of usage data lost between polling cycles will depend on the length of the polling cycle. See “Configuring MOM’s Polling Cycle” on page 65.
11.1.1 Windows Caveats

The `pbs_attach` command is not supported for Windows.

On Windows, if you can use MPI outside of PBS, you can use it in a PBS job. However, PBS cannot use it to start job processes or track a job’s resource usage on Windows.

11.1.2 Interfacing MPICH with PBS Professional on UNIX

The existing `mpirun` command can be modified to check for the PBS environment and use the PBS-supplied host file. Do this by editing the `.../mpich/bin/mpirun.args` file and adding the following near line 40 (depending on the version being used):

```bash
if [ "$PBS_NODEFILE" != "" ]
then
    machineFile=$PBS_NODEFILE
fi

IMPORTANT:
Additional information regarding checkpointing of parallel jobs is given in section 10.3.8 “Advice and Caveats” on page 760.

11.1.2.1 MPICH on Linux

On Linux systems running MPICH with P4, the existing `mpirun` command is replaced with `pbs_mpirun` The `pbs_mpirun` command is a shell script which attaches a user’s MPI tasks to the PBS job.

11.1.2.2 The `pbs_mpirun` Command

The PBS command `pbs_mpirun` replaces the standard `mpirun` command in a PBS MPICH job using P4. The usage is the same as `mpirun` except for the `machinefile` option. The value for this option is generated by `pbs_mpirun`. All other options are passed directly to `mpirun`. 
The value used for the -machinefile option is a temporary file created from the PBS_NODEFILE in the format expected by mpirun. If the -machinefile option is specified on the command line, a warning will be output saying "Warning, -machinefile value replaced by PBS". The default value for the -np option is the number of entries in PBS_NODEFILE.

### 11.1.2.3 Transparency to the User

Users should be able to continue to run existing scripts. To be transparent to the user, pbs_mpirun should replace standard mpirun. To do this, the link for mpirun should be changed to point to pbs_mpirun:

- Install MPICH into /usr/local/mpich (or note path for mpirun)
  
  ```
  mv /usr/local/mpich/bin/mpirun /usr/local/mpich/bin/mpirun.std
  ```

- Create link called “mpirun” pointing to pbs_mpirun in /usr/local/mpich/bin/

- Edit pbs_mpirun to change "mpirun" call to "mpirun.std"

At this point, using "mpirun" will actually invoke pbs_mpirun.

When pbs_mpirun is run, it runs pbs_attach, which attaches the user’s MPI process to the job.

### 11.1.2.4 Environment Variables and PATHS

The PBS_RSHCOMMAND environment variable should not be set by the user. For pbs_mpirun to function correctly for users who require the use of ssh instead of rsh, several approaches are possible:

- Set P4_RSHCOMMAND in the login environment.

- Set P4_RSHCOMMAND externally to the login environment, then pass the value to PBS via qsub(1)’s -v or -V arguments:
  
  ```
  qsub -v P4_RSHCOMMAND=ssh ...
  ```

  or
  
  ```
  qsub -V ...
  ```
Chapter 11 Integrations

A PBS administrator may set P4_RSHCOMMAND in the pbs_environment file in PBS_HOME and advise users to not set P4_RSHCOMMAND in the login environment.

PATH on remote machines must contain PBS_EXEC/bin. Remote machines must all have pbs_attach in the PATH.

11.1.2.5 Notes

- When using SuSE Linux, use “ssh -n” in place of “ssh”.
- Usernames must be identical across vnodes.

11.1.3 Integration with LAM MPI

11.1.3.1 The pbs_lamboot Command

The PBS command pbs_lamboot replaces the standard lamboot command in a PBS LAM MPI job, for starting LAM software on each of the PBS execution hosts.

Usage is the same as for LAM’s lamboot. All arguments except for bhost are passed directly to lamboot. PBS will issue a warning saying that the bhost argument is ignored by PBS since input is taken automatically from $PBS_NODEFILE. The pbs_lamboot program will not redundantly consult the $PBS_NODEFILE if it has been instructed to boot the hosts using the tm module. This instruction happens when an argument is passed to pbs_lamboot containing "-ssi boot tm" or when the LAM_MPI_SSI_boot environment variable exists with the value tm.

11.1.3.2 The pbs_mpilam Command

The PBS command pbs_mpilam replaces the standard mpirun command in a PBS LAM MPI job, for executing programs. It attaches the user’s processes to the PBS job. This allows PBS to collect accounting information, and to manage the processes.
Usage is the same as for LAM `mpirun`. All options are passed directly to `mpirun`. If the where argument is not specified, `pbs_mpilam` will try to run the user’s program on all available CPUs using the C keyword.

### 11.1.3.3 PATH

The PATH for `pbs_lamboot` and `pbs_mpilam` on all remote machines must contain `PBS_EXEC/bin`.

### 11.1.3.4 Transparency to the User

Both `pbs_lamboot` and `pbs_mpilam` should be transparent to the user. Users should be able to run existing scripts.

To be transparent to the user, `pbs_lamboot` should replace LAM `lamboot`. The link for `lamboot` should be changed to point to `pbs_lamboot`.

1. Install LAM MPI into `/usr/local/lam-<version>`
   ```
   mv /usr/local/lam-<version>/bin/lamboot /user/local/lam-<version>/bin/lamboot.lam
   ```
2. Edit `pbs_lamboot` to change “lamboot” call to “lamboot.lam”
3. Rename `pbs_lamboot` to `lamboot`:
   ```
   cd /usr/local/lam-<version>/bin
   ln -s PBS_EXEC/bin/pbs_lamboot lamboot
   ```
   At this point, using “lamboot” will actually invoke `pbs_lamboot`. 
To be transparent to the user, `pbs_mpilam` should replace LAM `mpirun`. The link for `mpirun` should be changed to point to `pbs_mpilam`.

1. Install LAM MPI into `/usr/local/lam-<version>`
   
   ```bash
   mv /usr/local/lam-<version>/bin/mpirun
   /usr/local/lam-<version>/bin/mpirun.lam
   ```

2. Edit `pbs_mpilam` to change “mpirun” call to “mpirun.lam”

3. Rename `pbs_mpilam` to `mpirun`:
   
   ```bash
   cd /usr/local/lam-<version>/bin
   ln -s PBS_EXEC/bin/pbs_mpilam mpirun
   ```

Either LAMRSH or LAM_SSI_rsh_agent will need to have the value "ssh -x", depending on whether you are using rsh or ssh.

### 11.1.4 Integration with HP MPI on HP-UX and Linux

#### 11.1.4.1 The `pbs_mpihp` Command

The PBS command `pbs_mpihp` replaces the standard `mpirun` and `mpiexec` commands in a PBS HP MPI job on HP-UX and Linux, for executing programs. It attaches the user’s processes to the PBS job. This allows PBS to collect accounting information, and to manage the processes.
11.1.4.2 Transparency to the User

To be transparent to the user, pbs_mpihp should replace HP mpirun. The recommended steps for making pbs_mpihp transparent to the user are:

1. Rename HP’s mpirun:
   ```
   cd <MPI installation location>/bin
   mv mpirun mpirun.hp
   ```

2. Link the user-callable “mpirun” to pbs_mpihp:
   ```
   cd <MPI installation location>/bin
   ln -s $PBS_EXEC/bin/pbs_mpihp mpirun
   ```

3. Create a link to mpirun.hp from PBS_EXEC/etc/pbs_mpihp. pbs_mpihp will call the real HP mpirun:
   ```
   cd $PBS_EXEC/etc
   ln -s <MPI installation location>/bin/mpirun.hp
   pbs_mpihp
   ```

When wrapping HP MPI with pbs_mpihp, note that rsh is the default used to start the mpids. If you wish to use ssh or something else, be sure to set the following or its equivalent in $PBS_HOME/pbs_environment:

```
PBS_RSHCOMMAND=ssh
```}

11.1.5 SGI MPI on the Altix Running Supported Versions of ProPack or SGI Performance Suite

PBS supplies its own mpiexec on the Altix. This mpiexec uses the standard SGI mpirun. No unusual setup is required for either mpiexec or mpirun, however, there are prerequisites. See the following section. If executed on a non-Altix system, PBS’s mpiexec will assume it was invoked by mistake. In this case it will use the value of PATH (outside of PBS) or PBS_O_PATH (inside PBS) to search for the correct mpiexec
and if one is found, exec it. The name of the array to use when invoking mpiexec is user-specifiable via the PBS_MPI_SGIARRAY environment variable.

The PBS mpiexec is transparent to the user; MPI jobs submitted outside of PBS will run as they would normally. MPI jobs can be launched across multiple Altixes. PBS will manage, track, and cleanly terminate multi-host MPI jobs. PBS users can run MPI jobs within specific partitions.

If CSA has been configured and enabled, PBS will collect accounting information on all tasks launched by an MPI job. CSA information will be associated with the PBS job ID that invoked it, on each execution host. While each host involved in an MPI job will record CSA accounting information for the job if able to do so on the execution hosts, there is no tool to consolidate the accounting information from multiple hosts.

If the PBS_MPI_DEBUG environment variable's value has a nonzero length, PBS will write debugging information to standard output.

PBS uses the MPI-2 industry standard mpiexec interface to launch MPI jobs within PBS.

11.1.5.1 Prerequisites

• In order to run single-host or multi-host jobs, the SGI Array Services must be correctly configured. An Array Services daemon (arrayd) must run on each host that will run MPI processes. For a single-host environment, arrayd only needs to be installed and activated. However, for a multi-host environment where applications will run across hosts, the hosts must be properly configured to be an array.

• Altix systems communicating via SGI's Array Services must all use the same version of the sgi-mpt and sgi-arraysvcs packages. Altix systems communicating via SGI's Array Services must have been configured to interoperate with each other using the default array. See SGI's array_services(5) man page.

• “rpm -qi sgi-arraysvcs” should report the same value for Version on all systems.

• “rpm -qi sgi-mpt” should report the same value for Version on all
Integrations

Chapter 11

systems.

- “chkconfig array” must return “on” for all systems
- /usr/lib/array/arrayd.conf must contain an array definition that includes all systems.
- /usr/lib/array/arrayd.auth must be configured to allow remote access:
  The “AUTHENTICATION NOREMOTE” directive must be commented out or removed
  Either “AUTHENTICATION NONE” should be enabled or keys should be added to enable the SIMPLE authentication method.
- If any changes have been made to the arrayd configuration files (arrayd.auth or arrayd.conf), the array service must be restarted.
- rsh(1) must work between the systems.
- PBS uses SGI's mpirun(1) command to launch MPI jobs. SGI’s mpirun must be in the standard location.
- The location of pbs_attach(8B) on each vnode of a multi-vnode MPI job must be the same as it is on the mother superior vnode.

11.1.5.2 Environment Variables

The PBS mpiexec script sets the PBS_CPUSET_DEDICATED environment variable to assert exclusive use of the resources in the assigned cpuset.

The PBS mpiexec checks the PBS_MPI_DEBUG environment variable. If this variable has a nonzero length, debugging information is written.

If the PBS_MPI_SGIARRAY environment variable is present, the PBS mpiexec will use its value as the name of the array to use when invoking mpirun.

The PBS_ENVIRONMENT environment variable is used to determine whether mpiexec is being called from within a PBS job.

The PBS mpiexec uses the value of PBS_O_PATH to search for the correct mpiexec if it was invoked by mistake.
11.1.6 SGI’s MPI (MPT) Over InfiniBand

PBS jobs can run using SGI’s MPI, called MPT, over InfiniBand. To use InfiniBand, set the MPI_USE_IB environment variable to 1.

11.1.7 The pbsrun_wrap Mechanism

PBS provides a mechanism for wrapping several versions/flavors of mpirun so that PBS can control jobs and perform accounting. PBS also provides a mechanism for unwrapping these versions of mpirun. The administrator wraps a version of mpirun using the pbsrun_wrap script, and unwraps it using the pbsrun_unwrap script. The pbsrun_wrap script is the installer script that wraps mpirun in a script called “pbsrun”. The pbsrun_wrap script instantiates the pbsrun script for each version of mpirun, renaming it to reflect the version/flavor of mpirun being wrapped. When executed inside a PBS job, the pbsrun script calls a version-specific initialization script which sets variables to control how the pbsrun script uses options passed to it. The pbsrun script uses pbs_attach to give MOM control of jobs.

The pbsrun_wrap command has a “-s” option. If -s is specified, then the "strict_pbs" options set in the various initialization scripts (e.g. pbsrun.ch_gm.init, etc...) will be set to 1 from the default 0. This means that the mpirun being wrapped by pbsrun will only get executed if inside a PBS environment. Otherwise, the user will get the error:

```
Not running under PBS
Exiting since strict_pbs is enabled; execute only in PBS
```

The pbsrun_wrap command has this format:

```
pbsrun_wrap [-s] <path_to_actual_mpirun> pbsrun.<keyword>
```

If the mpirun wrapper script is run inside a PBS job, then it will translate any mpirun call of the form:

```
mpirun [options] <executable> [args]
```
Integrations

Chapter 11

into

\texttt{mpirun [options] pbs\_attach [special\_option\_to\_pbs\_attach]}
\begin{verbatim}
<executable> [args]
\end{verbatim}

where [special options] refers to any option needed by pbs\_attach to do its job (e.g. \texttt{-j $PBS\_JOBID}).

If the wrapper script is executed outside of PBS, a warning is issued about "not running under PBS", but it proceeds as if the actual program had been called in standalone fashion.

Any mpirun version/flavor that can be wrapped has an initialization script ending in ".init", found in \$PBS\_EXEC/lib/MPI:

\begin{verbatim}
$PBS\_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init.
\end{verbatim}

The pbsrun\_wrap script instantiates the pbsrun wrapper script as pbsrun.<mpirun version/flavor> in the same directory where pbsrun is located, and sets up the link to the actual mpirun call via the symbolic link

\begin{verbatim}
$PBS\_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link
\end{verbatim}

For example, running:

\texttt{pbsrun\_wrap /opt/mpich-gm/bin/mpirun.ch\_gm pbsrun.ch\_gm}

causes the following actions:

\begin{itemize}
\item Save original mpirun.ch\_gm script:

\begin{verbatim}
mv /opt/mpich-gm/bin/mpirun.ch\_gm \\
/opt/mpich-gm/bin/mpirun.ch\_gm.actual
\end{verbatim}

\item Instantiate pbsrun wrapper script as pbsrun.ch\_gm:

\begin{verbatim}
cp $PBS\_EXEC/bin/pbsrun $PBS\_EXEC/bin/ pbsrun.ch\_gm
\end{verbatim}

\item Link "mpirun.ch\_gm" to actually call "pbsrun.ch\_gm":

\begin{verbatim}
ln -s $PBS\_EXEC/bin/pbsrun.ch\_gm \\
/opt/mpich-gm/bin/mpirun.ch\_gm
\end{verbatim}

\item Create a link so that "pbsrun.ch\_gm" calls "mpirun.ch\_gm.actual":

\begin{verbatim}
ln -s /opt/mpich-gm/bin/mpirun.ch\_gm.actual \\
$PBS\_EXEC/lib/MPI/pbsrun.ch\_gm.link
\end{verbatim}
\end{itemize}
The `mpirun` being wrapped must be installed and working on all the vnodes in the PBS cluster.

For all wrapped MPIs, the maximum number of ranks that can be launched in a job is the number of entries in the `$PBS_NODEFILE`.

### 11.1.7.1 The `pbsrun` Script

See “pbsrun” on page 126 of the PBS Professional Reference Guide for a description of the `pbsrun` script.

### 11.1.7.2 Wrapping Multiple MPI’s with the Same Name

You may want more than one MPI environment with the same name, for example a 32-bit and a 64-bit version of MPICH2.

1. Create two new MPICH2 initialization scripts by copying that for MPICH2:

   ```bash
   # cd $PBS_EXEC/lib/MPI
   # cp pbsrun.mpich2.init.in pbsrun.mpich2_32.init.in
   # cp pbsrun.mpich2.init.in pbsrun.mpich2_64.init.in
   ```

2. Then wrap them:

   ```bash
   # pbsrun_wrap <path to 32-bit MPICH2>/bin/mpirun
   pbsrun.mpich2_32
   # pbsrun_wrap <path to 64-bit MPICH2>/bin/mpirun
   pbsrun.mpich2_64
   ```

Calls to "<path to 32-bit MPICH2>/bin/mpirun" will invoke `/usr/pbs/bin/pbsrun.mpich2_32`. The 64-bit version is invoked with calls to "<path to 64-bit MPICH2>/bin/mpirun".

When you are done using them, unwrap them:

```bash
# pbsrun_unwrap pbsrun.mpich2_32
# pbsrun_unwrap pbsrun.mpich2_64
```
11.1.7.3 Using IBM’s `poe`

See “IBM’s poe: pbsrun.poe” on page 142 of the PBS Professional Reference Guide.

You can use set the number of HPS US mode jobs MOM will accept:

Example 1: set node aix_15 to only accept one HPS US mode job at any one time:

```
# qmgr -c 'set node aix_15 resources_available.hps = 1'
```

Example 2: set node aix_75 to accept multiple HPS US mode jobs at any one time:

```
# qmgr -c 'set node aix_75 resources_available.hps = 99999'
```

You will need to set up a custom resource for the HPS so that `hps` is a static consumable host-level resource. See section 5.14.5.2 “Static Host-level Resources” on page 377. Users can request the “hps” resource in their select statements.

If you have some machines in the complex that are not on the HPS, be sure that those machines have their `hps` resource set to zero.

```
# qmgr -c 'set node not_ibm resources_available.hps = 0'
```

As an alternative, you can set `sharing=force_excl` to limit the number of HPS US mode jobs to 1, but it would be more restrictive. In this case, one and only one job could run on the HPS.

An example of the way to do this (in this case, changing the `sharing` attribute for a vnode named aix_15) uses the script “change_sharing”. See section 3.5.3 “Creating Version 2 MOM Configuration Files” on page 60.

```
# cat change_sharing
$configversion 2
aix_15: sharing = force_excl
# . /etc/pbs.conf
# $PBS_EXEC/sbin/pbs_mom -s insert force_excl change_sharing
```

Stop and restart the MOM.
11.2 Support for AIX

PBS Professional supports Large Page Mode on AIX. No additional steps are required from the PBS administrator. Certain applications (like many FEA Solvers) can benefit from using large page support. This allows programs to do considerably less page “thrashing”.

Setting the PBS environment to request large page mode is not recommended because every process started by a job will use large page mode. It is better for the user to explicitly request large page mode for the processes that should use large page mode.

11.3 User Space Mode on InfiniBand Switches

PBS can track the resources for MPI, LAPI programs or a mix of MPI and LAPI programs. LoadLeveler is not required in order to use InfiniBand switches in User Space mode. PBS works with a standard InfiniBand configuration.

11.3.1 Overview

The steps required to configure PBS to use InfiniBand in User Space mode are:

- Wrap the `poe` command. See the next section.
- If you want all InfiniBand jobs to have the number of required network windows calculated by the PBS server, enable this as described in section 11.3.3.3 “Automatic Network Window Requests” on page 794.
- If the default for the number of switch networks for each chunk is unacceptable, you can set a value. See section 11.3.3.1 “Set Value for Number of Switch Networks” on page 793.
- To limit the number of instances that can be requested for each chunk, see section 11.3.3.2 “Set Maximum Value for Number of Instances” on...
11.3.2 Wrapping the poe Command

The administrator wraps POE using `pbsrun_wrap`. For more information about `pbsrun_wrap`, see “`pbsrun_wrap`” on page 147 of the PBS Professional Reference Guide. The `pbsrun` script is instantiated as `pbsrun.poe`, which wraps the IBM `poe` command. A job invoking the `poe` command is redirected to `pbsrun.poe`. The `pbsrun.poe` command performs preprocessing, then calls the IBM `poe` command. See the `pbsrun(8B)` man page. The permissions for the `pbsrun.poe` command are `-rwxr-xr-x`.

Wrap the IBM `poe` command as root. At the command line:

```
pbsrun_wrap <path_to_actual_poe> pbsrun.poe
```

11.3.3 Tracking Network Windows

PBS has a host-level read-only resource called `netwins` for representing the network windows on a high performance switch. This resource can be used to schedule jobs that use the InfiniBand switch. The PBS server can calculate a value to use for `netwins` for each chunk in a job. If the server does not calculate the value, the MOM can calculate a default value. For the server to calculate the value, the `PBS_GET_IBWINS` environment variable must be set to `1` in the job’s environment.

We recommend that the administrator does not set a value for `netwins`, via either `qmgr` or MOM configuration files.

The `netwins` resource is included in the “resources” line in `PBS_HOME/sched_priv/sched_config` by default.

The value calculated is logged in the server log at the event class of `pbs.LOG_WARNING`.

11.3.3.1 Set Value for Number of Switch Networks

The number of switch networks per chunk defaults to 2.
If the default is not acceptable, define a custom server-level resource called PBS_ibm_ib_default_networks in the resourcedef file:

```
PBS_ibm_ib_default_networks type=long flag=r
```

Restart the server; see section 5.14.3.1 “Restarting the Server” on page 367. Set the new resource to the number of switch networks that all the InfiniBand adapters are connected with.

### 11.3.3.2 Set Maximum Value for Number of Instances

The number of instances per chunk defaults to 1.

Users can request a different number of instances by setting the MP_INSTANCES environment variable to that number. To set the maximum number of instances per chunk that users can request, define a custom server-level resource called PBS_ibm_ib_max_instances. Put the following in the resourcedef file:

```
PBS_ibm_ib_max_instances type=long flag=r
```

Set this resource to the maximum number of network windows to be allowed to be set per process. A conservative number can be calculated by:

```
PBS_ibm_ib_max_instances = ((number of adapters per node) * (number of network windows per adapter)) / ((number of CPUs per node) * 2)
```

If the user requests more than the maximum specified in PBS_ibm_ib_max_instances, the requested value is replaced with the maximum value.

### 11.3.3.3 Automatic Network Window Requests

To guarantee use of the InfiniBand switch, a job must have the environment variable PBS_GET_IBWINS set to 1. This allows the PBS server to calculate the value for netwins for each chunk in a job’s select specification. The environment variable can be set one of the following ways:

- To set the environment variable for all jobs, the administrator includes “-v PBS_GET_IBWINS = 1” in the server’s
Integrations

Chapter 11

default_qsub_arguments attribute.

Qmgr: set server default_qsub_arguments += "-v
PBS_GET_IBWINS = 1"

- To set the environment variable for each job, the user sets
 PBS_GET_IBWINS = 1 in their shell environment, and uses the -V
 option to the qsub command.

  - csh:

        setenv PBS_GET_IBWINS 1

  - bash:

        PBS_GET_IBWINS = 1
        export PBS_GET_IBWINS

- To set the environment variable for one job, the user uses the “-v
 PBS_GET_IBWINS = 1” option to the qsub command.

11.3.3.4 Assigning Network Windows to Chunks

PBS calculates the value of netwins to be assigned to each chunk as fol-
lows:

\[
\text{number of mpiprocs in chunk} \times \text{number of switch networks} \times \text{number of instances} \times \text{msg_api multiplier}
\]

\text{number of mpiprocs in chunk}

- The value for the mpiprocs resource

\text{number of switch networks}

- Defaults to 2
- The value of the custom server-level resource called
 PBS.ibm.ibm_default_networks. If this resource is
defined, this value takes precedence over the default
value.

\text{number of instances}

- Defaults to 1
- Value passed via qsub option -v MP_INSTANCES, or
 value passed via qsub option -V when
MP_INSTANCES is set in user’s shell environment. If the value of MP_INSTANCES is given, it takes precedence over the default value.

**msg_api multiplier**

- Defaults to 1
- Value is 2 if the user specifies “mpi,lapi” either via the -v MP_MSG_API option to qsub, or via the -V option to qsub when MP_MSG_API is set in the user’s shell environment. If the user specifies “mpi, lapi”, this takes precedence over the default value.

### 11.3.3.5 Using Default Number of Network Windows

If PBS has not assigned a value for netwins for a job that will use the InfiniBand switch, the PBS MOM calculates the number of network windows to configure on each switch:

\[
\text{number of mpiprocs} \times \text{number of switch networks}
\]

MOM queries any high performance network that is attached to her execution host.

If the user requests more than one MPI process per CPU, this default number of network windows may be too large, and prevent use of the InfiniBand switch.

### 11.3.3.6 Scheduling Jobs Using Network Windows

PBS can schedule jobs using the netwins resource, ensuring that the load on the InfiniBand switch is correct. To schedule jobs using this resource:

- The environment variable PBS_GET_IBWINS must be set to 1 in the job’s environment, as in section 11.3.3.3 “Automatic Network Window Requests” on page 794.
- The netwins resource must be in the resources line in PBS_HOME/sched/priv/sched_config.
11.3.4 Errors and Logging

The PBS MOM may encounter the following errors, and write these error messages to the MOM logfile:

- Error return from `nrt_status_adapter`

  The message shows the adapter name, error code from `nrt_status_adapter`, and `errno` result:
  
  01/24/2007
  18:57:26;0008;pbs_mom;Job;1.cyan;nrt_status_adapter(iba0) err 4; I/O error

- Error return from `nrt_load_table`

  The message shows the adapter name, error code from `nrt_load_table`, and `errno` result:
  
  01/24/2007
  18:57:26;0008;pbs_mom;Job;1.cyan;nrt_load_table(iba0) err 1; Invalid input parameter

- Error return from `nrt_unload_window`

  The message shows the adapter name, error code from `nrt_unload_window`, and `errno` result:
  
  01/24/2007
  18:57:26;0008;pbs_mom;Job;1.cyan;nrt_unload_window(iba0) err 12; I/O error

- Error return from `nrt_adapter_resources`

  The message shows the adapter name, error code from `nrt_adapter_resources`, and `errno` result:
  
  01/24/2007
  18:57:26;0008;pbs_mom;Job;1.cyan;nrt_adapter_resources(iba0) err 3; I/O error
11.4 Notes on SGI ICE

Vnodes on a SGI ICE are not allocated exclusively. They are shared, as are nodes in a normal cluster.

11.5 Support for Cray Systems

PBS provides features designed to support the Cray. This section of the documentation describes the special behavior of PBS on Cray systems only.

11.5.1 Introduction

PBS provides support for Cray systems by providing the following:

• PBS automatically defines vnodes for Cray compute nodes
• PBS automatically sets resources and attributes for vnodes representing Cray nodes
• PBS automatically creates custom resources that correspond to Cray resources
• Cray users can submit jobs through PBS using the PBS select and place statements.

11.5.2 Relationship of PBS Vnodes to Cray Nodes

PBS represents each login node as a vnode, and each NUMA node as a vnode. A compute node containing a single NUMA node is represented as a single vnode. The Cray inventory uses the term “segment” to refer to a NUMA node. A PBS MOM runs on each login node; this MOM manages the vnodes representing the compute nodes associated with the login node. On systems with multiple login nodes, each MOM on each login node manages every compute node. When this is the case, each compute node is
reported by more than one login node. The **Mom** attribute of a vnode representing a compute node contains the hostname of each login node reporting the compute node. Each hostname is the FQDN or the short name of the reporting login node, depending on whatever is returned by the DNS.

11.5.2.1 **How PBS Handles Changes in Cray Inventory**

11.5.2.1.1 **Reporting Changes in Vnode List**

If a previously-reported vnode is no longer reported when the vnode list is created, because it is no longer available in the vnode definition file and the inventory, it is missing from the vnode list. The server marks missing vnodes as **stale**.

PBS provides a Boolean MOM configuration option that allows you to specify whether MOM tells the server that a vnode is missing. When the **$vnodedef_additive** MOM configuration option in **PBS_HOME/mom_priv/config** is **True**, MOM does not tell the server that any vnodes are missing. This means that the server does not mark missing vnodes as **stale**. When **$vnodedef_additive** is **False**, MOM tells the server that vnodes are missing, and the server marks the missing vnodes as **stale**. The default value for **$vnodedef_additive** for a MOM managing a Cray is **False**. The default value for other systems is **True**.

When a compute node goes down and the ALPS inventory no longer reports it, the vnode is marked **stale** once PBS queries for the inventory. This happens when PBS fails to confirm a Cray reservation. If you bring the compute node back up, you can HUP the MOM in order to make the vnode usable by PBS. Alternatively, if PBS fails to make a Cray reservation, MOM will re-read the inventory and re-create the vnode list.

Each compute node can be managed by more than one login node. This means that the vnodes representing the NUMA nodes in a compute node can have more than one MOM reporting them.

A vnode representing a NUMA node is marked **stale** when any of the MOMs that reported the vnode stop reporting it.

The state of a vnode representing a NUMA node that is managed by more than one login node is not changed by a MOM going down, unless all of its MOMs are down. A vnode representing a NUMA node is marked **down** when all of the MOMs that manage the vnode are **down**.
11.5.2.1.2 When MOMs Report Conflicting Information

When one or more MOMs report information about a vnode, and one or more MOMs do not list the vnode, PBS uses the information from the MOMs that still report the vnode.

When more than one MOM reports information about a vnode, and the information conflicts, PBS uses the most recent information.

11.5.2.1.3 Vnode Resources on Stale Vnodes

If the ALPS inventory no longer reports information for a vnode, and the vnode is not defined in a vnode definition file, all resource and attribute information for that vnode is removed or set to zero, and the vnode is marked stale.

If the ALPS inventory no longer reports information for a vnode, but the vnode is defined in a vnode definition file, the vnode’s attributes and resources retain their settings and the vnode is not marked stale.

11.5.3 Requirements

PBS requires CLE 3.0 or higher.

11.5.4 Restrictions

A Cray compute node cannot be used by more than one application at the same time.

11.5.5 Resources, Parameters, Etc. for the Cray

PBS provides built-in and custom resources for the Cray. The nchunk resource is listed here for clarity.
11.5.5.1 Built-in Resources

nchunk

Used on all platforms. This is the number of chunks requested between plus symbols in a select statement. For example, if the select statement is `-l select 4:ncpus=2+12:ncpus=8`, the value of `nchunk` for the first part is 4, and for the second part it is 12. The `nchunk` resource cannot be named in a select statement; it can only be specified as a number preceding the colon, as in the above example. When the number is omitted, `nchunk` is 1.

Non-consumable.

This resource can be used to specify the default number of chunks at the server or queue (replacing `mppwidth`).

Example:

```sh
Qmgr: set queue myqueue default_chunk.nchunk=2
```

Settable by Manager and Operator; readable by all.

This resource cannot be used in server and queue `resources_min` and `resources_max`.

Format: `Integer`

Python type: `int`

Default value: 1

vntype

This resource represents the type of the vnode. Automatically set by PBS to one of two specific values for cray vnodes. Has no meaning for non-Cray vnodes. Can be requested only inside of a select statement.

Non-consumable.

Format: `String array`

Automatically assigned values for Cray vnodes:

- `cray_compute`
  
  This vnode represents part of a compute node.

- `cray_login`
This vnode represents a login node.

Default value: None
Python type: str

11.5.5.2 Custom Resources for the Cray

PBS creates custom resources that represent various Cray information. You do not need to do anything to create these resources.

**PBScrayhost**

On CLE 2.2, this is set to “default”.

On CLE 3.0 and higher, used to delineate a Cray system, containing ALPS, login nodes running PBS MOMs, and compute nodes, from a separate Cray system with a separate ALPS. Non-consumable. The value of PBScrayhost is set to the value of mpp_host for this system.

Format: String
Default: CLE 2.2: “default”; CLE 3.0 and higher: None

**PBScraylabel_<label name>**

Tracks labels applied to compute nodes. For each label on a compute node, PBS creates a custom resource whose name is a concatenation of PBScraylabel_ and the name of the label. PBS sets the value of the resource to True on all vnodes representing the compute node.

Format: PBScraylabel_<label name>

For example, if the label name is Blue, the name of this resource is PBScraylabel_Blue.

Format: Boolean
Default: None

**PBScraynid**

Used to track the node ID of the associated compute node. All vnodes representing a particular compute node share a value for PBScraynid. Non-consumable.

The value of PBScraynid is set to the value of node_id for this compute node.
Non-consumable.
Format: String
Default: None

**PBScrayorder**

Used to track the order in which compute nodes are listed in the Cray inventory. All vnodes associated with a particular compute node share a value for PBScrayorder. Non-consumable.

Vnodes for the first compute node listed are assigned a value of 1 for PBScrayorder. The vnodes for each subsequent compute node listed are assigned a value one greater than the previous value.

Format: Integer
Default: None

**PBScrayseg**

Tracks the segment ordinal of the associated NUMA node. For the first NUMA node of a compute host, the segment ordinal is 0, and the value of PBScrayseg for the associated vnode is 0. For the second NUMA node, the segment ordinal is 1, PBScrayseg is 1, and so on. Non-consumable.

Format: String
Default: None

### 11.5.5.3 Scheduler Attribute

**do_not_span_psets**

Specifies whether or not the scheduler requires the job to fit within one of the existing placement sets. When do_not_span_psets is set to True, the scheduler will require the job to fit within a single existing placement set. The scheduler checks all placement sets, whether or not they are currently in use. If the job fits in a currently-used placement set, the job must wait for the placement set to be available. If the job cannot fit within a single placement set, it will not run.
When this attribute is set to False, the scheduler will first attempt to place the job in a single placement set, but if it cannot, it will allow the job to span placement sets, running on whichever vnodes can satisfy the job’s resource request.

Format: Boolean

Default value: False (This matches behavior of PBS 10.4 and earlier)

Example: To require jobs to fit within one placement set:

Qmgr: set sched do_not_span_psets=True

11.5.5.4 MOM Configuration Option

$vnodedef_additive

Specifies whether MOM considers a vnode that appeared previously either in the inventory or in a vnode definition file, but that does not appear now, to be in her list of vnodes.

When $vnodedef_additive is True, MOM treats missing vnodes as if they are still present, and continues to report them as if they are present. This means that the server does not mark missing vnodes as stale.

When $vnodedef_additive is False, MOM does not list missing vnodes, the server’s information is brought up to date with the inventory and vnode definition files, and the server marks missing vnodes as stale.

Visible in configuration file on Cray only.

Format: Boolean

Default for MOM on Cray login node: False
11.5.6 Automatic Configuration

11.5.6.1 Vnode List Creation

You must create a vnode for each login node, but PBS automatically creates the vnodes representing NUMA nodes. PBS creates a list of vnodes by reading the Cray inventory and any vnode definition files. PBS does this automatically when any of the following happens:

- Startup of the MOM
- The MOM is HUPed
- PBS queries the Cray for the inventory (e.g., when PBS fails to confirm a Cray reservation)

11.5.6.2 Automatic Configuration of Compute Node Vnodes

PBS automatically creates one or more vnodes to represent each Cray compute node. Each vnode represents one Cray NUMA node.

11.5.6.3 Vnode Attribute and Resource Settings

PBS automatically sets the values of certain vnode attributes and resources. Vnode attribute and resource settings are derived from values returned in the inventory, according to the following rules:

resources_available.host
Values of node’s mpp_host and node_id are concatenated.
Format: &lt;mpp_host&gt;.&lt;node_id&gt;
Example: Given a compute node where mpp_host = examplehost and node_id = 8, resources_available.host is set to examplehost_8.
On CLE 2.2, “default” replaces value of mpp_host.

resources_available.PBScrayhost
On CLE 2.2, set to “default”.
On CLE 3.0 and higher, set to value of mpp_host
resources-available.PBScraylabel_<label name>
For each label on a compute node, PBS creates a custom Boolean resource, and sets it to True on all vnodes representing that compute node. The format for the name of these resources is PBScraylabel_<label>. For example, if the label is Blue, then the name of the Boolean resource is PBScraylabel_Blue.

resources-available.PBScraynid
The value of PBScraynid is set to the value of node_id for this compute node.

resources-available.PBScrayorder
Set to the position in the Cray node list of the associated node. If this vnode’s associated node was nth in the node list, the value of PBScrayorder is n.

resources-available.PBScrayseg
Set to the segment ordinal of the associated NUMA node.

resources-available.vntype
On compute nodes, set to cray_compute
On internal login nodes, set to cray_login

Mom vnode attribute
This is the canonical hostname of the login node where MOM runs.

Name of vnode
Value of node’s mpp_host, node_id, and segment ordinal are concatenated.

Format: <mpp_host>_ <node_id>_ <segment>

Example: Given segment 0, node_id = 8 and mpp_host = examplehost, the vnode name is examplehost_8_0.

On CLE 2.2, “default” replaces value of mpp_host.

sharing vnode attribute
Set to force_exclhost
11.5.6.4 MOM Parameter Settings

$vnodedef_additive MOM configuration option
PBS automatically sets the value of the $vnodedef_additive MOM configuration option to False on any MOM on a login node. See section 11.5.5.4 “MOM Configuration Option” on page 804.

11.5.6.5 Scheduler Attribute Settings

do_not_span_psets
This attribute is set to False by default. See section 11.5.5.3 “Scheduler Attribute” on page 803.

11.5.7 Recommended Manual Configuration

11.5.7.1 Set Scheduling Parameters

You must add the vntype resource to the “resources:” line in PBS_HOME/sched_priv/sched_config. If you want the scheduler to honor the following resources, add them as well:

- PBScrayhost
- PBScraynid
- PBScrayseg

Do not add the following resources to the “resources:” line:

- PBScrayorder
- nchunk
### 11.5.7.2 Setting Server and Queue Defaults

PBS does not support server or queue defaults for `mpp*` resources. You must any replace `resources_default.mpp*` settings as follows:

<table>
<thead>
<tr>
<th>Old Resource Default</th>
<th>New Resource Default</th>
<th>Set New Value to Old Value of This:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resources_default.mppwidth</code></td>
<td><code>default_chunk.nchunk</code></td>
<td><code>resources_default.mppwidth</code></td>
</tr>
<tr>
<td><code>resources_default.mppdepth</code></td>
<td><code>default_chunk.ncpus</code></td>
<td><code>resources_default.mppdepth</code></td>
</tr>
<tr>
<td><code>resources_default.mppnppn</code></td>
<td><code>default_chunk.mpprocs</code></td>
<td><code>resources_default.mppnppn</code></td>
</tr>
<tr>
<td>(must replace with two settings)</td>
<td><code>default_chunk.ncpus</code></td>
<td><code>resources_default.mppdepth</code></td>
</tr>
<tr>
<td><code>resources_default.mpparch</code></td>
<td><code>default_chunk.arch</code></td>
<td><code>resources_default.mpparch</code></td>
</tr>
<tr>
<td><code>resources_default.mpphost</code></td>
<td><code>default_chunk.PBScrayhost</code></td>
<td><code>CLE 2.2: default</code> \ <code>CLE 3.0 and higher: resources_default.mpphost</code></td>
</tr>
<tr>
<td><code>resources_default.mppmem</code></td>
<td><code>default_chunk.mem</code></td>
<td><code>resources_default.mppmem</code></td>
</tr>
</tbody>
</table>
11.5.7.3 Setting Server and Queue Minima and Maxima

PBS does not support server or queue minimum or maximum settings for *mpp* resources. You must any replace `resources_min.mpp*` and `resources_max.mpp*` settings as follows:

**Table 11-2: How to Replace mpp* Server/Queue Min/Max**

<table>
<thead>
<tr>
<th>Old Min/Max Setting</th>
<th>New Min/Max Setting</th>
<th>Set New Value to Old Value of This:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resources_min.mppwidth</code></td>
<td><code>resources_min.mpi procs</code></td>
<td><code>resources_min.mppwidth</code></td>
</tr>
<tr>
<td><code>resources_min.mppdepth</code></td>
<td><code>&lt;no replacement&gt;</code></td>
<td><code>&lt;no replacement&gt;</code></td>
</tr>
<tr>
<td><code>resources_min.mppnppn</code></td>
<td><code>&lt;no replacement&gt;</code></td>
<td><code>&lt;no replacement&gt;</code></td>
</tr>
<tr>
<td><code>resources_min.mpparch</code></td>
<td><code>resources_min.arch</code></td>
<td><code>resources_min.mpparch</code></td>
</tr>
<tr>
<td><code>resources_min.mppmem</code></td>
<td><code>&lt;no replacement&gt;</code></td>
<td><code>&lt;no replacement&gt;</code></td>
</tr>
<tr>
<td><code>resources_max.mppwidth</code></td>
<td><code>resources_max.mpi procs</code></td>
<td><code>resources_max.mppwidth</code></td>
</tr>
<tr>
<td><code>resources_max.mppdepth</code></td>
<td><code>&lt;no replacement&gt;</code></td>
<td><code>&lt;no replacement&gt;</code></td>
</tr>
<tr>
<td><code>resources_max.mppnppn</code></td>
<td><code>&lt;no replacement&gt;</code></td>
<td><code>&lt;no replacement&gt;</code></td>
</tr>
<tr>
<td><code>resources_max.mpparch</code></td>
<td><code>resources_max.arch</code></td>
<td><code>resources_max.mpparch</code></td>
</tr>
<tr>
<td><code>resources_max.mppmem</code></td>
<td><code>&lt;no replacement&gt;</code></td>
<td><code>&lt;no replacement&gt;</code></td>
</tr>
</tbody>
</table>

There is no support for `resources_min.nchunk` and `resources_max.nchunk`. If set, their behavior is undefined.
If you wish to set `resources_min.mpiprocs` or `resources_max.mpiprocs`, you must make sure that `mpiprocs` can be counted for each job chunk. If the job did not request `mpiprocs` with each chunk, the job must inherit `mpiprocs = 1` for each chunk.

Set `default_chunk.mpiprocs` to 1 on the server:

```
Qmgr: s s default_chunk.mpiprocs = 1
```

11.5.7.4 Using the `nchunk` Resource to Replace `mppwidth`

You can use the `nchunk` resource to replace the functionality of `mppwidth`. The `mppwidth` resource is used to indicate the number of PEs required by a job. Since a chunk is typically 1 PE, the `nchunk` resource can be used to indicate the number of PEs.

11.5.7.5 Setting Chunk Default For Vnode Type

Any job or reservation submitted with `mpp*` syntax is translated into select and place statements. Because the use of `mpp*` implies a request to use compute nodes, users should specify a vnode type of `cray_compute`. However, they may forget to do so. You can help out the users who are not used to specifying a vnode type to indicate that they are requesting compute nodes by setting `default_chunk.vntype` to `cray_compute`.

11.5.7.6 Allowing Scheduling on Cray Resources

If you want to be able to use the Cray resources such as `PBScrayhost`, `PBScrayseg`, `PBScraynid`, `vntype`, etc., you must add them to the `resources:` line in `PBS_HOME/sched_priv/config`.

Do not add `PBScrayorder` to the `resources:` line.

11.5.7.7 Using `qstat` Wrappers and Scripts

You must check any `qstat` wrappers or scripts. Either replace any `mpp*` resources, or modify the wrapper or script to use values in the job’s `Submit_arguments` attribute. This attribute contains the original job submission line.
11.5.7.8 Keeping Jobs Within One Host

To prevent jobs from being scheduled across multiple Cray hosts, you must limit jobs to a single value for PBScrayhost. Do the following:

- Set node_group_enable to True:
  \[
  \texttt{Qmgr: s s node_group_enable=True}
  \]
- Set node_group_key to PBScrayhost:
  \[
  \texttt{Qmgr: s s node_group_key=PBScrayhost}
  \]

If a job requests more resources than can be supplied from a single host, and the job does not specify a value for PBScrayhost, the job is scheduled across multiple Cray systems. To prevent this from happening, you can set the do_not_span_psets scheduler attribute to True. See section 11.5.5.3 “Scheduler Attribute” on page 803.

11.5.7.8.1 Caveat

You cannot use this technique on CLE 2.2, because PBScrayhost is set to “default” for all Cray hosts.

11.5.7.9 Allowing Scheduling on Nearby Vnodes

To help the scheduler place each job requiring more than one vnode on vnodes that are close to each other, make the scheduler sort the vnodes based on their values for PBScrayorder. The vnodes will be listed in the order that their nodes are listed in the Cray inventory. To do this, specify the following in PBS_HOME/sched_priv/config:

\[
\texttt{node_sort_key: “PBScrayorder LOW”}
\]
11.5.7.10 Allowing Users to Request Useful Groups of Nodes

Job submitters can use select and place to request the groups of vnodes they want. This is the same behavior that they were accustomed to getting with mppnodes. However, you must provide the tools. Users may need to group their nodes by the certain criteria, for example:

- Certain nodes are fast nodes
- Certain nodes share a required or useful characteristic
- Some combination of nodes gives the best performance for an application

For these cases, you can do either of the following:

- Create custom resources, and set them on each vnode so that the important characteristics of the vnode can be requested. For example, if a vnode is fast, create a custom string resource called “speed” and set it to fast on that vnode.
- Label each node with its important characteristics. For example, if a node is both fast and best for App1, give it two labels, fast, and BestForApp1. PBS creates custom Boolean resources called PBScraylabel_<label name> and sets them to True on the appropriate vnodes.

11.5.7.11 Allowing Users To Reserve N NUMA Nodes Per Compute Node

You can allow a user to submit a Cray job with a PBS resource request that matches the Cray aprun -sn feature, which reserves some number of NUMA nodes per host. PBS can request the correct resources, but the user must translate the aprun syntax into PBS select and place syntax, and you must create the required custom resource for vnode selection, and set its value to create groupings of vnodes.

For example, a user wants 24 PEs in total, but only wants to use three NUMA nodes per host. The Cray syntax is aprun -sn 3 -n 24. If you have two compute nodes nid2 and nid5, each with 4 NUMA nodes, and each NUMA node has 4 PEs, this request can be satisfied by taking the...
first 3 NUMA nodes on each host. The list of NUMA nodes is nid2_0, nid2_1, nid2_2, nid5_0, nid5_1, and nid5_2. In this example, the first 3 NUMA nodes on each host form a vnode grouping.

11.5.7.11.1 Creating Custom Resource for Vnode Selection

You must create vnode groupings for the vnodes representing each compute node. Create groupings for each possible number of NUMA nodes. For example, for a compute node with 2 NUMA nodes, and therefore 2 vnodes, you create a grouping made up of 1 vnode, and a grouping of two vnodes. For a compute node with 4 NUMA nodes, groupings can be as large as 4 vnodes.

You create groupings by creating a custom string array resource, for example named craysn, and setting it to the list of grouping sizes that can contain that vnode. So, for example, the grouping that contains 2 vnodes is made up of NUMA nodes 0 and 1, and the craysn resource contains the value 2 on those vnodes.

One vnode always participates in all groupings on its compute node. One vnode only participates in the grouping that contains all the vnodes on the compute node.

The following table shows the values for craysn for a 4-NUMA-node compute node:

<table>
<thead>
<tr>
<th>NUMA node</th>
<th>Value of craysn on This NUMA node’s Vnode</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMA node 0</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>NUMA node 1</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>NUMA node 2</td>
<td>3, 4</td>
</tr>
<tr>
<td>NUMA node 3</td>
<td>4</td>
</tr>
</tbody>
</table>

When a user requests a value of 3 for the craysn resource, the vnodes representing NUMA nodes 0, 1, and 2 are assigned to the job.
11.5.7.11.2  PBS Resource Request Reserving N NUMA Nodes Per Host

To make a request equivalent to `aprun -sn 3 -n 24`, and match the compute node exclusive behavior of the Cray, the user can specify the following:

```bash
qsub -lselect=24:ncpus=1:craysn=3 -lplace=exclhost
```

11.5.7.12  Allowing Users To Reserve Specific NUMA Nodes

You can allow a user to submit a Cray job with a PBS resource request that matches the Cray `aprun -sl` feature, which reserves specific NUMA nodes on each compute node on which the job runs. In this case, the user does not request specific compute nodes.

PBS can request the correct resources, but the user must translate the `aprun` syntax into PBS select and place syntax. Information on translating the `aprun` syntax into select and place is in section 11.3.2, "Automatic Translation of mpp* Resource Requests", on page 297 of the PBS Professional User’s Guide.

For example, a user wants a total of 8 PEs, while using only NUMA node 1 on each compute node. The `aprun` syntax is `aprun -sl 1 -n 8`.

There are multiple options for how users place their jobs when specifying NUMA nodes.

11.5.7.12.1  Specifying Individual Vnodes

PBS automatically creates a custom string resource called `PBScrayseg`, and sets the value for each vnode to be the segment ordinal for the associated NUMA node. See “PBScrayseg” on page 803.

If the `aprun` syntax is the following:

```bash
aprun -sl 1 -n 8
```

An equivalent resource request for a PBS job is the following:

```bash
qsub -lselect=8:ncpus=1:PBScrayseg=1
```
11.5.7.12.2 Specifying Multiple Vnodes

If `-s l` specifies a list of NUMA nodes, as in the following:

```
aprun -s l 0,1 -n 8
```

there are several ways to allow users to make the equivalent resource request.

- The PBS user can request separate chunks for each NUMA node:
  
  ```
  qsub -lselect=4:ncpus=1:PBScrayseg=0
  +4:ncpus=1:PBScrayseg=1
  ```

- If the user knows about the underlying hardware, the PBS resource request can take advantage of that. On a homogenous system with 2 NUMA nodes per compute node and 4 PEs per NUMA node, the PBS resource request can be the following:
  
  ```
  qsub -lselect=8:ncpus=1 -lplace=pack
  ```

- You can create custom resources that group the underlying NUMA nodes together. Do the following:
  
  a. Create a string array resource named, for example, `NUMAnode-`
**combo**

b. On every vnode, set `NUMAnodecombo` equal to all possible `-s1` combinations that might be requested that include that vnode (NUMA node).

The following table shows the values for the `s1` combinations for a 4-NUMA-node compute node:

**Table 11-4: s1 Combinations for 4-NUMA-node Compute Node**

<table>
<thead>
<tr>
<th>NUMA node</th>
<th>s1 Combinations for NUMA node</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,01,02,03,012,013,023,0123</td>
</tr>
<tr>
<td>1</td>
<td>1,01,12,13,012,013,123,0123</td>
</tr>
<tr>
<td>2</td>
<td>2,02,12,23,012,023,123,0123</td>
</tr>
<tr>
<td>3</td>
<td>3,03,13,23,013,023,123,0123</td>
</tr>
</tbody>
</table>

An equivalent select statement which uses this resource is the following:

```
qsub -lselect=8:ncpus=1:NUMAnodecombo=01 jobscript
```

• You can use the `vnsetofx` resource:

You create placement sets by creating a custom string array resource, for example named `vnsetof<number of vnodes>`, and setting it to the list of placement sets containing `<number of vnodes>` that can contain that vnode.

On a 4 NUMA-node compute node represented by 4 vnodes, the complete list of possible placement sets is:

\{0,1,2,3,01,02,03,12,13,23,012,013,023,123,0123\}.

The following table shows the values for `vnsetof<number of vnodes>` for a 4-NUMA-node compute node:

**Table 11-5: Values for vnsetof<number of vnodes>**

<table>
<thead>
<tr>
<th>NUMA Node</th>
<th>vnsetof1</th>
<th>vnsetof2</th>
<th>vnsetof3</th>
<th>vnsetof4</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMA node 0</td>
<td>0</td>
<td>01,02,03</td>
<td>012,013,023</td>
<td>0123</td>
</tr>
</tbody>
</table>
11.5.7.12.3 Caution When Creating Resource

You must use the same resource string values on each vnode. “012” is not the same as “102” or “201”.

11.5.7.13 Allowing Users to Request Login Node Groups

If users need to request groups of both esLogin nodes and internal login nodes, do the following:

1. Create a new string value for the vntype resource, for example `cray_compile`.

2. Use `qmgr` to set the value for vntype on the vnodes representing esLogin nodes:

   ```
   qmgr -c "set node es_Login
   resources_available.vntype+="cray_compile"
   "
   ```

3. Use `qmgr` to add `cray_compile` to the vnodes representing internal login nodes. (resources_available.vntype is automatically set to `cray_login`.)

   ```
   qmgr -c "set node internal_Login
   resources_available.vntype +="cray_compile"
   "
   ```

If you use `pbsnodes -av` to check resources_available.vntype for internal_Login, it now looks like this:

```
resources_available.vntype=cray_login,cray_compile
```
11.5.7.14 Allowing Users to Request Interlagos Hardware

You can provide users with a way to request or avoid Interlagos hardware. We recommend that you use `xtproccadmin` to set a label called “*interlagos*” on each compute node that uses Interlagos hardware. This way, PBS automatically creates the Boolean `PBScraylabel_interlagos` and sets it to `True` on each corresponding vnode.

If you cannot set the label, you can do the following:

1. On each vnode that belongs to the same host, edit the `server_priv/resourcedef` file and add a Boolean to represent whether or not a vnode uses Interlagos hardware:
   ```
   PBScraylabel_interlagos type=boolean flag=h
   ```
2. Restart the PBS server. See section 5.14.3.1 “Restarting the Server” on page 367.
3. Use `qmgr` to set the resource on the corresponding vnodes:
   ```
   Qmgr: set node <vnode name> resources_available.PBScraylabel_interlagos = True
   ```

Users can then request or avoid this resource using `PBScraylabel_interlagos=True` or `PBScraylabel_interlagos=False`. For example:

```
qsub -lselect=3:ncpus=2:PBScraylabel_interlagos=true myjob
```

11.5.8 Viewing Cray Information

11.5.8.1 Listing Vnodes

Each vnode appears only once in the output of any command that lists all vnodes, such as `pbs_statnode()`, `pbsnodes -av`, or `qmgr -c “list nodes @default”`. 
11.5.8.2 Contents of Vnode Mom Attribute

When multiple login nodes are defined, each vnode representing a compute node lists every reporting login node in its Mom attribute. The Mom attribute lists multiple fully-qualified host names in a comma-separated list format.

For example, MOM1 reports compute nodes with node_id 1 through 4, and MOM2 reports compute nodes with node_id 3 through 6. In this case, the vnodes representing compute nodes with node_id 1 and 2 list MOM1 in the Mom attribute, vnodes representing compute nodes with node_id 3 and 4 list MOM1 and MOM2 in the Mom attribute, and vnodes representing compute nodes with node_id 5 and 6 list MOM2 in the Mom attribute.

11.5.8.3 Viewing Job and Reservation Information

PBS translates all job and reservation mpp* syntax into select and place statements. See section 11.3.2, "Automatic Translation of mpp* Resource Requests", on page 297 of the PBS Professional User’s Guide. The job’s Submit_arguments attribute contains the original submission arguments. When you use the qstat -f, -x, or -H command to view job attributes, you see the following:

- The Resource_List attribute contains the translated select and place syntax, instead of the mpp* syntax
- The Submit_arguments attribute contains the original resource request

When you use the pbs_rstat -F command to view reservation attributes, you see the following:

- The Resource_List attribute contains the translated select and place syntax, instead of the mpp* syntax

11.5.8.4 Viewing Vnode Information

Each vnode’s jobs attribute lists the jobs that have processes executing on that vnode. Jobs launched from an internal login node, requesting a vntype of cray_compute only, are not listed in the internal login node’s
vnode’s jobs attribute. Jobs that are actually running on a login node, which requested a vntype of cray_login, do appear in the login node’s vnode’s jobs attribute.

You can view vnode attributes using the pbsnodes -av command.

If esLogin and internal login nodes are grouped by adding a string such as cray_compile to the vntype resource, the pbs_rstat -F command shows the following:

resources_available.vntype=cray_login,cray_compile

### 11.5.9 Jobs on the Cray

Cray users can submit jobs using the mpp* syntax; this is translated into select and place statements. See section 11.3.2, "Automatic Translation of mpp* Resource Requests", on page 297 of the PBS Professional User’s Guide for a translation description.

#### 11.5.9.1 Effect on Jobs of Stopping and Starting Vnodes

If a job is launched from a login node, and the MOM goes down, the impact on the job depends on the command-line options specified when the MOM is restarted. See “pbs_mom” on page 66 of the PBS Professional Reference Guide.

If a job is launched from a login node, and the login node goes down, the job does not continue to run.

If there are multiple login nodes, and one login node or its MOM goes down, jobs that were launched from other login nodes are not affected.

#### 11.5.9.2 Resource Request Translation

When a PBS job or reservation is submitted using the mpp* syntax, PBS translates the mpp* resource request into PBS resources. The translation uses the rules described in section 11.3.2, "Automatic Translation of mpp* Resource Requests", on page 297 of the PBS Professional User’s Guide.
11.5.9.3 Resource Accounting

Comprehensive System Accounting (CSA) runs on the compute nodes, under the control of the Cray system. PBS performs resource accounting on the login nodes, under the control of their MOMs.

11.5.10 Resource Restrictions and Deprecations

PBS translates only the following mpp* resources into select and place syntax:

mppwidth
mppdepth
mppnppn
mppmem
mpparch
mpphost
mpplabels
mppnodes

The mpp* syntax is deprecated. See section 1.3 “Deprecations” on page 10.

The following are not supported, and if set, behavior is undefined.

resources_min.nchunk
resources_max.nchunk.
11.5.11 Caveats and Advice

11.5.11.1 Configuring Cray MOMs

When configuring a Cray MOM, follow the rules in section 3.5.2.2 “Configuring Multi-vnode Machines without cpusets” on page 59. Note that since the vnode sharing attribute must be set using the `pbs_mom -s insert` command, it is not recommended to set the sharing attribute on a Cray vnode.

11.5.11.2 Vnode Definition Files Not Recommended

Using vnode definition files on a Cray is not recommended. Use `qmgr` where possible instead.

Any attribute and resource settings for a specific vnode in a vnode definition file cause PBS to believe that that vnode is still usable. If PBS reads the Cray inventory, and a vnode is not listed in the inventory, but it is listed in a vnode definition file, the vnode is not marked as `stale`. This will cause a problem when the scheduler tries to schedule jobs onto this vnode.

If you create a vnode definition file for a vnode that has more than one MOM, you must make sure that the files are consistent on all of the MOMs that manage the vnode.

Settings in a vnode definition file override those from inventory. Settings in `qmgr` override both vnode definition files and inventory. Be careful not to overwrite information from the inventory when creating a vnode definition file. For example, if the `vntype` resource for a vnode is set to `cray_login` when PBS reads the inventory, and it is set to `cray_compile` in a vnode definition file, the value for `vntype` becomes `cray_compile` only.

11.5.11.3 Use Correct Name When Creating Vnode

When creating a vnode to represent a login node, use the short name returned by the `gethostname` command on the login node. For example, if `gethostname` returns `HostA`, do the following:

```
Qmgr: create node HostA
```
If you create a vnode with a different name from the short name returned by gethostname, the following happens:

- MOM creates a vnode whose name is the short name returned by gethostname
- The vnode you created is not recognized by MOM, and is marked stale

11.5.11.4 Deleting Vnodes on Cray

You can delete a natural vnode only if no other vnodes list this vnode in their Mom attributes. In order to delete a natural vnode which is listed in another vnode’s Mom attribute, you must first delete the vnode with this vnode in its Mom attribute.

After removing a vnode that is managed by more than one MOM, you must HUP all of the managing MOMs, otherwise the vnode is not marked stale by the server.

11.5.11.5 Do Not Make Vnode Definitions Additive

On a Cray MOM, the $vnodedef_additive parameter in PBS_HOME/mom_priv/config is set to False or 0 by default. Do not unset or change the setting of the $vnodedef_additive parameter.

11.5.11.6 For CLE 2.2, No Value for mpp_host

On CLE 2.2, with BASIL 1.1, there is no value for mpp_host. Therefore, you cannot use this to distinguish between Cray hosts.

11.5.11.7 Do Not Use configrm

It is not recommended to use the configrm pbs_tclsh call.

11.5.11.8 Using Gating Values As Defaults

For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. If this is set at the queue, the queue value of resources_max.<resource> is used. If this is set only at the server, the
job inherits the value set at the server. However, for mpp* resources, the job does not inherit the gating value. For example, if the job does not request mppnppn, and no defaults are set at the server and queue, but resources_max.mppnppn is set at the queue, the job does not inherit the queue’s value.

### 11.5.11.9 Marking Cray Vnodes offline

You can use the `qmgr` command to individually mark each vnode representing a compute node offline. This is independent of the vnodes representing login nodes.

### 11.5.11.10 Do Not Use PBS-reserved Resource Names

Do not create resources with names that could be used by PBS to create a Cray resource equivalent. For example, do not create a resource with the name `PBScraylabel_small`.

### 11.5.11.11 Regularly Re-reading Inventory

If you want PBS to re-read the Cray inventory on a regular schedule, you can set up a `cron` job to HUP the Cray MOM(s).

### 11.5.11.12 Do Not Schedule on PBScrayorder

Do not add PBScrayorder to the `resources:` line.

### 11.5.12 Errors and Logging

#### 11.5.12.1 Creating Custom Resources

When a custom resource is created for a Cray vnode, the server logs a message containing the resource name and type, and the vnode name. This is logged at event class 0x080.
If a custom resource can't be created, the following error message is printed in the server log:

```
error: resource <name> for vnode <name> cannot be defined
```

### 11.5.12.2 Job Requests More Than Available

If `do_not_span_psets` is set to `True`, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:
  
  "Not Running: can't fit in the largest placement set, and can't span placement sets"

- The following message is printed to the scheduler's log:
  
  "Can't fit in the largest placement set, and can't span placement sets"

### 11.5.12.3 Invalid Cray Requests

It is possible to create a select and place statement that meets the requirements of PBS but not of the Cray. The Cray `width` and `depth` values cannot be calculated from `ncpus` and `mpiprocs` values. For example, if `ncpus` is 2 and `mpiprocs` is 4, the `depth` value is calculated by dividing `ncpus` by `mpiprocs`, and is one-half. This is not a valid `depth` value for Cray. When a select statement does not meet Cray requirements, and the Cray reservation fails, the following error message is printed in MOM's log, at log event class 0x080:

```
Fatal MPP reservation error preparing request
```

### 11.5.12.4 Ompthreads and ncpus Not Equal

If the value of `ompthreads` does not match the value of `ncpus` when PBS is constructing exec_vnode for a job, the following is printed in the MOM log, at event class 0x080:

```
"ompthreads <value> does not match ncpus <value>"
```
11.5.12.5 All Requested mppnodes Not Found

If mppnodes are requested, but there are no vnodes that match the requested mppnodes (i.e. 0% of the mppnodes list is found), the job or reservation is rejected with the following message:

“The following error was encountered: No matching vnodes for the given mppnodes <mppnodes>”

A log message is printed to the server log at event class 0x0004:

“translate mpp: ERROR: could not find matching vnodes for the given mppnodes <mppnodes (as input)>”

11.5.12.6 Some Requested mppnodes Not Found

If mppnodes are requested, and only some of the mppnodes are found to match the vnodes, then the job/reservation is accepted, but the following is printed in the server log at event class 0x0004:

“translate mpp: could not find matching vnodes for these given mppnodes [<comma separated list of mppnodes>]”

The job may or may not run depending on whether the vnodes that were matched up to the requested mppnodes have enough resources for the job.

11.5.12.7 Bad mppnodes Range

If the resource request specifies an mppnodes range with the value on the right hand side of the range less than or equal to the value on the left hand side of the range, the job or reservation is rejected with the following message:

The following error was encountered:

Bad range '<range>', the first number (<left_side>) must be less than the second number (<right_side>)

A log message is printed to the server log at event class 0x0004:

“translate mpp: ERROR: bad range '<range>', the first number (<left_side>) must be less than the second number (<right_side>)"
11.5.12.8 Resource Request Containing Both mpp* and select/place

If a resource request contains both mpp* and select/place, the job or reservation is rejected, and the following error is printed:

"The following error was encountered:
mpp resources cannot be used with "select" or "place""

11.6 Configuring MOM for Machines with cpusets

The enhanced PBS MOM called pbs_mom.cpuset is designed to manage a machine with cpusets. Using cpusets on the Altix requires the SGI ProPack library or SGI Performance Suite. See SGI’s documentation for more information. The standard PBS MOM can also manage a machine with cpusets, but PBS and the jobs it manages will not create or otherwise make use of them.

11.6.1 Vnodes and cpusets

A cpuset is a list of CPUs and memory nodes managed by the OS. Processes executing within a cpuset are typically confined to use only the resources defined by the set. An Altix using pbs_mom.cpuset will present multiple vnodes to the PBS server; these in turn are visible when using commands such as pbsnodes. Each of these vnodes is being managed by the one instance of pbs_mom.cpuset.

11.6.2 Rules for Creating cpusets

When you configure vnodes on an Altix, you can tell PBS that there are up to the actual number of CPUs in each vnode, but no more. The Altix assigns real hardware when it creates a cpuset. It tries to create cpusets containing the number of CPUs that you specified to PBS. PBS will try to assign all the CPUs in a cpuset to a job requiring that number. So if you
tell PBS that a cpuset contains more than the number of actual CPUs, then when the Altix tries to create a cpuset for that job, it will fail and the job won’t run.

For example, if a vnode has 2 physical CPUs, you can tell PBS that there are 0, 1, or 2 CPUs, but no more. If you tell PBS that the vnode has 4 CPUs, the Altix will not be able to create the cpuset since only 2 CPUs are available.

It is not recommended to change the number of CPUs reported by MOM.

11.7 Configuring MOM on a Cpusetted Machine

The rules of thumb for configuring a cpusetted machine are different from the rules for a non-cpusetted machine. Use the rules in section 3.5.2.3 “Configuring Machines with cpusets” on page 59.

To verify which CPUs are included in a cpuset created by PBS, on supported versions of ProPack or Performance Suite, use:

```
    cpuset -d <set name> | egrep cpus
```

This will work either from within a job or not.

The alt_id returned by MOM has the form cpuset=<name>. <name> is the name of the cpuset, which is the $PBS_JOBID.

A cpusetted machine can have a "boot cpuset" defined by the administrator. A boot cpuset contains one or more CPUs and memory boards and is used to restrict the default placement of system processes, including login. If defined, the boot cpuset will contain CPU 0. By default, the PBS MOM will not use the boot cpuset. The CPUSET_CPU_EXCLUSIVE flag prevents CPU 0 from being used by the MOM in the creation of job cpusets. This flag is set by default.

The MOM excludes from its use all CPUs in sets not belonging to PBS. The way to reserve some for other uses is to create a boot CPU set.
In order to use `pbs_mom.cpuset` on an Altix, you will need a vnode definitions file, which contains all the information about the machine’s vnodes and their resources. This is used by PBS for scheduling jobs. Each Altix may have a different topology, depending on how it is wired. The PBS startup script creates the vnode definitions file for supported versions of ProPack and SGI Performance Suite if it detects that `pbs_mom.cpuset` has been copied to `pbs_mom`.

The cpuset hierarchy has changed for version 8.0 and later. There are no directories under `/PBSPro` for shared or suspended cpusets.

On a suspend request, the cpuset MOM will move the processes to the global cpuset, then restore them later upon restart.

When PBS Professional creates job cpusets, it does not set the CPU or memory exclusive flags. PBS manages the exclusivity on these cpusets.

### 11.7.1 Configuring MOM for an Altix Running Supported Versions of ProPack or SGI Performance Suite

On an Altix running supported versions of ProPack, the vnode definitions file is generated automatically by PBS. The MOM includes routers automatically when she generates the file. There is a script which can be modified to produce different vnode definitions. The script is `$PBS_EXEC/lib/init.d/sgigenvnodelist.awk`. This script is designed to be modified by the PBS administrator. It is an alternative to using `pbs_mom -s` to insert changed vnode definitions. This script affects only the local host.

### 11.7.1.1 Manual Creation of cpusets Not Managed by PBS

You may wish to create cpusets not managed by PBS on an Altix running supported versions of ProPack or SGI Performance Suite. If you have not started PBS, create these cpusets before starting PBS. If you have started PBS, requeue any jobs, stop PBS, create your cpuset(s), then restart PBS.
For example, to create a boot cpuset, stop the PBS MOM, make the change to the boot cpuset, then restart the PBS MOM. The PBSPro cpuset is created at start up and will be smaller by the number of CPUs and amount of memory in the boot cpuset.

11.7.2 SGI cpuset-specific Configuration Parameters in Version 1 MOM Configuration File

cpuset_create_flags <flags>

    CPUSet_CPU_EXCLUSIVE | 0

Default: CPUSet_CPU_EXCLUSIVE

cpuset_destroy_delay <delay>

    MOM will wait delay seconds before destroying a cpuset of a just-completed job. This allows processes time to finish.

Default: 0. Integer. For example,

    cpuset_destroy_delay 10
11.7.3 Switching From Standard MOM to Cpusetted MOM on Altix

Here are the steps:

1. Using `qmgr`, delete the vnode run by the MOM to be switched:
   
   `Qmgr: delete node foo`

2. Stop PBS:

   `/etc/init.d/pbs stop`

3. Change directory to `PBS_EXEC`

4. Copy cpusetted MOM to MOM:

   `cp pbs_mom.cpuset pbs_mom`

5. Start PBS:

   `/etc/init.d/pbs start`

6. Using `qmgr`, create natural vnode:

   `Qmgr: create node foo`

For information on using `qmgr`, see “qmgr” on page 178 of the PBS Professional Reference Guide.

When you switch from the standard MOM to the cpusetted MOM, you’ll need to create a modified vnode definitions file with any changes that you made previously via `qmgr`. Use the `pbs_mom -s insert` command to add it.

Do not set `mem`, `vmem`, `ncpus` or `sharing` on the new vnode.

11.7.4 Switching From Cpusetted MOM to Standard MOM on Altix

If you switch from the cpusetted MOM to the standard MOM on the Altix, you’ll need to remove any vnode definition files you added that contain information dependent on the automatically-generated ones.
Chapter 11 Integrations

Remove your own vnode definitions files. List them:

\texttt{pbs\_mom \textasciitilde s list}

Remove each file you added:

\texttt{pbs\_mom \textasciitilde s remove <scriptname>}

Add new configuration files with any information you need:

\texttt{pbs\_mom \textasciitilde s insert <new scriptname>}

Then stop and start the MOM to get the changes to take effect.

### 11.7.5 Troubleshooting Supported Versions of ProPack and Performance Suite cpusets

The ProPack cpuset-enabled mom may occasionally encounter errors during startup from which it cannot recover without help. If \texttt{pbs\_mom} was started without the \texttt{-p} flag, one may see

"/PBSPro hierarchy cleanup failed in <dir> - restart \texttt{pbs\_mom} with \texttt{'-p'}"

where \texttt{<dir>} is one of \texttt{/PBSPro}, \texttt{/PBSPro/shared}, or \texttt{/PBSPro/suspended}. If this occurs, try restarting \texttt{pbs\_mom} with the \texttt{-p} flag. If this succeeds, no further action will be necessary to fix this problem. However, it is possible that if \texttt{pbs\_mom} is started with the \texttt{-p} flag, one may then see any of these messages:

"cpuset\_query for / failed – manual intervention is needed"

"/PBSPro query failed – manual intervention is needed"

"/PBSPro cpuset\_getmems failed – manual intervention is needed"
In this case, there is likely to be something wrong with the PBSPro cpuset hierarchy. First, use the `cpuset(1)` utility to test it:

```bash
# cpuset -s /PBSPro -r | while read set
  do
    cpuset -d $set > /dev/null
  done
```

If `cpuset` detects no problems, no output is expected. If a problem is seen, expect output of the form:

```
cpuset </badset> query failed
/badset: Unknown error
```

In this case, try to remove the offending cpuset by hand, using the `cpuset(1)` utility,

```bash
# cpuset -x badset
cpuset <badset> removed.
```

This may fail because the named cpuset contains other cpusets, because tasks are still running attached to the named set, or other unanticipated reasons. If the set has subsets,

```bash
# cpuset -x nonempty
cpuset <nonempty> remove failed
/nonempty: Device or resource busy
```
first remove any cpusets it contains:

```
# cpuset -s nonempty -r
/nonempty
/nonempty/subset
...
```

```
# cpuset -s nonempty -r | tac | while read set
do
cpuset -x $set
done
...
cpuset </nonempty/subset> removed.
cpuset </nonempty> removed.
```

Note that output is previous output, reversed.

If the set has processes that are still attached,

```
# cpuset -x busy
cpuset <busy> remove failed
/busy: Device or resource busy
```

one can choose to either kill off the processes,

```
# kill `cpuset -p busy`
# cpuset -x busy
cpuset <busy> removed.
```

or wait for them to exit. In the latter case, be sure to restart `pbs_mom` using the `-p` flag to prevent it from terminating the running processes.
Finally, note that if removing a cpuset with `cpuset -x` should fail, one may also try to remove it with `rmdir(1)`, provided one takes care to prepend the cpuset file system mount point first. For example,

```
  # mount | egrep cpuset
  cpuset on /dev/cpuset type cpuset (rw)
  # find /dev/cpuset/nonempty -type d -print | tac |
  while read set
    do
      rmdir $set
    done
```

11.7.6 Configuring MOM on SGI ICE with ProPack and SGI Performance Suite

If you are running the cpuset MOM, PBS will manage the cpusets on the SGI ICE. If you are running the standard MOM, PBS will not manage the cpusets.

If you are running the cpuset MOM, the `init.d/pbs` script will configure one vnode per MOM. This enables cpusets and sets sharing to `default_shared`. The `init.d/pbs` script affects only the local host.

To provide the maximum number of available CPUs on a small node, make sure that the file `/etc/sgi-compute-node-release` is present. This way, on installation the `pbs_habitat` script will add a "`cpuset_create_flags 0`" to MOM's `config` file.

In order to exclude CPU 0, change the MOM configuration file line to

```
  cpuset_create_flags CPUSET_CPU_EXCLUSIVE
```

This flag controls only whether CPU 0 is included in the PBS cpuset.

There is only one logical memory pool available per node on the SGI ICE.

If, at startup, MOM finds that any CPU in an existing, non-root, non-PBS cpuset, and/or the flag `cpuset_create_flags CPUSET_CPU_EXCLUSIVE` is set in the `mom_priv/config` file (as described in the paragraph above), meaning that CPU 0 will not be used by PBS, MOM will do the following:
Chapter 11  Integrations

- Exclude that CPU from the top set /dev/cpuset/PBSPro
- Create the top set with mem_exclusive set to False

Otherwise, the top set is created using all CPUs and with mem_exclusive set to True.

11.8  MOM Globus Configuration

For the optional Globus MOM, the same configuration mechanism applies as with the regular MOM except only three MOM configuration parameters are applicable:

$clienthost

$restrict_user

$logevent


11.9  Using Comprehensive System Accounting

11.9.1  Using CSA with PBS

If CSA is enabled, PBS can request the kernel to write user job accounting data to accounting records. These records can then be used to produce reports for the user.
If PBS finds the CSA shared object libraries, and CSA is enabled, PBS can cause a workload management record to be written for each job. If MOM is configured for CSA support, MOM can issue CSA workload management record requests to the kernel. The kernel writes workload management accounting records associated with the PBS job to the system-wide process accounting file. The default for this file is `/var/csa/day/pacct`.

### 11.9.2 CSA Configuration Parameter

`pbs_accounting_workload_mgmt <value>`

MOM configuration parameter. Controls whether CSA accounting is enabled. The name does not start with a dollar sign. If set to “`1`”, “`on`”, or “`true`”, CSA accounting is enabled. If set to “`0`”, “`off`”, or “`false`”, CSA accounting is disabled. Values are case-insensitive. Default: “`true`”; enabled.

### 11.9.3 Requirements for CSA

PBS supports CSA for the Linux operating system on SGI Altix and ICE and on Cray machines. On SGI Altix machines, CSA requires supported versions of ProPack, and CSA facilities. SGI Performance Suite 1 does not come with CSA included. CSA requires CSA support Linux kernel modules.

On the supported platforms, the PBS MOM is CSA-enabled. If CSA workload management and user job accounting are available, PBS can use them.

See the Release Notes for information on which versions of ProPack provide support for CSA with PBS.
11.9.4 Configuring MOM for CSA

CSA support is specified in the `pbs_accounting_workload_mgt` line in MOM’s Version 1 configuration file. CSA support is enabled by default; you must explicitly disable it if you want it disabled. If the `pbs_accounting_workload_mgt` line is absent, CSA is still enabled.

To disable CSA support, modify `$PBS_HOME/mom_priv/config`, by setting `pbs_accounting_workload_mgt` to `false`, `off`, or `0`.

To enable CSA support, either remove the `pbs_accounting_workload_mgt` line, or set it to `true`, `on`, or `1`.

After modifying the MOM config file, either restart `pbs_mom` or send it SIGHUP.

11.9.5 Enabling CSA Support

In order for CSA user job accounting and workload management accounting requests to be acted on by the kernel, the administrator needs to make sure that the parameters `CSA_START` and `WKMG_START` in the `/etc/csa.conf` configuration file are set to "on" and that the system reflects this. You can check this by running the command:

```bash
csaswitch -c status
```

To set `CSA_START` to `on`, use the command:

```bash
csaswitch -c on -n csa
```

To set `WKMG_START` to `on`, use:

```bash
csaswitch -c on -n wkmg
```

Alternatively, you can use the CSA startup script `/etc/init.d/csa` with the desired argument (`on/off`) - see the system's manpage for csaswitch and how it is used in the `/etc/init.d/csa` startup script.
11.10 Systems with Hyper-Threaded Technology

On Linux machines that have Hyper-Threading Technology, PBS can end up reporting and using the number of logical processors, instead of the number of physical CPUs, as the value for `resources_available.ncpus`.

PBS does not control how CPUs are allocated to processes within a job. That is handled by the OS kernel.

11.10.1 Linux Machines with HTT

On Linux, PBS uses the number of CPUs shown in `/proc/cpuinfo`. If the CPUs are hyper-threaded and hyper-threading is enabled, the number of virtual and physical CPUs will be different.

11.10.2 Windows Machines with HTT

On Windows, PBS calls the `CPUCount` Windows function, which reports whether hyper-threading is enabled. If hyper-threading is enabled, MOM uses the number of physical CPUs. If hyper-threading is not enabled, MOM uses the number of CPUs reported by the OS. MOM logs whether or not hyper-threading is enabled.

11.10.3 Using Number of Physical CPUs

If you do not wish to use hyper-threading, you can configure PBS to use the number of physical CPUs. Do this by setting `resources_available.ncpus=<number of physical CPUs>`.
11.10.4 Hyperthreading Caveats

On a cpusetted system, NEVER change the value for resources_available.ncpus, resources_available.vmem, or resources_available.mem.
Chapter 12

Managing Jobs

This chapter covers information on managing PBS jobs.

12.1 Routing Jobs

You can route jobs to various places and by various criteria. You can reject submission of jobs that request too much of a given resource. You can force jobs into the correct queues. You can have all jobs submitted to a routing queue, then route them to the correct execution queues. You can use peer scheduling to have jobs executed at other PBS complexes. You can use hooks to move jobs. For information on routing jobs, see section 4.8.39 “Routing Jobs” on page 292.
12.2 Allocating Resources to Jobs

You can make sure that jobs request or inherit any resources required to manage those jobs. If a job does not request a resource, you can make sure that the resource is allocated to the job anyway.

In order for limits to be effective, each job must request each limited resource. For a complete description of how limits work, see section 5.15 “Managing Resource Usage” on page 409.

You can create custom resources specifically to allocate them to jobs. These resources can be visible, alterable, and requestable by users, or invisible, unalterable, and unrequestable, or visible but unalterable and unrequestable. For instructions on creating invisible or unrequestable resources, see section 5.14.2.5 “Resource Permission Flags” on page 362.

You can alter a job’s resource request using the following methods:

- You can set defaults for resources at the server or at each queue. This way, you can have jobs inherit specific values for the resources by routing them to special queues, where they inherit the defaults. For how jobs inherit resources, see section 5.9.4 “Allocating Default Resources to Jobs” on page 342. For how to specify default resources, see section 5.9.3 “Specifying Job Default Resources” on page 338. For how resource defaults change when a job is moved, see section 5.9.4.2 “Moving Jobs Between Queues or Servers Changes Defaults” on page 344.

- You can use a hook to assign a specific resource value to a job, if a job requests the wrong value for a resource. For how to use a hook to assign a resource to a job, see “Hooks” on page 461. For examples of using hooks to assign resources to jobs, see section 6.4.1.2 “Managing Job Resource Requests” on page 465.

- You can use the `qalter` command to change a job’s resource request. For how to use the qalter command, see “qalter” on page 152 of the PBS Professional Reference Guide.

- You can set default arguments the `qsub` command via the `default_qsub_arguments` server attribute. For how to use default arguments to `qsub`, see “Server Attributes” on page 365 of the PBS Professional Reference Guide.
12.3 Job Prologue and Epilogue

PBS provides a facility for the administrator to run a site-supplied script or program before and/or after each job runs. This allows initialization or cleanup of resources, such as temporary directories or scratch files. The script or program that runs before the job is the *prologue*; the one that runs after the job is the *epilogue*.

The primary purpose of the prologue is to provide a site with some means of performing checks prior to starting a job. The epilogue can be used to requeue a checkpointed job. See section 10.3.7.3 “Requeueing via Epilogue” on page 757.

Only one prologue and one epilogue may be used per PBS server. The same prologue and/or epilogue runs for every job in the complex.

Each script may be either a shell script or an executable object file.

12.3.1 When Prologue and Epilogue Run

The prologue runs before the job is executed. The epilogue runs after the job terminates for any reason, including normal termination, job deletion while running, error exit, or even if *pbs_mom* detects an error and cannot completely start the job. If the job is deleted while it is queued, then neither the prologue nor the epilogue is run.

If a prologue or epilogue script is not present, MOM continues in a normal manner.

12.3.2 Where Prologue and Epilogue Run

When multiple vnodes are allocated to a job, these scripts are run only by the MOM on the primary execution host.

The prologue runs with its current working directory set to `PBS_HOME/mom_priv`.

The epilogue runs with its current working directory set to the job's staging and execution directory. This is also where the job shell script is run.
12.3.3 Prologue and Epilogue Location

Both the prologue and the epilogue must reside in the 
PBS_HOME/mom_priv directory.

12.3.4 Prologue and Epilogue Requirements

In order to be run, the script must adhere to the following rules:

- The script must be in the PBS_HOME/mom_priv directory
- The prologue must have the exact name “prologue” under UNIX/Linux, or “prologue.bat” under Windows
- The epilogue must have the exact name “epilogue” under UNIX/Linux, or “epilogue.bat” under Windows
- The script must be written to exit with one of the zero or positive exit values listed in section 12.3.12 “Prologue and Epilogue Exit Codes” on page 848. The negative values are set by MOM
- Under UNIX/Linux, the script must be owned by root, be readable and executable by root, and cannot be writable by anyone but root
- Under Windows, the script’s permissions must give “Full Access” to the local Administrators group on the local computer

12.3.5 Prologue and Epilogue Environment Variables

The prologue and epilogue run with the following set in their environment:

- The contents of the pbs_environment file
- The PBS_JOBDIR environment variable
12.3.6 Prologue and Epilogue Permissions

Both the prologue and epilogue are run under root on UNIX/Linux, or under an Admin-type account on Windows, and neither is included in the job session.

12.3.7 Prologue and Epilogue Limitations and Caveats

- The prologue does not have access to the $PBS_NODEFILE environment variable.
- The prologue cannot be used to modify the job environment or to change limits on the job.

12.3.8 Prologue and Epilogue Arguments

The prologue is called with the following arguments:

Table 12-1: Arguments to Prologue

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv[1]</td>
<td>Job ID</td>
</tr>
<tr>
<td>argv[2]</td>
<td>User name under which the job executes</td>
</tr>
<tr>
<td>argv[3]</td>
<td>Group name under which the job executes</td>
</tr>
</tbody>
</table>

The epilogue is called with the following arguments:

Table 12-2: Arguments to Epilogue

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv[1]</td>
<td>Job ID</td>
</tr>
<tr>
<td>argv[2]</td>
<td>User name under which the job executes</td>
</tr>
<tr>
<td>argv[3]</td>
<td>Group name under which the job executes</td>
</tr>
</tbody>
</table>
12.3.8.1 Epilogue Argument Caveats

Under Windows and with some UNIX shells, accessing argv[10] in the epilogue requires a shift in positional parameters. To do this, the script must do the following:

1. Call the arguments with indices 0 through 9
2. Perform a shift /8
3. Access the last argument using %9%

### Table 12-2: Arguments to Epilogue

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv[4]</td>
<td>Job name</td>
</tr>
<tr>
<td>argv[5]</td>
<td>Session ID</td>
</tr>
<tr>
<td>argv[6]</td>
<td>Requested resources (job’s Resource_List)</td>
</tr>
<tr>
<td>argv[7]</td>
<td>List of resources used (job’s resources_used)</td>
</tr>
<tr>
<td>argv[8]</td>
<td>Name of the queue in which the job resides</td>
</tr>
<tr>
<td>argv[9]</td>
<td>Account string, if one exists</td>
</tr>
<tr>
<td>argv[10]</td>
<td>Exit status of the job</td>
</tr>
</tbody>
</table>
For example:

```bash
  cat epilogue
  > #!/bin/bash
  >
  > > echo "argv[0] = $0" > /tmp/epiargs
  > > echo "argv[1] = $1" >> /tmp/epiargs
  > > echo "argv[2] = $2" >> /tmp/epiargs
  > > echo "argv[3] = $3" >> /tmp/epiargs
  > > echo "argv[4] = $4" >> /tmp/epiargs
  > > echo "argv[5] = $5" >> /tmp/epiargs
  > > echo "argv[6] = $6" >> /tmp/epiargs
  > > echo "argv[7] = $7" >> /tmp/epiargs
  > > echo "argv[8] = $8" >> /tmp/epiargs
  > > echo "argv[9] = $9" >> /tmp/epiargs
  > shift
  > > echo "argv[10] = $9" >> /tmp/epiargs
```

12.3.9 Standard Input to Prologue and Epilogue

Both scripts have standard input connected to a system-dependent file. The default for this file is `/dev/null`.

12.3.10 Standard Output and Error for Prologue and Epilogue

The standard output and standard error of the scripts are connected to the files which contain the standard output and error of the job. Under UNIX/Linux, there is one exception: if a job is an interactive PBS job, the standard output and error of the epilogue is pointed to `/dev/null` because the pseudo-terminal connection used was released by the system when the job terminated. Interactive jobs are supported on UNIX/Linux only.
12.3.11 Prologue and Epilogue Timeout

When the scheduler runs a job, it waits until the prologue has ended. To prevent an error condition within the prologue or epilogue from delaying PBS, MOM places an alarm around the script’s/program’s execution. The default value is 30 seconds. If the alarm timeout is reached before the script has terminated, MOM will kill the script. The alarm value can be changed via the $prologalarm MOM configuration parameter. See section 10.4.6 “Prologue & Epilogue Running Time” on page 772.

12.3.12 Prologue and Epilogue Exit Codes

Normally, the prologue and epilogue programs should exit with a zero exit status. The prologue and epilogue should be written to exit with one of the zero or positive values listed here. When there is a problem with the script, MOM sets the exit value to one of the negative values. Exit status values and their impact on the job are listed in the following table:

Table 12-3: Prologue and Epilogue Exit Codes

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Meaning</th>
<th>Prologue</th>
<th>Epilogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>The script timed out (took too long).</td>
<td>The job will be requeued.</td>
<td>Ignored</td>
</tr>
<tr>
<td>-3</td>
<td>The wait(2) call waiting for the script to exit returned with an error.</td>
<td>The job will be requeued</td>
<td>Ignored</td>
</tr>
<tr>
<td>-2</td>
<td>The input file to be passed to the script could not be opened.</td>
<td>The job will be requeued.</td>
<td>Ignored</td>
</tr>
<tr>
<td>-1</td>
<td>The script has a permission error, is not owned by root, and/or is writable by others than root.</td>
<td>The job will be requeued.</td>
<td>Ignored</td>
</tr>
<tr>
<td>0</td>
<td>The script was successful.</td>
<td>The job will run.</td>
<td>Ignored</td>
</tr>
</tbody>
</table>
12.3.12.1 Logging Exit Status

MOM records in her log any case of a non-zero exit code, at event class 0x0001.

12.3.12.2 Epilogue Exit Status Caveats

Interactive-batch jobs cannot be requeued if the epilogue exits with a non-zero status. When this happens, these jobs are aborted.

12.3.13 Prologue Caveats

The administrator must exercise great caution in setting up the prologue to prevent jobs from being flushed from the system.

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Meaning</th>
<th>Prologue</th>
<th>Epilogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The script returned an exit value of 1.</td>
<td>The job will be aborted.</td>
<td>Ignored</td>
</tr>
<tr>
<td>&gt;1</td>
<td>The script returned a value greater than one.</td>
<td>The job will be requeued.</td>
<td>Ignored</td>
</tr>
<tr>
<td>2</td>
<td>The script returned a value of 2.</td>
<td>The job will be requeued.</td>
<td>If the job was checkpointed under the control of PBS, the job is requeued.</td>
</tr>
</tbody>
</table>

Table 12-3: Prologue and Epilogue Exit Codes
12.4 UNIX Shell Invocation

When PBS starts a job, it invokes the user’s login shell, unless the user submitted the job with the -S option. PBS passes the job script, which is a shell script, to the login process.

PBS passes the name of the job script to the shell program. This is equivalent to typing the script name as a command to an interactive shell. Since this is the only line passed to the script, standard input will be empty to any commands. This approach offers both advantages and disadvantages:

12.4.1 Advantages

• Any command which reads from standard input without redirection will get an EOF.
• The shell syntax can vary from script to script. It does not have to match the syntax for the user’s login shell. The first line of the script, even before any #PBS directives, should be

  `#!/shell`

  where `shell` is the full path to the shell of choice, `/bin/sh`, `/bin/csh`, ...

  The login shell will interpret the `#!` line and invoke that shell to process the script.

12.4.2 Disadvantages

• An extra shell process is run to process the job script.
• If the script does start with a `#!` line, the wrong shell may be used to interpret the script and thus produce errors.
• If a non-standard shell is used via the -S option, it will not receive the script, but its name, on its standard input.
12.5 Sequence of Events for Start of Job

This is the order in which events take place on an execution host at the start of a job:

1. Licenses are obtained
2. Any job-specific staging and execution directories are created:
   - PBS_JOBDIR and job’s jobdir attribute are set to pathname of staging and execution directory
   - Files are staged in
3. $TMPDIR is created
4. The job’s cpusets are created
5. The prologue is executed
6. The job script is executed
12.6 Sequence of Events for End of Job

This is the order in which events generally take place at the end of a job:

7. The job script finishes
8. The epilogue is run
9. The obit is sent to the server
10. Any specified file staging out takes place, including stdout and stderr
11. Files staged in or out are removed
12. Any job-specific staging and execution directories are removed
13. Job files are deleted
14. Application licenses are returned to pool
15. The job’s cpusets are destroyed

12.7 Job Termination

12.7.1 Normal Job Termination

When there is no $action terminate script and a running job is terminated, via the qdel <job ID> command, because of a server shutdown, or because the job has exceeded a limit, PBS waits for a configurable amount of time between sending a SIGTERM and a SIGKILL signal to the job. The amount of time is specified in the kill_delay queue attribute. The default value for this attribute is 10 seconds. PBS takes the following steps.
For a single-vnode job:

1. PBS sends the job a SIGTERM
2. PBS waits for the amount of time specified in the kill_delay queue attribute
3. PBS sends the job a SIGKILL

For a multi-vnode job:

1. Mother superior sends a SIGTERM to all processes on the primary execution host
2. If any of the processes of the top task of the job are still running, PBS waits a minimum of kill_delay seconds
3. Mother Superior sends a SIGKILL to all remaining job processes on the primary execution host
4. The subordinate MOMs send a SIGKILL to all their processes belonging to this job

### 12.7.2 Using the qdel Command to Terminate a Job

You can delete a job using the qdel command. See “qdel” on page 169 of the PBS Professional Reference Guide.

```
qdel <job ID>
```

If there is an $action terminate script, it is used to terminate the job.

If there is no $action terminate script, the SIGTERM-delay-SIGKILL sequence described in section 12.7.1 “Normal Job Termination” on page 852 is used to terminate the job.

This command does not terminate provisioning jobs.

```
qdel -Wforce <job ID>
```

If MOM is reachable, MOM sends the job a SIGKILL signal, and files are staged out. If MOM is unreachable, the
server discards the job. The job may or may not continue to run on the execution host(s).

This command terminates provisioning jobs.

### 12.7.3 Killing Job Processes

If you need to kill job processes, you can use the `printjob` command to find the job’s session ID, and then kill those processes. See “printjob” on page 149 of the PBS Professional Reference Guide.

### 12.7.4 Configuring Site-specific Job Termination

The default behavior of PBS is for MOM to terminate a job under the following circumstances:

- The job's usage of a resource exceeds the limit requested
- The job is deleted by the Server on shutdown
- The job is deleted via the `qdel` command

MOM normally uses `SIGTERM`, waits for the amount of time specified in the server’s `kill_delay` attribute, then issues a `SIGKILL`. See section 12.7 “Job Termination” on page 852.

You may want PBS to run your own job termination script in place of the normal action. The termination script is run in place of a `SIGTERM`. The termination script runs only on the primary execution host. After the top job process is terminated, a `KILL` signal is sent to any other job processes running on other hosts.

You can define the desired termination behavior by specifying the script you want to run in the `$action terminate` parameter in the Version 1 configuration file. The `$action terminate` parameter takes this form:

```
$action terminate <timeout> ! <path to script> [args]
```

Where

- `<timeout>` is the time, in seconds, allowed for the script to complete. A value of `zero (0)` indicates infinite time is allowed for the script to run.
<path to script> is the path to the script. If it is a relative path, it is evaluated relative to the PBS_HOME/mom_priv directory.

<args> are optional arguments to the script. Values for <args> may be any string not starting with a percent sign (“%”).

Arguments with a percent sign, making up any of the following keywords, are replaced by MOM with the corresponding value:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value Used by MOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>%jobid</td>
<td>Job ID</td>
</tr>
<tr>
<td>%sid</td>
<td>Session ID of task (job)</td>
</tr>
<tr>
<td>%uid</td>
<td>Execution UID of job</td>
</tr>
<tr>
<td>%gid</td>
<td>Execution GID of job</td>
</tr>
<tr>
<td>%login</td>
<td>Login name associated with UID</td>
</tr>
<tr>
<td>%owner</td>
<td>Job owner in form name@host</td>
</tr>
<tr>
<td>%auxid</td>
<td>Auxiliary ID (system-dependent)</td>
</tr>
</tbody>
</table>

### 12.7.4.1 Requirements for Termination Script

The script should exit with a value of zero when the job is terminated successfully. If the script exits successfully (with a zero exit status and before the time-out period), PBS does not send any signals or attempt to terminate the job. It is the responsibility of the termination script in this situation to ensure that the job has been terminated.

The script should exit with a non-zero value if the job was not successfully terminated. If the script exits with a non-zero exit status, the job is sent SIGKILL by PBS.

If the script does not complete in the time-out period, it is aborted and the job is sent SIGKILL.

### 12.7.4.2 Examples of Configuring Termination

UNIX:
Example 1: To use a 60-second timeout, run PBS_HOME/mom_priv/endjob.sh, and pass the job’s session ID, user ID, and PBS jobs ID to the script:

```
$action terminate 60 !endjob.sh %sid %uid %jobid
```

Example 2: To use an infinite timeout, run the system kill command with the signal 13, and pass the job’s session ID:

```
$action terminate 0 !/bin/kill -13 %sid
```

Windows:

Example 1: To use a 60-second timeout, run endjob.bat, and pass the job’s session ID, user ID, and PBS jobs ID to the script:

```
$action terminate 60 !endjob.bat %sid %uid %jobid
```

Example 2: To use an infinite timeout, run the pbskill command, and pass the job’s session ID:

```
$action terminate 0 !"C:/Program Files/PBS Pro/exec/bin/pbskill" %sid
```

12.7.4.3 Caveats and Restrictions on Termination

Under Windows, <path to script> must have a “.bat” suffix since it will be executed under the Windows command prompt cmd.exe. If the <path to script> specifies a full path, be sure to include the drive letter so that PBS can locate the file. For example, C:\winnt\temp\terminate.bat. The script must be writable by no one but an Administrator-type account.
### 12.8 Job Exit Codes

The exit value of a job may fall in one of three ranges, listed in the following table:

<table>
<thead>
<tr>
<th>Exit Code Range</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X &lt; 0$</td>
<td>The job could not be executed</td>
<td>See section 12.8.1 “Negative Job Exit Codes” on page 857</td>
</tr>
<tr>
<td>$0 \leq X &lt; 128$</td>
<td>Exit value of shell</td>
<td>See section 12.8.2 “Job Exit Codes Between 0 and 128 (or 256)” on page 858</td>
</tr>
<tr>
<td>$X \geq 128$</td>
<td>Job was killed with a signal</td>
<td>See section 12.8.3 “Job Exit Codes $\geq 128$ (or 256)” on page 859</td>
</tr>
</tbody>
</table>

#### 12.8.1 Negative Job Exit Codes

This is a PBS special return value indicating that the job could not be executed. These negative values are listed in the table below:

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>JOB_EXEC_OK</td>
<td>Job execution was successful</td>
</tr>
<tr>
<td>-1</td>
<td>JOB_EXEC_FAIL1</td>
<td>Job execution failed, before files, no retry</td>
</tr>
<tr>
<td>-2</td>
<td>JOB_EXEC_FAIL2</td>
<td>Job execution failed, after files, no retry</td>
</tr>
<tr>
<td>-3</td>
<td>JOB_EXEC_RETRY</td>
<td>Job execution failed, do retry</td>
</tr>
</tbody>
</table>
### 12.8.2 Job Exit Codes Between 0 and 128 (or 256)

This is the exit value of the top process in the job, typically the shell. This may be the exit value of the last command executed in the shell or the .logout script if the user has such a script (csh).

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>JOB_EXEC_INITABT</td>
<td>Job aborted on MOM initialization</td>
</tr>
<tr>
<td>-5</td>
<td>JOB_EXEC_INITRST</td>
<td>Job aborted on MOM initialization, checkpoint, no migrate</td>
</tr>
<tr>
<td>-6</td>
<td>JOB_EXEC_INITRMG</td>
<td>Job aborted on MOM initialization, checkpoint, ok migrate</td>
</tr>
<tr>
<td>-7</td>
<td>JOB_EXEC_BADRESRT</td>
<td>Job restart failed</td>
</tr>
<tr>
<td>-8</td>
<td>JOB_EXEC_GLOBUS_INIT_RETRY</td>
<td>Initialization of Globus job failed; do retry</td>
</tr>
<tr>
<td>-9</td>
<td>JOB_EXEC_GLOBUS_INIT_FAIL</td>
<td>Initialization of Globus job failed; no retry</td>
</tr>
<tr>
<td>-10</td>
<td>JOB_EXEC_FAILUID</td>
<td>Invalid UID/GID for job</td>
</tr>
<tr>
<td>-11</td>
<td>JOB_EXEC_RERUN</td>
<td>Job was rerun</td>
</tr>
<tr>
<td>-12</td>
<td>JOB_EXEC_CHKP</td>
<td>Job was checkpointed and killed</td>
</tr>
<tr>
<td>-13</td>
<td>JOB_EXEC_FAIL_PASSWORD</td>
<td>Job failed due to a bad password</td>
</tr>
<tr>
<td>-14</td>
<td>JOB_EXEC_RERUN_ON_SIS_FAIL</td>
<td>Job was requeued (if rerunnable) or deleted (if not) due to a communication failure between Mother Superior and a Sister</td>
</tr>
</tbody>
</table>

Table 12-6: Job Exit Codes
12.8.3  Job Exit Codes >= 128 (or 256)

This means the job was killed with a signal. The signal is given by X modulo 128 (or 256). For example an exit value of 137 means the job's top process was killed with signal 9 (137 % 128 = 9).

The exit status values greater than 128 (or 256) indicate which signal killed the job. Depending on the system, values greater than 128 (or on some systems 256; see wait(2) or waitpid(2) for more information), are the value of the signal that killed the job.

To interpret (or “decode”) the signal contained in the exit status value, subtract the base value from the exit status. For example, if a job had an exit status of 143, that indicates the job was killed via a SIGTERM (e.g. 143 - 128 = 15, signal 15 is SIGTERM). See the kill(1) manual page for a mapping of signal numbers to signal name on your operating system.

12.8.4  Logging Job Exit Codes

The exit status of jobs is recorded in the PBS Server logs and the accounting logs.

12.9  Rerunning or Requeueing a Job

You can re-run a job using the qrerun command. To re-run a job means to kill it, and requeue it in the execution queue from which it was run. See “qrerun” on page 207 of the PBS Professional Reference Guide.

12.9.1  Output from a Re-run Job

When you re-run a job, the job’s existing standard output and error files are copied back to the server host and stored in PBS_HOME/spool. They are then sent with the job to MOM when the job is again run. The output of a job that is re-run is appended to the output from prior runs of the same job.
12.9.2 Caveats for qrerun

- Jobs lose their queue wait time when they are requeued, including when they are checkpointed or requeued during preemption.
- The qrerun command and its API equivalent are not supported for jobs running on an HPCBP Server.

12.9.3 Requeueing Caveats

- When requeueing a job fails, for example because the queue does not exist, the job is deleted.

12.9.4 Caveats for Jobs Started by PBS

PBS attempts to run a job a certain number of times before placing a hold on the job. You cannot prevent a job from being held after this number of attempts. You must explicitly release the hold.

12.10 Job IDs

12.10.1 Format of Job IDs

Job Identifier

sequence_number.[server_name]@[server]

Job Array Identifier

Job array identifiers are a sequence number followed by square brackets:

sequence_number[].[server_name][@server]

Example:

1234[]}
Note that some shells require that you enclose a job array ID in double quotes.

12.10.2 Range of IDs

The largest possible job ID is the 7-digit number 9999999. After this has been reached, job IDs start again at zero.

12.10.3 Job IDs and Moving Jobs

If a job is moved from one server to another, the job’s ID does not change.

12.10.4 Job IDs and Requeueing and Checkpoint/Reraunt

If a job is requeued without being checkpointed, or checkpointed and requeued, it keeps its original job ID.

12.11 Where to Find Job Information

Information about jobs is found in PBS_HOME/server_priv/jobs and PBS_HOME/mom_priv/jobs.

12.11.1 Deleted Jobs

If PBS tries to requeue a job and cannot, for example when the queue doesn’t exist, the job is deleted.
12.11.2 Finding the cpuset Assigned to a Job

To find out which cpuset is assigned to a running job, the alt_id job attribute has a field called cpuset that will show this information. The cpusets are created with the name of the jobid for which they are created.

12.11.3 Failed Jobs

Once a job has experienced a certain number of failures, PBS holds the job.

12.11.4 Job Information When Server is Down

When the PBS server is down, you can use the pbs_dataservice command to start the PBS data service by hand, and then run the printjob command at the server host. See “pbs_dataservice” on page 54 of the PBS Professional Reference Guide and “printjob” on page 149 of the PBS Professional Reference Guide.

12.11.5 Job Information on Execution Host

You can use the printjob command to look at job information on the execution host. See “printjob” on page 149 of the PBS Professional Reference Guide.

12.12 The Job’s Node File

For each job, PBS creates a job-specific “host file” or “node file”, which is a text file containing the name(s) of the host(s) containing the vnode(s) allocated to that job. The file is created by the MOM on the primary execution host, and is available only on that host.
12.12.1 Node File Format and Contents

The node file contains a list of host names, one per line.

The order in which hosts appear in the PBS node file is the order in which chunks are specified in the selection directive.

Each hostname appears once per MPI process on that host, so that HostA appears $X$ times, and HostB appears $Y$ times, where $X$ is the number of MPI processes on HostA and $Y$ is the number of MPI processes on HostB.

Example 1: Two MPI processes run on HostA and one MPI process runs on HostB. The node file looks like this:

HostA
HostA
HostB

The number of MPI processes on each host is specified in the job’s `mpiprocs` resource request. For each chunk requesting `mpiprocs=M`, the name of the host from which that chunk is allocated is written in the node file $M$ times. Therefore the number of lines in the node file is the sum of requested `mpiprocs` for all chunks requested by the job.

The name of the host is the value in `resources_available.host` of the allocated vnode(s).

The number of OpenMP threads for a job is specified in the job’s `ompthreads` resource request. The `ompthreads` resource request sets the values of the `NCPUS` and `OMP_NUM_THREADS` environment variables for every PBS task, including the top PBS task. If a chunk requests `ncpus=N`, with $N > 1$, PBS will only create one MPI process for that chunk, but set the number of OpenMP threads to $N$.

See “Built-in Resources” on page 350 of the PBS Professional Reference Guide for definitions of the `mpiprocs` and `ompthreads` resources.

12.12.2 Name and Location of Node File

The file is created by the MOM on the primary execution host, in `PBS_HOME/aux/JOB_ID`, where `JOB_ID` is the job identifier for that job.
The full path and name for the node file is written to the job’s environment via the environment variable PBS_NODEFILE.

### 12.13 Job Directories

PBS jobs use two kinds of directories. The first is the job’s staging and execution directory. Files are staged into and out of this directory, and this is the directory where the job script executes.

The second is the job’s temporary directory, where the job can create scratch files if necessary. The location of this directory is specified in the $tmpdir MOM configuration parameter. This parameter is copied by MOM to the TMPDIR job environment variable. The job can then use this environment variable. See section 12.14.1 “Creation of TMPDIR” on page 869.

#### 12.13.1 The Job’s Staging and Execution Directories

A job’s staging and execution directory is the directory to which input files are staged, and from which output files are staged. It is also the current working directory for the job script, for tasks started via the `pbs_t( )` API, and for the epilogue.

Each PBS user may submit several jobs at once. Each job may need to have data files staged in or out. Each execution host needs a staging and execution directory for jobs. PBS can provide a job-specific staging and execution directory on each execution host for each job. The job’s sandbox attribute controls whether PBS creates a staging and execution directory for each job, or uses the user’s home directory for staging and execution.

PBS stages files to and from the primary execution host only. If the execution hosts use a shared file system, the staging and execution directory is available everywhere the job needs it.
When a job uses a job-specific staging and execution directory created by PBS, PBS does not require the job’s owner to have a home directory on the execution host(s), as long as each MOM’s $jobdir_root configuration option is set, and is set to something other than the user’s home directory.

Staging is specified via the job’s stagein and stageout attributes. The format is the following:

\[ \text{execution_path}@[\text{storage_host:}]\text{storage_path} \]

The execution_path is the path to the staging and execution directory. On stagein, storage_path is the path where the input files normally reside, and on stageout, storage_path is the path where output files will end up.

### 12.13.1.1 The Job’s sandbox Attribute

If the job’s sandbox attribute is set to PRIVATE, PBS creates a job-specific staging and execution directory for that job. If sandbox is unset, or is set to HOME, PBS uses the user’s home directory as the job’s staging and execution directory. Using the server’s default_qsub_arguments attribute, you can specify the default for the sandbox attribute for all jobs.

By default, the sandbox attribute is not set.

The user can set the sandbox attribute via qsub, for example:

```
qsub -Wsandbox=PRIVATE
```

The -Wsandbox option to qsub overrides default_qsub_arguments. The job’s sandbox attribute cannot be altered while the job is executing.

#### Table 12-7: Effect of Job’s sandbox Attribute on Location of Staging and Execution Directory

<table>
<thead>
<tr>
<th>Job’s sandbox attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>not set</td>
<td>Job’s staging and execution directory is the user’s home directory</td>
</tr>
<tr>
<td>HOME</td>
<td>Job’s staging and execution directory is the user’s home directory</td>
</tr>
</tbody>
</table>
Managing Jobs

12.13.1.2 Options, Attributes and Environment Variables Affecting Staging

The environment variable PBS_JOBDIR is set to the pathname of the staging and execution directory on the primary execution host. PBS_JOBDIR is added to the job script process, any job tasks created by the pbs_tm() API, the prologue and epilogue, and the MOM $action scripts.

The job’s jobdir attribute is read-only, and is also set to the pathname of the staging and execution directory on the primary execution host. The jobdir attribute can be viewed using the -f option to qstat.

<table>
<thead>
<tr>
<th>Job’s sandbox attribute</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIVATE</td>
<td>Job’s staging and execution directory is created under the directory specified in MOM $jobdir_root configuration option. If $jobdir_root is unset, the staging and execution directory is created under the user’s home directory.</td>
</tr>
</tbody>
</table>
The following table lists the options, attributes, etc., affecting staging:

**Table 12-8: Options, Attributes, Environment Variables, etc., Affecting Staging**

<table>
<thead>
<tr>
<th>Option, Attribute, Environment Variable, etc.</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MOM’s $jobdir_root option</strong></td>
<td>Directory under which PBS creates job-specific staging and execution directories. Defaults to user’s home directory if unset. If $jobdir_root is unset, the user’s home directory must exist. If $jobdir_root does not exist when MOM starts, MOM will abort. If $jobdir_root does not exist when MOM tries to run a job, MOM will kill the job. Permissions on the directory specified in this option must be 1777.</td>
</tr>
<tr>
<td><strong>MOM’s $usecp option</strong></td>
<td>Tells MOM where to look for files in a shared file system; also tells MOM that she can use the local copy agent.</td>
</tr>
<tr>
<td><strong>Job’s sandbox attribute</strong></td>
<td>Determines which directory PBS uses for the job's staging and execution. If value is PRIVATE, PBS uses a job-specific directory it creates under the location specified in the MOM $jobdir_root configuration option. If value is HOME or is unset, PBS uses the user's home directory for staging and execution. User-settable per-job via qsub -W or through a PBS directive. See the pbs_mom.8B man page.</td>
</tr>
<tr>
<td><strong>Job’s stagein attribute</strong></td>
<td>Sets list of files or directories to be staged in. User-settable per job via qsub -W.</td>
</tr>
<tr>
<td><strong>Job’s stageout attribute</strong></td>
<td>Sets list of files or directories to be staged out. User-settable per job via qsub -W.</td>
</tr>
</tbody>
</table>
### Table 12-8: Options, Attributes, Environment Variables, etc., Affecting Staging

<table>
<thead>
<tr>
<th>Option, Attribute, Environment Variable, etc.</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job’s jobdir attribute</td>
<td>Set to pathname of staging and execution directory on primary execution host. Read-only; viewable via <code>qstat -f</code>.</td>
</tr>
<tr>
<td>Job’s Keep_Files attribute</td>
<td>Determines whether output and/or error files remain on execution host. User-settable per job via <code>qsub -k</code> or through a PBS directive. If the Keep_Files attribute is set to <code>o</code> and/or <code>e</code> (output and/or error files remain in the staging and execution directory) and the job’s sandbox attribute is set to <code>PRIVATE</code>, standard output and/or error files are removed when the staging directory is removed at job end along with its contents.</td>
</tr>
<tr>
<td>Job’s PBS_JOBDIR environment variable</td>
<td>Set to pathname of staging and execution directory on primary execution host. Added to environments of job script process, <code>pbs_tm</code> job tasks, prologue and epilogue, and MOM <code>$action</code> scripts.</td>
</tr>
<tr>
<td>Job’s TMPDIR environment variable</td>
<td>Location of job-specific scratch directory.</td>
</tr>
<tr>
<td>PBS_RCP string in <code>pbs.conf</code></td>
<td>Location of <code>rcp</code> command</td>
</tr>
<tr>
<td>PBS_SCP string in <code>pbs.conf</code></td>
<td>Location of <code>scp</code> command; setting this parameter causes PBS to first try <code>scp</code> rather than <code>rcp</code> for file transport.</td>
</tr>
<tr>
<td>Server’s default_qsub_arguments attribute</td>
<td>Can contain a default for job’s sandbox (and other) attributes.</td>
</tr>
</tbody>
</table>
12.13.1.3 Getting Information About the Job’s Staging and Execution Directory

The job’s jobdir attribute is viewable via qstat or the equivalent API while a job is executing. The value of jobdir is not retained if a job is rerun; it is undefined whether jobdir is visible or not when the job is not executing.

12.13.1.4 Example of Setting Location for Creation of Staging and Execution Directories

To make it so that jobs with sandbox=PRIVATE have their staging and execution directories created under /scratch, as /scratch/<job-specific_dir_name>, put the following line in MOM’s configuration file:

$jobdir_root /scratch

12.13.1.5 Staging and Execution Directory Caveats

If the user home directory is NFS mounted, and you want to use sandbox=PRIVATE, then root must be allowed write privilege on the NFS filesystem on which the users' home directories reside.

The directory specified in MOM’s $jobdir_root parameter must have permissions set to 1777.

12.14 The Job’s Lifecycle

12.14.1 Creation of TMPDIR

For each host allocated to the job, PBS creates a job-specific temporary scratch directory for this job. The location of TMPDIR is set by MOM to the value of MOM’s $tmpdir configuration option, which is the pathname of the job-specific temporary scratch directory. This directory is for the use of the job, not PBS. This directory and its contents are removed when the job is finished.
Chapter 12  Managing Jobs

The recommended TMPDIR configuration is to have a separate, local directory on each host. If the temporary scratch directory cannot be created, the job is killed.

12.14.2 Choice of Staging and Execution Directories

If the job’s sandbox attribute is set to PRIVATE, PBS creates job-specific staging and execution directories for the job. If the job’s sandbox attribute is set to HOME, or is unset, PBS uses the user’s home directory for staging and execution. The staging and execution directory may be shared (e.g., cross-mounted) among all the hosts allocated to the job, or each host may use a separate directory. This is true whether or not the directory is the user’s home directory.

12.14.2.1 Choosing Job-specific Staging and Execution Directories

When PBS creates a job-specific staging and execution directory, it does so under the directory specified in the MOM configuration option $jobdir_root. If the $jobdir_root option is not set, job-specific staging and execution directories are created under the user’s home directory.

If the staging and execution directory is accessible on all of the job’s execution hosts, these hosts will log the following message at the 0x0400 event class:

"the staging and execution directory <full path> already exists".

If the staging and execution directory is not cross-mounted so that it is accessible on all the job’s execution hosts, each secondary host also creates a directory using the same base name as was used on the primary host.

If the staging and execution directory cannot be created the job is aborted. The following error message is logged at 0x0001 event class:

"unable to create the job directory <full path>".
When PBS creates a directory, the following message is logged at 0x0008 event class:

“created the job directory <full path>”

### 12.14.2.1.1 Job-specific Staging and Execution Directory Caveats

- You should not depend on any particular naming scheme for the new directories that PBS creates for staging and execution. The pathname to each directory on each node may be different, since each depends on the corresponding MOM's $jobdir_root.
- The permissions for the directory specified in $jobdir_root must be 1777.

### 12.14.2.2 Choosing User’s Home Directory as Staging and Execution Directory

If the job’s sandbox attribute is unset or is set to HOME, PBS uses the user’s home directory for the job’s staging and execution directory.

The user must have a home directory on each execution host. The absence of the user's home directory is an error and causes the job to be aborted.

### 12.14.3 Setting PBS_JOBDIR and the Job’s jobdir Attribute

PBS sets PBS_JOBDIR and the job’s jobdir attribute to the pathname of the staging and execution directory.

### 12.14.4 Staging Files Into Staging and Execution Directories

PBS evaluates execution_path and storage_path relative to the staging and execution directory given in PBS_JOBDIR, whether this directory is the user’s home directory or a job-specific directory created by PBS. PBS stages files to the primary execution host only. Staging is done as the job owner.
12.14.5 Running the Prologue

The MOM’s prologue is run on the primary host as root, with the current working directory set to PBS_HOME/mom_priv and with PBS_JOBDIR set in its environment.

12.14.6 Job Execution

PBS runs the job script on the primary host as the user. PBS also runs any tasks created by the job via the pbs_tm() API as the user. The job script and tasks are executed with their current working directory set to the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in their environment. The job attribute jobdir is set to the pathname of the staging and execution directory on the primary host.

12.14.7 Standard Out, Standard Error and TMPDIRs

The job's stdout and stderr files are created directly in the job's staging and execution directory on the primary execution host.

12.14.7.1 Output and Error with Job-specific Staging and Execution Directories

If the qsub -k option is used, the stdout and stderr files will not be automatically copied out of the staging and execution directory at job end; they will be deleted when the directory is automatically removed.


If the -k option to qsub is used, standard out and/or standard error files are retained on the primary execution host instead of being returned to the submission host, and are not deleted after job end.
12.14.8 Running the Epilogue

PBS runs MOM's epilogue script on the primary host as root. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with PBS_JOBDIR set in its environment.

12.14.9 Staging Files Out and Removing Execution Directory

When PBS stages files out, it evaluates execution_path and storage_path relative to PBS_JOBDIR. Files that cannot be staged out are saved in PBS_HOME/undelivered. See section 14.5.6 “Non-delivery of Output” on page 912. PBS stages files out from the primary execution host only. Staging is done as the job owner.

When the job is done, PBS writes the final job accounting record and purges job information from the Server’s database.

12.14.9.1 Staging Out with Job-specific Staging and Execution Directories

If PBS created job-specific staging and execution directories for the job, it cleans up at the end of the job. If no errors are encountered during stageout and all stageouts are successful, the staging and execution directory and all of its contents are removed, on all execution hosts.

Files to be staged out are deleted all together, only after successful stageout of all files. If any errors are encountered during stageout, no files are deleted on the primary execution host, and the execution directory is not removed.

If PBS created job-specific staging and execution directories on secondary execution hosts, those directories and their contents are removed at the end of the job, regardless of stageout errors.

Files that are successfully staged out are deleted immediately, without regard to files that were not successfully staged out.

12.14.10 Removing TMPDIRs

PBS removes all TMPDIRs, along with their contents.

12.15 Managing Job History

12.15.1 Introduction

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used.

PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server.

12.15.2 Definitions

Moved jobs
Jobs which were moved to another server
Finished jobs

Jobs whose execution is done, for any reason:
- Jobs which finished execution successfully and exited
- Jobs terminated by PBS while running
- Jobs whose execution failed because of system or network failure
- Jobs which were deleted before they could start execution

History jobs

Jobs which will no longer execute at this server:
- Moved jobs
- Finished jobs

12.15.3 Job History Information

PBS can keep all job attribute information, including the following kinds of job history information:

- Submission parameters
- Whether the job started execution
- Whether execution succeeded
- Whether staging out of results succeeded
- Which resources were used

PBS keeps job history for the following jobs:

- Jobs that have finished execution
- Jobs that were deleted
- Jobs that were moved to another server

You can configure whether PBS preserves job history, and for how long. The job history for finished and moved jobs is preserved and available for the specified duration. After the duration has expired, PBS deletes the job history information and it is no longer available.
12.15.4 Configuring Job History Management

To configure job history, you enable it and you set the job history duration. You configure PBS to manage job history using the following server attributes:

job_history_enable
   Enables or disables job history management. Setting this attribute to True enables job history management.
   Format: Boolean.
   Default: False

job_history_duration
   Specifies the length of time that PBS will keep each job’s history.
   Format: duration: [[hours:]minutes:]seconds[.milliseconds]
   Default: Two weeks (336:00:00)

12.15.4.1 Enabling Job History

To enable job history management, set the server’s job_history_enable attribute to True:

   Qmgr: set server job_history_enable=True

12.15.4.2 Setting Job History Duration

To set the length of time that job history is preserved, set the server’s job_history_duration attribute to the desired duration:

   Qmgr: set server job_history_duration=<duration>

If the job history duration is set to zero, no history is preserved.

If job history is enabled and job history duration is unset, job history information is kept for the default 2 weeks.
12.15.5 Changing Job History Settings

12.15.5.1 Disabling Job History

If job history is being preserved, and you unset the `job_history_enable` server attribute, PBS deletes all job history information. This information is no longer available.

12.15.5.2 Enabling Job History

If job history is not being preserved, and you set the `job_history_enable` server attribute, PBS begins preserving job history information for any jobs that are queued or running.

12.15.5.3 Modifying Job History Duration

Every job’s history duration is set to the current value of the `job_history_duration` server attribute.

Example 1: Reducing job history duration:

The value of `job_history_duration` was “00:10:00” when a job finished execution. After 2 minutes, you change the duration to “00:06:00”. This job’s history is kept for a total of 6 minutes.

Example 2: Increasing job history duration:

The value of `job_history_duration` was “00:10:00” when a job finished execution. After 8 minutes you change the duration to “00:30:00”. This job’s history is kept for a total of 30 minutes.

Example 3: Increasing job history duration:

The value of `job_history_duration` was “00:10:00” when a job finished execution. After 11 minutes you change the duration to “00:30:00”. This job’s history is kept for a total of 10 minutes. The job’s history is deleted after it is kept for 10 minutes.
12.15.6 Backward Compatibility

To have PBS behave as it did before the job history management feature was introduced, disable job history management. Do one of the following:

- Set the server’s job_history_enable attribute to False:
  
  ```
  Qmgr: set server job_history_enable=False
  ```

- Unset the server’s job_history_enable attribute:
  
  ```
  Qmgr: unset server job_history_enable
  ```

- Set the value of the server’s job_history_duration attribute to zero, by doing one of the following:
  
  ```
  Qmgr: set server job_history_duration=0
  Qmgr: set server job_history_duration=00:00
  Qmgr: set server job_history_duration=00:00:00
  ```

12.15.7 Logging Moved Jobs

Jobs can be moved to another server for one of the following reasons:

- Moved for peer scheduling
- Moved via the qmove command
- Job was submitted to a routing queue, then routed to a destination queue at another server

When a job is moved, the server logs the event in the server log and the accounting log. The server log messages are logged at log event class 0x0008.

Format for the server log file:

```
7/08/2008 16:17:38;0008;Server@serverhost1;Job;97.serverhost1.domain.com;Job moved to destination:workq@serverhost2
```

Format for the accounting log entry:

```
7/08/2008 16:17:38;M;97.serverhost1.domain.com;destination=workq@serverhost2
```
Record type: $M$ (moved job)

### 12.15.8 Job History Caveats

- Enabling job history requires additional memory for the server. When the server is keeping job history, it needs 8kb-12kb of memory per job, instead of the 5kb it needs without job history. Make sure you have enough memory: multiply the number of jobs being tracked by this much memory. For example, if you are starting 100 jobs per day, and tracking history for two weeks, you’re tracking 1400 jobs at a time. On average, this will require 14.3M of memory.

- If the server is shut down abruptly, there is no loss of job information. However, the server will require longer to start up when keeping job history, because it must read in more information.

### 12.16 Environment Variables

The settings in `$_PBS_HOME/pbs_environment` are available to user job scripts. You must HUP the MOM if you change the file. This file is useful for setting environment variables for `mpirun` etc.
Chapter 13

Administration

This chapter covers information on the maintenance and administration of PBS, and is intended for the PBS administrator. Topics covered include starting and stopping PBS, event logging, and accounting.

13.1 The PBS Configuration File

During the installation of PBS Professional, the installation script creates a configuration file named `pbs.conf`. This configuration file controls which daemons are to run on the local system, the directory tree location, and various runtime configuration options. Each host in a complex should have its own `pbs.conf` file.
13.1.1 Location of Configuration File

The configuration file is located in one of the following:

UNIX/Linux:

/etc/pbs.conf

Windows:

[PBS Destination Folder]/pbs.conf

where [PBS Destination Folder] is the path specified when PBS is installed on the Windows platform, for example:

C:/Program Files/PBS Pro/pbs.conf

or

C:/Program Files (x86)/PBS Pro/pbs.conf

13.1.2 Example of Configuration File

The following is an example of a pbs.conf file for a host which is to run the server, the scheduler, and a MOM. The server runs on the host named Host1.ExampleDomain.

PBS_EXEC=/opt/pbs/M.N.P.S
PBS_HOME=/var/spool/PBS
PBS_START_SERVER=1
PBS_START_MOM=1
PBS_START_SCHED=1
PBS_SERVER=Host1.ExampleDomain
13.1.3 Contents of Configuration File

The following table describes the available parameters for the `pbs.conf` configuration file:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_BATCH_SERVICE_PORT</td>
<td>Port Server listens on</td>
</tr>
<tr>
<td>PBS_BATCH_SERVICE_PORT_DIS</td>
<td>DIS port server listens on</td>
</tr>
<tr>
<td>PBS_DATA_SERVICE_PORT</td>
<td>Used to specify non-default port for connecting to data service.</td>
</tr>
<tr>
<td>PBS_ENVIRONMENT</td>
<td>Location of <code>pbs_environment</code> file</td>
</tr>
<tr>
<td>PBS_EXEC</td>
<td>Location of PBS <code>bin</code> and <code>sbin</code> directories</td>
</tr>
<tr>
<td>PBS_HOME</td>
<td>Location of PBS working directories</td>
</tr>
<tr>
<td>PBS_LICENSE_FILE_LOCATION</td>
<td>Location of license server.</td>
</tr>
<tr>
<td>PBS_LOCALLOG</td>
<td>Enables logging to local PBS log files</td>
</tr>
<tr>
<td>PBS_MANAGER_GLOBUS_SERVICE_PORT</td>
<td>Port Globus MOM listens on</td>
</tr>
<tr>
<td>PBS_MANAGER_SERVICE_PORT</td>
<td>Port MOM listens on</td>
</tr>
<tr>
<td>PBS_MOM_GLOBUS_SERVICE_PORT</td>
<td>Port Globus MOM listens on</td>
</tr>
<tr>
<td>PBS_MOM_HOME</td>
<td>Location of MOM working directories</td>
</tr>
<tr>
<td>PBS_MOM_SERVICE_PORT</td>
<td>Port MOM listens on</td>
</tr>
</tbody>
</table>
For information on how to use the pbs.conf file when configuring PBS for failover, see section 10.2.5.2 “Configuring the pbs.conf File” on page 719.

### 13.1.4 Configuration File Caveats and Recommendations

If you specify a location for PBS_HOME in the shell environment, make sure that this agrees with that specified in pbs.conf.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_PRIMARY</td>
<td>Hostname of primary Server</td>
</tr>
<tr>
<td>PBS_RCP</td>
<td>Location of rcp command if rcp is used</td>
</tr>
<tr>
<td>PBS_SCHEDULER_SERVICE_PORT</td>
<td>Port Scheduler listens on</td>
</tr>
<tr>
<td>PBS_SCP</td>
<td>Location of scp command if scp is used; setting this parameter causes PBS to first try scp rather than rcp for file transport.</td>
</tr>
<tr>
<td>PBS_SECONDARY</td>
<td>Hostname of secondary Server</td>
</tr>
<tr>
<td>PBS_SERVER</td>
<td>Hostname of host running the Server</td>
</tr>
<tr>
<td>PBS_START_MOM</td>
<td>Set to 1 if a MOM is to run on this vnode</td>
</tr>
<tr>
<td>PBS_START_SCHED</td>
<td>Set to 1 if Scheduler is to run on this vnode</td>
</tr>
<tr>
<td>PBS_START_SERVER</td>
<td>Set to 1 if Server is to run on this vnode</td>
</tr>
<tr>
<td>PBS_SYSLOG</td>
<td>Controls use of syslog facility</td>
</tr>
<tr>
<td>PBS_SYSLOGSEVR</td>
<td>Filters syslog messages by severity</td>
</tr>
</tbody>
</table>
13.2 Environment Variables

PBS sets environment variables for different purposes: some variables are used by the daemons, commands, and jobs, and some environment variables are set individually for each job.

13.2.1 Environment Variables For Daemons, Commands, and Jobs

The PBS installer creates an environment file called `pbs_environment`. This file is used by the daemons, commands, and jobs:

- Each PBS daemon initializes its environment using this environment file.
- Several commands use environment variables to determine things like the name of the default server. The environment file is useful for setting environment variables for `mpirun`, etc.
- Jobs inherit the contents of this environment file before they acquire settings from `.profile` and `.login` files. Job scripts can use the environment variables set in the job’s environment.

You can edit the environment file.

13.2.1.1 Contents of Environment File

When this file is created, it contains the following:

```
TZ=<local timezone, e.g. US/Pacific>
PATH=/bin:/usr/bin
```

For a list of PBS environment variables, see “PBS Environment Variables” on page 517 of the PBS Professional Reference Guide.
13.2.1.2 Location of Environment File

The PBS environment file is located here:

PBS_HOME/pbs_environment

13.2.1.3 Environment File Requirements

You must restart each daemon after making any changes to the environment file.

13.2.2 Job-specific Environment Variables

For each job, the qsub command creates environment variables beginning with PBS_O_, and puts them in the job’s environment. They are not written to pbs_environment. The server sets some of these environment variables if the qsub command does not set them.

For each job, the MOM on the primary execution host creates a file of the hosts to be used by the job. The node file is put in the job’s environment, but the host list is not written to pbs_environment. The location of the node file is specified in the PBS_NODEFILE environment variable, which is set for the job only. See section 12.12 “The Job’s Node File” on page 862.

Some environment variables are set by commands. The PBS mpiexec script sets PBS_CPUSET_DEDICATED.

For a list of environment variables used and set by the qsub command, see “Environment Variables” on page 265 of the PBS Professional Reference Guide.

13.3 The Accounting Log

The PBS Server maintains an accounting log. This file is maintained on the Server host only; it is not written on the execution hosts. The log name defaults to PBS_HOME/server_priv/accounting/ccyymmdd where ccyymmdd is the date. The accounting log files may be placed else-
where by specifying the -A option on the pbs_server command line. The option argument is the full (absolute) path name of the file to be used. If a null string is given, then the accounting log will not be opened and no accounting records will be recorded. For example

```
pbs_server -A ""
```

The accounting file is changed according to the same rules as the event log files. If the default file is used, named for the date, the file will be closed and a new one opened every day on the first event (write to the file) after midnight. With either the default file or a file named with the -A option, the Server will close the accounting log upon daemon/service shutdown and reopen it upon daemon/service startup.

On UNIX the Server will also close and reopen the account log file upon the receipt of a SIGHUP signal. This allows you to rename the old log and start recording again on an empty file. For example, if the current date is February 9, 2005 the Server will be writing in the file 20050209. The following actions will cause the current accounting file to be renamed feb9 and the Server to close the file and start writing a new 20050209:

```
   cd $PBS_HOME/server_priv/accounting
   mv 20050209 feb9
   kill -HUP 1234     (the Server’s pid)
```

On Windows, to manually rotate the account log file, shut down the Server, move or rename the accounting file, and restart the Server. For example, to cause the current accounting file to be renamed feb9 and the Server to close the file and start writing a new 20050209:

```
   cd "%PBS_HOME%\server_priv\accounting"
   net stop pbs_server
   move 20050209 feb9
   net start pbs_server
```

### 13.3.1 How To Find Accounting Information

Accounting logs are written on the Server host only.
13.3.1.1 All Log Files

To get information about a job that is running or has finished, use the `tracejob` command at the Server host and any execution hosts on which the job ran. The `tracejob` command looks at all log files.

```
tracejob <job ID>
```

See “tracejob” on page 271 of the PBS Professional Reference Guide, or the `tracejob(8B)` man page for details about using the `tracejob` command.

13.3.1.2 Server and Accounting Log Files

To get information about a job that has finished, use the `pbs-report` command at the Server host. The `pbs-report` command looks at Server logs and accounting logs.

```
pbs-report [options]
```

See “pbs-report” on page 38 of the PBS Professional Reference Guide or the `pbs-report(8B)` man page for details about using the `pbs-report` command.

13.3.2 Accounting Log Contents

The PBS accounting file is a text file with each entry terminated by a newline. There is no limit to the size of an entry.

13.3.2.1 Log Entry Format


13.3.2.2 Record Types

13.3.2.3 Logging Resource Usage

MOM periodically polls jobs for usage by the jobs running on her host, collects the results, and reports this to the server. When a job exits, she polls again to get the final tally of usage for that job.

For example, MOM polls the running jobs at times T1, T2, T4, T8, T16, T24, and so on.

The output shown by a `qstat` during the window of time between T8 and T16 shows the resource usage up to T8.

If the `qstat` is done at T17, the output shows usage up through T16. If the job ends at T20, the accounting log (and the final log message, and the email to the user if "`qsub -me`" was used in job submission) contains usage through T20.

The final report does not include the epilogue. The time required for the epilogue is treated as system overhead.

13.3.3 PBS Accounting and Windows

PBS will save information such as user name, group name, and account name in the accounting logs found in `PBS_HOME\server_priv\accounting`. Under Windows, these saved entities can contain space characters, thus PBS will put a quote around string values containing spaces. For example,

```
user=pbstest group=None account="Power Users"
```
Otherwise, one can specify the replacement for the space character by adding the -s option to the pbs_server command line option. This can be set as follows:

2. Select PBS_SERVER.
3. Stop the Server
4. Specify in start parameters the option for example “-s %20”.
5. Start the Server

This will replace space characters as “%20” in user=, group=, account= entries in accounting log file:

\[
\text{user=pbstest group=None account=Power%20Users}
\]

Note: If the first character of the replacement string argument to -s option appears in the input string itself, then that character will be replaced by its hex representation prefixed by %. For example, given:

\[
\text{account=Po%wer Users}
\]

Since % also appears the above entry and our replacement string is “%20”, then replace this % with its hex representation (%25):

\[
\text{account="Po%25wer%20Users"}
\]

### 13.4 Event Logging

PBS provides event logging for the Server, the Scheduler, and each MOM. You can use logfiles to monitor activity in the PBS complex.

#### 13.4.1 PBS Events

The amount and type of output in the PBS event logfiles depends on the specified log filters for each component. All three PBS components can be directed to record only messages pertaining to certain event types, called
event classes. The specified event classes are logically “or-ed” to produce a mask representing the events to be logged by the Server or MOM, or the events to be excluded by the scheduler. The hexadecimal value for each event class is shown in section 13.4.3 “Log Event Classes” on page 892. When events appear in the log file, they are tagged with their hexadecimal value, without a preceding “0x”.

### 13.4.2 Event Logfiles

Each PBS daemon writes a separate event logfile. By default, each daemon writes a file that has the current date as its name in the PBS_HOME/<component>_logs directory. The location of the logfile can be overridden with the -L option to each daemon’s command. For example, to override the Server’s logfile location:

```bash
pbs_server -L <new path>
```

Each daemon closes the day’s log file and opens a new log file on the first message written after midnight. If no messages are written, the old log file stays open. Each daemon closes and reopens the same logfile when the daemon receives a SIGHUP.

Each daemon writes its version and build information to its event logfile each time it is started or restarted, and also when the logfile is automatically rotated out. The version and build information appear in individual records. These records contain the following substrings:

```plaintext
pbs_version = <PBSPro_stringX.stringY.stringZ.5-digit seq>
build = <status line from config.status, etc>
```

Example:

```plaintext
pbs_version = PBSPro_9.2.0.63106
build = '--set-cflags=-g -O0' --enable-security=KCRYPT ...
```

If the daemon cannot write to its log file, it writes the error message to the console. Some errors that appear before the daemon has backgrounded itself may appear on standard error.
The maximum number of characters in the message portion of a log entry is 4096.

13.4.3 Log Event Classes

PBS allows specification of the types of events that are logged for each daemon. Each type of log event has a different log event class. All daemons use the same log event class for the same type of event. The following table lists the log event class for each type of event.

Table 13-2: PBS Events and Log Event Classes

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0001</td>
<td>Internal PBS errors</td>
</tr>
<tr>
<td>2</td>
<td>0x0002</td>
<td>System (OS) errors, such as malloc failure</td>
</tr>
<tr>
<td>4</td>
<td>0x0004</td>
<td>Administrator-controlled events, such as changing queue attributes</td>
</tr>
<tr>
<td>8</td>
<td>0x0008</td>
<td>Job related events, e.g. submitted, ran, deleted</td>
</tr>
<tr>
<td>16</td>
<td>0x0010</td>
<td>Job resource usage</td>
</tr>
<tr>
<td>32</td>
<td>0x0020</td>
<td>Security related events</td>
</tr>
<tr>
<td>64</td>
<td>0x0040</td>
<td>When the Scheduler was called and why</td>
</tr>
<tr>
<td>128</td>
<td>0x0080</td>
<td>Common debug messages</td>
</tr>
<tr>
<td>256</td>
<td>0x0100</td>
<td>Debug event class 2</td>
</tr>
<tr>
<td>512</td>
<td>0x0200</td>
<td>Reservation-related messages</td>
</tr>
<tr>
<td>1024</td>
<td>0x0400</td>
<td>Debug event class 3. Most prolific debug messages</td>
</tr>
<tr>
<td>2048</td>
<td>0x0800</td>
<td>Debug event class 4. Limit messages.</td>
</tr>
</tbody>
</table>
13.4.3.1 Specifying Log Event Classes

Each daemon uses an integer representation of a bit string to specify its log event classes. The bit string can be decimal or hexadecimal. Each daemon’s log event classes are specified in a different way: the Server and the MOMs use a bit string that includes the events to be logged, whereas the Scheduler’s bit string represents the events that are to be excluded from its log.

For example, if you want the Server to log all events except those at event classes 512 and 1024 (hex 0x200 and 0x400), you would use a log event class of 511. This is 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1. If you want to log events at event classes 1, 2, and 16, you would set the log event class to 19.

The following table shows the log event parameter for each daemon:

<table>
<thead>
<tr>
<th>PBS</th>
<th>Parameter/Attribute</th>
<th>Reference</th>
<th>How to Make Parameter Take Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>log_events attribute</td>
<td>pbs_server_attributes(7B) man page</td>
<td>Takes effect immediately with qmgr</td>
</tr>
<tr>
<td>MOM</td>
<td>$logevent parameter</td>
<td>pbs_mom(8B) man page</td>
<td>Requires SIGHUP to MOM</td>
</tr>
<tr>
<td>Scheduler</td>
<td>log_filter parameter</td>
<td>pbs_sched(8B) man page.</td>
<td>Requires SIGHUP to Scheduler</td>
</tr>
</tbody>
</table>

When reading the PBS event logfiles, you may see messages of the form “Type 19 request received from PBS_Server...”. These “type codes” correspond to different PBS batch requests.
13.4.3.1.1 Specifying Server Log Events

The Server’s log events are specified in the Server’s log_events attribute. The attribute is an integer representation of a bit string, where the integer includes all events to be logged. To set the value, use the qmgr command:

```
qmgr: set server log_events = <value>
```

For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to 1025 (1024 + 1, or 0x0401). To include all events, set the value to 0xffffffff. The default value for this attribute is 511 (0x1ff). It can be set by Operators and Managers only. See the pbs_server_attributes(7B) manual page.

13.4.3.1.2 Specifying MOM Log Events

Each MOM’s log events are specified in the $logevent parameter in that MOM’s configuration file PBS_HOME/mom_priv/config. The parameter is an integer representation of a bit string, where the integer includes all events to be logged. For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to 1025 (1024 + 1, or 0x0401). To include all events, set the value to 4095 (0xffffffff). The default value used by MOM is 975 (0x03cf). This parameter can be set by root only. See the pbs_mom(8B) manual page.

13.4.3.1.3 Specifying Scheduler Log Events

The Scheduler’s log events are specified in the log_filter Scheduler configuration parameter in the Scheduler configuration file PBS_HOME/sched_priv/sched_config. This parameter is a bit string that is made up of all the log events to be excluded from the Scheduler’s log file. To set the value, edit the log_filter line in PBS_HOME/sched_priv/sched_config. For example, to exclude common debug messages (128, or 0x0080) and reservation-related messages (512, or 0x0200), set the parameter value to 640 or 0x0280 (128 + 512, or 0x0200 + 0x0080). To include all events, set the value to 0 (zero). To log nothing, set the value to 4095 (0xffffffff). (The default value is 3328 (0x0D00). This parameter can be set by root only. See the pbs_sched(8B) manual page.
13.4.4 Event Logfile Format and Contents

13.4.4.1 Event Logfile Format

Each component event logfile is a text file with each entry terminated by a new line. The format of an entry is:

logfile-date-time;event_code;server_name;object_type;object_name;message

- The logfile-date-time field is a date and time stamp in the format: mm/dd/yyyy hh:mm:ss
- The event_code is a bitmask for the type of event which triggered the event logging. It corresponds to the bit position, 0 to n, of each log event in the event mask of the PBS component writing the event record. See section 13.4.1 “PBS Events” on page 890 for a description of the event mask.
- The server_name is the name of the Server which logged the message. This is recorded in case a site wishes to merge and sort the various logs in a single file.
- The object_type is the type of object which the message is about. All messages are associated with an object_type. The following lists each
possible *object_type*:

Table 13-4: List of Event Logfile Object Types

<table>
<thead>
<tr>
<th>object_type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Svr</td>
<td>for server</td>
</tr>
<tr>
<td>Que</td>
<td>for queue</td>
</tr>
<tr>
<td>Job</td>
<td>for job</td>
</tr>
<tr>
<td>Req</td>
<td>for request</td>
</tr>
<tr>
<td>Fil</td>
<td>for file</td>
</tr>
<tr>
<td>Act</td>
<td>for accounting string</td>
</tr>
<tr>
<td>Node</td>
<td>for vnode or host</td>
</tr>
<tr>
<td>Resv</td>
<td>for reservation</td>
</tr>
<tr>
<td>Sched</td>
<td>for scheduler</td>
</tr>
</tbody>
</table>

- The *object_name* is the name of the specific object.
- The *message* field is the text of the log message.

### 13.4.4.2 Scheduler Commands

These commands tell the scheduler why a scheduling cycle is being started. These commands appear in the Server’s logfile. Each has a decimal value, shown below. The following table shows commands from the Server to the scheduler.

Table 13-5: Commands from Server to Scheduler

<table>
<thead>
<tr>
<th>Value</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>New job enqueued</td>
</tr>
<tr>
<td>2</td>
<td>Job terminated</td>
</tr>
<tr>
<td>3</td>
<td>Scheduler time interval reached</td>
</tr>
<tr>
<td>4</td>
<td>Cycle again after scheduling one job</td>
</tr>
</tbody>
</table>
13.4.5 Logging Job Usage

PBS can log per-vnode cputime usage. The mother superior logs cputime in the format “hh:mm:ss” for each vnode of a multi-vnode job. The log event class of these messages is 0x0100.

To append job usage to standard output for an interactive job, use a shell script for the epilogue which contains the following:

```bash
#!/bin/sh
tracejob -sl $1 | grep 'cput'
```
13.4.6 Managing Log Files

13.4.6.1 Disk Space for Log Files

It is important not to run out of disk space for logging. You should periodically check the available disk space, and check the size of the log files PBS is writing, so that you know how fast you are using up disk space. Make sure that you always have more than enough disk space available for log files.

13.4.6.2 Dividing Up Log Files

You may wish to divide a day’s logging up into more than one file. You may want to create a logfile that contains only the entries of interest. You can specify a file for a daemon’s event log. See section 13.4.6.3 “Specifying Log File Path” on page 899. The next sections describe how to break up your log files.

13.4.6.2.1 Dividing Log Files on UNIX/Linux

On UNIX and Linux systems, all daemons close and reopen the same named log file when they are sent a SIGHUP. The process identifier (PID) of each daemon is available in its lock file in its home directory. You can move the current log file to a new name and send SIGHUP to restart the file using the following commands:

```
cd $PBS_HOME/<daemon>_logs
mv <current log file> <archived log file>
kil -HUP ‘cat ../<daemon>_priv/<daemon>.lock’
```
13.4.6.2.2 Dividing Log Files on Windows

On Windows systems, you can rotate the event log files by stopping the service for which you want to rotate the logfile, moving the file, and then restarting that service. For example:

```
cd "%PBS_HOME%\<service>_logs"
net stop pbs_<service>
mv <current log file> <archived log file>
net start pbs_<service>
```

13.4.6.3 Specifying Log File Path

You may wish to specify an event logfile path that is different from the default path. Each daemon has an option to specify a different path for the daemon’s event logfile. This option is the `-L logfile` option, and it is the same for all daemons. For example, to start the Scheduler so that it logs events in `/scratch/my_sched_log`:

```
pbs_sched -L /scratch/my_sched_log
```

See the `pbs_server(8B)`, `pbs_sched(8B)`, and `pbs_mom(8B)` manual pages.

13.4.7 Extracting Logged Information

You can use the `tracejob` command to extract information from log files, such as why a job is not running or when a job was queued. The `tracejob` command can read both event logs and accounting logs. See the `tracejob(8B)` manual page.
13.5 Using the UNIX syslog Facility

Each PBS component logs various event classes of information about events in its own log file. While having the advantage of a concise location for the information from each component, the disadvantage is that in a complex, the logged information is scattered across each execution host. The UNIX syslog facility can be useful.

If your site uses the syslog subsystem, PBS may be configured to make full use of it. The following entries in pbs.conf control the use of syslog by the PBS components:

<table>
<thead>
<tr>
<th>Table 13-6: Entries in pbs.conf for Using Syslog</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PBS_LOCALLOG=x</strong></td>
</tr>
<tr>
<td>Enables logging to local PBS log files. Only possible when logging via syslog feature is enabled.</td>
</tr>
<tr>
<td>0 = no local logging</td>
</tr>
<tr>
<td>1 = local logging enabled</td>
</tr>
<tr>
<td><strong>PBS_SYSLOG=x</strong></td>
</tr>
<tr>
<td>Controls the use of syslog and syslog “facility” under which the entries are logged. If x is:</td>
</tr>
<tr>
<td>0 - no syslogging</td>
</tr>
<tr>
<td>1 - logged via LOG_DAEMON facility</td>
</tr>
<tr>
<td>2 - logged via LOG_LOCAL0 facility</td>
</tr>
<tr>
<td>3 - logged via LOG_LOCAL1 facility</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>9 - logged via LOG_LOCAL7 facility</td>
</tr>
</tbody>
</table>
Table 13-6: Entries in pbs.conf for Using Syslog

<table>
<thead>
<tr>
<th>PBS_SYSLOGSEVR=y</th>
<th>Controls the severity level of messages that are logged; see /usr/include/sys/syslog.h. If y is:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 - only LOG_EMERG messages are logged</td>
</tr>
<tr>
<td></td>
<td>1 - messages up to LOG_ALERT are logged</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>7 - messages up to LOG_DEBUG are logged</td>
</tr>
</tbody>
</table>

**IMPORTANT:**

PBS_SYSLOGSEVR is used in addition to PBS's log_events mask which controls the class of events (job, vnode, ...) that are logged.

## 13.6 Managing Machines

### 13.6.1 Offlining Hosts and Vnodes

To offline an entire host, use the pbsnodes command. Use the name of the natural vnode, which is usually the name of the host:

```
pbsnodes -o <name of natural vnode>
```

All vnodes on this host are offlined.

To offline a single vnode, use the qmgr command, with the name of the vnode:

```
qmgr -c "set node foo[3] state=offline"
```
13.6.1.1 Caveats of Offlining

If you set a vnode with no running jobs offline, the server will not attempt to communicate with the vnode. Therefore, the server will not notice that the vnode is up until you clear the offline state. For example, a vnode that is both down and offline will not be marked up by the server until you clear the offline state.

13.7 Managing the Data Service

13.7.1 PBS Monitors Data Service

PBS monitors its connection to the data service. If the connection is broken (for example, because the data service is down), PBS tries to reestablish the connection. If necessary, PBS restarts the data service.

If failover is configured, and PBS cannot reestablish a connection, PBS quits.

If failover is not configured, PBS attempts to reestablish the connection until it succeeds.

When the server is stopped, it stops the data service.

13.7.2 Data Service on UNIX/Linux

The data service daemon requires, and runs from, the data service user account. On UNIX, the requirements for this account are the following:

- Non-root account
- Account is enabled
- If you are using failover, the UID of this account must be the same on both primary and secondary server hosts
- We recommend that the account is called pbsdata. The installer looks for an account called pbsdata. If this account exists, the installer does
not need to prompt for a username, and can install silently.

- Root must be able to `su` to the data service account and run commands as that user. Do not add lines such as `exec bash` to the `.profile` of the data service account. If you want to use `bash` or similar, set this in the `/etc/passwd` file, via the OS tools for user management.

- The data service account must have a home directory.

### 13.7.3 Data Service on Windows

On Windows, this account is the PBS service account from which the other PBS daemons run.

The data service account must be the same as the PBS service account.

The data service account must have a home directory.

### 13.7.4 Setting Data Service Account and Password

You can change the user account and password for the PBS data service using the `pbs_ds_password` command. Use this command if you need to change the user account or update the password for the data service. See “pbs_ds_password” on page 56 of the PBS Professional Reference Guide.

To change the data service user account:

```
pbs_ds_password -C <new user account>
```

To change the data service password:

```
pbs_ds_password
```

### 13.7.4.1 Caveats

- The specified new user account must already exist.
- The account name cannot be changed while the data service is running.
- Do not delete `PBS_HOME/server_priv/db_password`. Doing so will prevent the `pbs_ds_password` command from being able to
function.

- Do not change the data service password using any method other than the `pbs_ds_password` command.

### 13.7.5 Starting and Stopping the Data Service

PBS automatically starts and stops the data service. However, you can start, stop, or check the status of the PBS data service using the `pbs_dataservice` command. See “`pbs_dataservice`” on page 54 of the PBS Professional Reference Guide.

To start the data service:

```
pbs_dataservice start
```

To stop the data service:

```
pbs_dataservice stop
```

To get the status of the data service:

```
pbs_dataservice status
```

#### 13.7.5.1 Caveats

The data service cannot be stopped while the PBS server is running.

### 13.7.6 Changing Data Service Port

You can change the port that the data service listens on by changing the setting of the `PBS_DATA_SERVICE_PORT` entry in `pbs.conf`.

#### 13.7.6.1 Caveats

- The PBS daemons must not be running when the port is changed.
- The data service must not be running when the port is changed.
13.7.7 File Ownership

The files under PBS_HOME/datastore are owned by the data service user account.

13.7.8 Recommendations

13.7.8.1 Avoiding Harmless Warnings

On Itanium architecture, you may see many harmless warnings in the logs about floating-point arithmetic. To avoid this:

- Log in as root
- Type the following:
  
  dmesg -n4

For examples, see section 14.5.17 “Floating-point Arithmetic Warnings in Logs” on page 916.
Chapter 14

Problem Solving

The following is a list of common problems and recommended solutions. Additional information is always available online at the PBS website, www.pbspro.com/UserArea. The last section in this chapter gives important information on how to get additional assistance from the PBS Support staff.

14.1 Server Host Bogs Down After Startup

If the server host becomes unresponsive a short time after startup, the server may be trying to contact the wrong license server.
14.1.1 Symptoms

15 seconds to one or two minutes after you start the PBS server, the system becomes unresponsive.

14.1.2 Problem

The problem may be caused by the `pbs_license_info` server attribute pointing to an old FLEX license server. This attribute should point to the new LM-X license server. See section 4.4 “Configuring PBS for Licensing” on page 105 in the PBS Professional Installation & Upgrade Guide.

14.1.3 Treatment

On some Linux systems, the effects of memory starvation on subsequent responsiveness may be long-lasting. Therefore, instead of merely killing and restarting the PBS server, we recommend rebooting the machine.

Take the following steps:

1. Reboot the machine into single-user mode.
2. Determine the correct value for `pbs_license_info` and set the `PBS_LICENSE_FILE_LOCATION` entry in `pbs.conf` to this value.
3. Reboot, or change `runlevel` to multi-user.
4. Using `qmgr`, set the `pbs_license_info` server attribute to the correct value:

   ```
   # qmgr -c "set server pbs_license_info = <value of PBS_LICENSE_FILE_LOCATION entry in pbs.conf>"
   ```

5. Stop the PBS server process.
6. Remove the `PBS_LICENSE_FILE_LOCATION` entry in `pbs.conf`.
7. Continue normally.
14.2 Finding PBS Version Information

Use the `qstat` command to find out what version of PBS Professional you have.

```
qstat -fB
```

In addition, each PBS command will print its version information if given the `--version` option. This option cannot be used with other options.

14.3 Troubleshooting and Hooks

You may wish to disable hook execution in order to debug PBS issues. To verify whether hooks are part of the problem, disable each hook by setting its `enabled` attribute to `False`.

14.4 Directory Permission Problems

If for some reason the access permissions on the PBS file tree are changed from their default settings, a component of the PBS system may detect this as a security violation, and refuse to execute. If this is the case, an error message to this effect will be written to the corresponding log file. You can run the `pbs_probe` command to check (and optionally correct) any directory permission (or ownership) problems. See “pbs_probe” on page 89 of the PBS Professional Reference Guide for details on usage of the `pbs_probe` command.
14.5 Common Errors

14.5.1 Clients Unable to Contact Server

If a client command (such as `qstat` or `qmgr`) is unable to connect to a Server there are several possibilities to check. If the error return is 15034, “No server to connect to”, check (1) that there is indeed a Server running and (2) that the default Server information is set correctly. The client commands will attempt to connect to the Server specified on the command line if given, or if not given, the Server specified by `SERVER_NAME` in `pbs.conf`.

If the error return is 15007, “No permission”, check for (2) as above. Also check that the executable `pbs_iff` is located in the search path for the client and that it is setuid root. Additionally, try running `pbs_iff` by typing:

```
pbs_iff -t server_host 15001
```

Where `server_host` is the name of the host on which the Server is running and 15001 is the port to which the Server is listening (if started with a different port number, use that number instead of 15001). Check for an error message and/or a non-zero exit status. If `pbs_iff` exits with a non-zero status, either the Server is not running or was installed with a different encryption system than was `pbs_iff`.

14.5.2 Vnodes Down

The PBS Server determines the state of vnodes (up or down), by communicating with MOM on the vnode. The state of vnodes may be listed by two commands: `qmgr` and `pbsnodes`

```
Qmgr: list node @active
pbsnodes -a
```

```
Node jupiter  state = state-unknown, down
```

A vnode in PBS may be marked “down” in one of two substates. For example, the state above of vnode “jupiter” shows that the Server has not had contact with MOM since the Server came up. Check to see if a MOM
is running on the vnode. If there is a MOM and if the MOM was just started, the Server may have attempted to poll her before she was up. The Server should see her during the next polling cycle in 10 minutes. If the vnode is still marked “state-unknown, down” after 10+ minutes, either the vnode name specified in the Server’s node file does not map to the real network hostname or there is a network problem between the Server’s host and the vnode.

If the vnode is listed as:

```bash
pbsnodes -a
Node jupiter state = down
```

then the Server has been able to ping MOM on the vnode in the past, but she has not responded recently. The Server will send a “ping” PBS message to every free vnode each ping cycle, 10 minutes. If a vnode does not acknowledge the ping before the next cycle, the Server will mark the vnode down.

### 14.5.3 Requeueing a Job “Stuck” on a Down Vnode

PBS Professional will detect if a vnode fails when a job is running on it, and will automatically requeue and schedule the job to run elsewhere. If the user marked the job as “not rerunnable” (i.e. via the `qsub -r n` option), then the job will be deleted rather than requeued. If the affected vnode is vnode 0 (Mother Superior), the requeue will occur quickly. If it is another vnode in the set assigned to the job, it could take a few minutes before PBS takes action to requeue or delete the job. However, if the auto-requeue feature is not enabled, or if you wish to act immediately, you can manually force the requeueing and/or rerunning of the job. See section 10.4.2 “Node Fail Requeue: Jobs on Failed Vnodes” on page 763.

If you wish to have PBS simply remove the job from the system, use the “-Wforce” option to `qdel`:

```bash
qdel -Wforce jobID
```

If instead you want PBS to requeue the job, and have it immediately eligible to run again, use the “-Wforce” option to `qrerun`

```bash
qrerun -Wforce jobID
```
14.5.4 File Stagein Failure

When stagein fails, the job is placed in a 30-minute wait to allow the user time to fix the problem. Typically this is a missing file or a network outage. Email is sent to the job owner when the problem is detected. Once the problem has been resolved, the job owner or the Operator may remove the wait by resetting the time after which the job is eligible to be run via the -a option to qalter. The server will update the job’s comment with information about why the job was put in the wait state. The job’s exec_host string is cleared so that it can run on any vnode(s) once it is eligible.

14.5.5 File Stageout Failure

When stageout encounters an error, there are three retries. PBS waits 1 second and tries again, then waits 11 seconds and tries a third time, then finally waits another 21 seconds and tries a fourth time. PBS sends the job’s owner email if the stageout is unsuccessful. For each attempt, if PBS is using scp and that doesn’t work, PBS will then try rcp.

14.5.6 Non-delivery of Output

If the output of a job cannot be delivered to the user, it is saved in a special directory:
PBS_HOME/undelivered and mail is sent to the user. The typical causes of non-delivery are:

1. The destination host is not trusted and the user does not have a .rho-sts file.
2. An improper path was specified.
3. A directory in the specified destination path is not writable.
4. The user’s .cshrc on the destination host generates output when executed.
5. The path specified by PBS_SCP in pbs.conf is incorrect.
6. The PBS_HOME/spool directory on the execution host does not have the correct permissions. This directory must have mode 1777 drwxr-xr-x (on UNIX) or “Full Control” for “Everyone” (on Windows).

See section 7.5, ”Delivery of Output Files”, on page 165 of the PBS Professional User’s Guide.

14.5.7 Job Cannot be Executed

If a user receives a mail message containing a job ID and the line “Job cannot be executed”, the job was aborted by MOM when she tried to place it into execution. The complete reason can be found in one of two places, MOM’s log file or the standard error file of the user’s job. If the second line of the message is “See Administrator for help”, then MOM aborted the job before the job’s files were set up. The reason will be noted in MOM’s log. Typical reasons are a bad user/group account, checkpoint/restart file (Cray or SGI), or a system error. If the second line of the message is “See job standard error file”, then MOM had created the job’s file and additional messages were written to standard error. This is typically the result of a bad resource request.

14.5.8 Running Jobs with No Active Processes

On very rare occasions, PBS may be in a situation where a job is in the Running state but has no active processes. This should never happen as the death of the job’s shell should trigger MOM to notify the Server that the
job exited and end-of-job processing should begin. If this situation is noted, PBS offers a way out. Use the `qsig` command to send `SIGNULL`, signal 0, to the job. (Usage of the `qsig` command is provided in the PBS Professional User’s Guide.) If MOM finds there are no processes then she will force the job into the exiting state.

### 14.5.9 Job Held Due to Invalid Password

If a job fails to run due to an invalid password, then the job will be put on hold (hold type “p”), its `comment` field updated as to why it failed, and an email sent to user for remedy action. See also the `qhold` and `qrls` commands in the PBS Professional User’s Guide.

### 14.5.10 SuSE 9.1 with mpirun and ssh

Use “`ssh -n`” instead of “`ssh`”.

### 14.5.11 Jobs that Can Never Run

If backfilling is being used, the scheduler looks at the job being backfilled around and determines whether that job can never run.

If backfilling is turned on, the scheduler determines whether that job can or cannot run now, and if it can't run now, whether it can ever run. If the job can never run, the scheduler logs a message saying so.

The scheduler only considers the job being backfilled around. That is the only job for which it will log a message saying the job can never run.

This means that a job that can never run will sit in the queue until it becomes the most deserving job. Whenever this job is considered for having small jobs backfilled around it, the error message “resource request is impossible to solve: job will never run” is printed in the scheduler’s log file. If backfilling is off, this message will not appear.

If backfilling is turned off, the scheduler determines only whether that job can or cannot run now. The scheduler won't determine if a job will ever run or not.
14.5.12 Server Won’t Start

- The Server will not start if the nodes file contains a bad vnode name. Add the offending vnode to the local /etc/hosts file (even if you must use a bogus address), start server and use qmgr to delete the vnode. Once the server is up and the vnode is deleted, the entry can be removed from /etc/hosts.

- The Server may not start due to problems with the data service. Call PBS technical support; see “Technical Support” on page ii. For more on the PBS data service, see “pbs_dataservice” on page 54 of the PBS Professional Reference Guide.

14.5.13 PBS Data Service Doesn’t Start

- You may need to create the data service user account. This must be creating before installing PBS. See section 3.4.1 “Create Required Accounts” on page 43 in the PBS Professional Installation & Upgrade Guide.

14.5.14 Problem With Dynamic Resource

If you need to debug a dynamic resource being supplied by an external script, it may help to follow these steps:

1. Set the scheduler’s log_filter parameter to zero (everything is logged)
   - Edit PBS/Home/sched_priv/sched_config
   - Change the log_filter line to 0
2. Send a SIGHUP to the scheduler (pbs_sched)
3. The scheduler log will contain the value the scheduler reads from the external script
14.5.15 Can’t Create Formula or Hook

You must run `qmgr` at the server host when operating on the server’s `job_sort_formula` attribute or on hooks. For example, attempting to create the formula at another host will result in the following error:

```
qmgr obj= svr=default: Unauthorized Request
   job_sort_formula
```

14.5.16 Data Service Running When PBS Server is Down

You can use the `pbs_dataservice` command to stop the data service. See “pbs_dataservice” on page 54 of the PBS Professional Reference Guide.

14.5.17 Floating-point Arithmetic Warnings in Logs

On Itanium architecture, some floating-point calculations trigger the kernel to log a warning message in `/var/log/messages`. These messages can be ignored, but they can fill up space. The messages look like this:

```
Oct 14 02:00:03 altix-09 kernel: postgres(24535):
   floating-point assist fault at ip 400000000057b191,
   isr 0000020000000008

Oct 14 02:00:03 altix-09 kernel: postgres(24535):
   floating-point assist fault at ip 400000000057b1a1,
   isr 0000020000000008

Oct 14 02:00:03 altix-09 kernel: postgres(24535):
   floating-point assist fault at ip 400000000057b212,
   isr 0000020000000001
```
To avoid this, turn off floating-point assist messages:

- Log in as root
- Type the following:
  
  \[ \texttt{dmesg -n4} \]

### 14.5.18 Unrecognized Timezone Variable

In order to create reservations, the PBS server must recognized the PBS\_TZID environment variable at the submission host. The appropriate zone location for the submission host can be obtained from the machine on which the PBS Professional server is installed.

- On Linux platforms, either use the `tzselect` command, if it is available, or look in the underlying operating system’s `zone.tab` timezone location file, which may be found under `/usr/share/zoneinfo/zone.tab`.
- On all other platforms, look in the list of libical supported zoneinfo locations available under `$PBS_EXEC/lib/ical/zoneinfo/zones.tab`.

### 14.6 Errors on Windows

This section discusses errors encountered under Windows.

#### 14.6.1 Windows: Services Don’t Start

In the case where the PBS daemons, the Active Directory database, and the domain controller are all on the same host, some PBS services may not start up immediately. If the Active Directory services are not running when the PBS daemons are started, the daemons won’t be able to talk to the domain controller. This can prevent the PBS daemons from starting. As a workaround, wait until the host is completely up, then retry starting the failing service.
Example:

```bash
net start pbs_server
```

### 14.6.2 MOMs Won’t Start

In a domained environment, if the PBS service account is a member of any group besides “Domain Users”, the install program will fail to add the PBS service account to the local Administrators group on the install host. Make sure that the PBS service account is a member of only one group, “Domain Users” in a domained environment.

### 14.6.3 Windows: qstat Errors

If the `qstat` command produces an error such as:

```
illegally formed job identifier.
```

This means that the DNS lookup is not working properly, or reverse lookup is failing. Use the following command to verify DNS reverse lookup is working:

```bash
pbs_hostn -v hostname
```

If however, `qstat` reports “No Permission”, then check `pbs.conf`, and look for the entry “PBS_EXEC”. `qstat` (in fact all the PBS commands) will execute the command “PBS_EXEC\sbin\pbs_iff” to do its authentication. Ensure that the path specified in `pbs.conf` is correct.

### 14.6.4 Windows: qsub Errors

If, when attempting to submit a job to a remote server, `qsub` reports:

```
BAD uid for job execution
```

Then you need to add an entry in the remote system's `.rhosts` or `hosts.equiv` pointing to your Windows machine. Be sure to put in all hostnames that resolve to your machine. See section 9.10.3 “User Authentication” on page 693.
If remote account maps to an Administrator-type account, then you need to set up a `.rhosts` entry, and the remote server must carry the account on its `acl_roots` list.

### 14.6.5 Windows: Server Reports Error 10035

If Server is not able to contact the Scheduler running on the same local host, it may print to its log file the error message,

```
10035 (Resources Temporarily Unavailable)
```

This is often caused by the local hostname resolving to a bad IP address. Perhaps, in `%WINDIR%\system32\drivers\etc\hosts`, localhost and `hostname` were mapped to 127.0.0.1.

### 14.6.6 Windows: Server Reports Error 10054

If the Server reports error 10054 `rp_request()`, this indicates that another process, probably `pbs_sched`, `pbs_mom`, or `pbs_send_job` is hung up causing the Server to report bad connections. If you desire to kill these services, then use Task Manager to find the Service’s process ID, and then issue the command:

```
pbskill process-id
```

### 14.6.7 Windows: PBS Permission Errors

If the Server, MOM, or Scheduler fails to start up because of permission problems on some of its configuration files like `pbs_environment` or `mom_priv/config`, then correct the permission by running:

```
pbs_mkdirs server
pbs_mkdirs mom
pbs_mkdirs sched
```
14.6.8  Windows: Errors When Not Using Drive C:

If PBS is installed on a hard drive other than C:, it may not be able to locate the pbs.conf global configuration file. If this is the case, PBS will report the following message:

```
E:\Program Files\PBS Pro\exec\bin>qstat -
pbsconf error: pbs conf variables not found:
PBS_HOME PBS_EXEC
No such file or directory
qstat: cannot connect to server UNKNOWN (errno=0)
```

To correct this problem, set PBS_CONF to point pbs.conf to the right path. Normally, during PBS Windows installation, this would be set in system autoexec.bat which will be read after the Windows system has been restarted. Thus, after PBS Windows installation completes, be sure to reboot the Windows system in order for this variable to be read correctly.

14.6.9  Windows: Vnode Comment “ping: no stream”

If a vnode shows a “down” status in xpbsmon or “pbsnodes -a” and contains a vnode comment with the text “ping: no stream” and “write err”, then attempt to restart the Server as follows to clear the error:

```
net stop pbs_server
net start pbs_server
```

14.6.10 Windows: Services Debugging Enabled

The PBS services, pbs_server, pbs_mom, pbs_sched, and pbs_rshd are compiled with debugging information enabled. Therefore you can use a debugging tool (such as Dr. Watson) to capture a crash dump log which will aid the developers in troubleshooting the problem. To con-
figure and run Dr. Watson, execute `drwtsn32` on the Windows command line, set its “Log Path” appropriately and click on the button that enables a popup window when Dr. Watson encounters an error. Then run a test that will cause one of the PBS services to crash and email to PBS support the generated output in `Log_Path`. Other debugging tools may be used as well.

### 14.6.11 Windows: Client Commands Slow

PBS caches the IP address of the local host, and uses this to communicate between the server, scheduler, and MOM. If the cached IP address is invalidated, PBS can become slow. In both scenarios, jobs must be killed and restarted.

#### 14.6.11.1 Scenario 1: Wireless Router, DHCP Enabled

The system is connected to a wireless router that has DHCP enabled. DHCP returned a new IP address for the server short name, but DNS is resolving the server full name to a different IP address.

The IP address and server full name have become invalid due to the new DHCP address. PBS has cached the IP address of the server full name. Therefore, the PBS server times out when trying to connect to the scheduler and local MOM using the previously cached IP address. This makes PBS slow.

**Symptom:**

1. PBS is slow.
   
   a. Server logs show "Could not contact scheduler".
   
   b. `pbsnodes -a` shows that the local node is down.

2. First IP addresses returned below don't match:
   
   ```
   cmd.admin> pbs_hostn -v <server_short_name>
   cmd.admin> pbs_hostn -v <server_full_name>
   ```
Chapter 14  Problem Solving

Workaround: cache the correct new IP address of the local server host.

1. Add the address returned by `pbs_hostn -v <server_short_name>` (normally the DHCP address) to the `%WINDIR%\system32\drivers\etc\hosts` file as follows:

   `<DHCP address> <server_full_name> <server_short_name>`

2. Restart all the PBS services:

   ```
   cmd.admin> net stop pbs_sched
   cmd.admin> net stop pbs_mom
   cmd.admin> net stop pbs_rshd
   cmd.admin> net stop pbs_server
   cmd.admin> net start pbs_sched
   cmd.admin> net start pbs_mom
   cmd.admin> net start pbs_rshd
   cmd.admin> net start pbs_server
   ```

14.6.11.2 Scenario 2: DHCP-Enabled Environment

The system is running in a DHCP-enabled environment. Both the server short name and server full name resolve to the same DHCP address. Then the DHCP address expires and the local server host gets a new address, invalidating what's been cached by PBS.

Symptom:

1. PBS is slow.
   a. Server logs show "Could not contact scheduler".
   b. `pbsnodes -a` shows local node is down.

2. The first IP addresses below match, but it's now a different IP address:

   ```
   cmd.admin> pbs_hostn -v <server_short_name>
   cmd.admin> pbs_hostn -v <server_full_name>
   ```
Workaround: obtain the correct new IP address of the local server host.

1. Simply restart all the PBS services:
   
   ```
   cmd.admin> net stop pbs_sched
   cmd.admin> net stop pbs_mom
   cmd.admin> net stop pbs_rshd
   cmd.admin> net stop pbs_server
   
   cmd.admin> net start pbs_sched
   cmd.admin> net start pbs_mom
   cmd.admin> net start pbs_rshd
   cmd.admin> net start pbs_server
   ```

   **IMPORTANT:**
   
   When contacting PBS Professional Support, please provide as much of the following information as possible:
   
   - PBS SiteID
   - Output of the following commands:
     ```
     qstat -Bf
     qstat -Qf
     pbsnodes -a
     ```
   - If the question pertains to a certain type of job, include:
     `qstat -f job_id`
   - If the question is about scheduling, also send your `(PBS_HOME)/sched_priv/sched_config` file.

To expand, renew, or change your PBS support contract, contact our Sales Department. (See contact information on the inside front cover of this manual.)
14.7  Troubleshooting PBS Licenses

14.7.1  Wrong License Server: Out of Memory

If you run out of memory shortly after startup, the server may be looking for the wrong license server. See section 14.1 “Server Host Bogs Down After Startup” on page 907.

14.7.2  Unable to Connect to License Server

If PBS cannot contact the license server, the server will log a message:

"Unable to connect to license server at
pbs_license_info=<X>"

If the license file location is incorrectly initialized (e.g. if the host name or port number is incorrect), PBS may not be able to pinpoint the misconfiguration as the cause of the failure to reach a license server.

If PBS cannot detect a license server host and port when it starts up, the server logs an error message:

"Did not find a license server host and port
(pbs_license_info=<X>). No external license server will be contacted"

14.7.3  Unable to Run Job; Unable to Obtain Licenses

If the PBS scheduler cannot obtain the licenses to run or resume a job, the scheduler will log a message:

"Could not run job <job>; unable to obtain <N> CPU licenses. avail licenses=<Y>"

"Could not resume <job>; unable to obtain <N> CPU licenses. avail licenses=<Y>"
14.7.4  Job in Reservation Fails to Run

A job in a reservation may not be able to run due to a shortage of licenses. The scheduler will log a message similar to the following:

"Could not run job <job>; unable to obtain <N> CPU licenses. avail_licenses=<Y>"

If the value of the pbs_license_min attribute is less than the number of CPUs in the PBS complex when a reservation is being confirmed, the server will log a warning:

"WARNING: reservation <resID> confirmed, but if reservation starts now, its jobs are not guaranteed to run as pbs_license_min=<X> < <Y> (# of CPUs in the complex)"

14.7.5  New Jobs Not Running

If PBS loses contact with the Altair License Server, any jobs currently running will not be interrupted or killed. The PBS server will continually attempt to reconnect to the license server, and re-license the assigned vnodes once the contact to the license server is restored.

No new jobs will run if PBS server loses contact with the License server.

14.7.6  Insufficient Minimum Licenses

If the PBS server cannot get the number of licenses specified in pbs_license_min from the license server, the server will log a message:

"checked-out only <X> CPU licenses instead of pbs_license_min=<Y> from license server at host <H>, port <P>. Will try to get more later."
14.7.7  Wrong Type of License

If the PBS server encounters a proprietary license key that is of not type “T”, then the server will log the following message:

“license key #1 is invalid: invalid type or version”.

14.7.8  User Error Messages

If a user’s job could not be run due to unavailable licenses, the job will get a comment:

“Could not run job <job>; unable to obtain <N> CPU licenses. avail_licenses=<Y>”

If a user's job could not be resumed due to unavailable licenses, the job will get a comment:

“Could not resume job <job>; unable to obtain <N> CPU licenses. avail_licenses=<Y>”

14.8  Security-related Problems

14.8.1  PBS Daemon Will Not Start

If the PBS Server, MOM, or Scheduler fails to start up, it may be refusing to start because it has detected permissions problems in its directories or on one or more of its configuration files, such as pbs_environment or mom_priv/config.
14.8.1.1 Correcting Permissions Problems on UNIX/Linux

You can use the `pbs_probe` command to detect and repair file and directory permissions problems. You can run `pbs_probe` in report mode or fix mode; in report mode, it reports the errors found; in fix mode, it attempts to fix detected problems, and reports any problems it could not fix.

To fix permissions errors, log into the host you wish to check, and run the following command:

```
pbs_probe -f
```

See the `pbs_probe(8B)` manual page.

14.8.1.2 Correcting Permissions Problems on Windows

You can use the `pbs_mkdirs` command to correct file and directory permissions problems on Windows. The command checks and if necessary repairs the permissions of configuration files such as `pbs_environment` and `mom_priv/config`. You should run the `pbs_mkdirs` command only while the PBS services are stopped.

To repair permissions on the Server/Scheduler host, log into the host and run the following commands:

```
net stop pbs_server
net stop pbs_sched
pbs_mkdirs server
pbs_mkdirs sched
net start pbs_server
net start pbs_sched
```

To repair permissions on an execution host, log into the host and run the following commands:

```
net stop pbs_mom
pbs_mkdirs mom
net start pbs_mom
```
14.8.2 Crash Recovery

PBS daemons could terminate unexpectedly either because the host machine stops running or because the daemon itself stops running. The daemon may be killed by mistake, or may (rarely) crash. The Server may terminate if the filesystem runs out of space.

14.8.2.1 Recovery When Host Machine Stops

If the host machine stops running, no special steps are required, since PBS will be started when the machine starts.

14.8.2.1.1 Execution Host Stops

If the host machine is an execution host, any jobs that were running on that host were terminated when the machine stopped, and when MOM is restarted, she will report to the Server that those jobs are dead, and begin normal activity. The Server will automatically restart any jobs that can be restarted.

Shutting down one host of a multi-host job will cause that job to be killed. The job will have to be rerun; restarting the MOM on the stopped host with the -p option will not help the job. See section 14.8.2.2.2 “MOM Start Options” on page 929.

14.8.2.1.2 Server/Scheduler Host Stops

If the host machine is the Server/Scheduler host, no data is lost and no jobs are lost, because the Server writes everything to disk. The Server is restarted automatically upon machine startup.

The Scheduler is started automatically upon machine startup. The Scheduler starts fresh each cycle, so it does not lose data.
14.8.2.2 Recovery When Daemon Stops

14.8.2.2.1 Restarting MOM

If the daemon that stops running is the MOM, you can restart it according to your needs. The default is for MOM to start up as if the machine had just been started. You can choose to preserve running jobs, to kill and requeue running jobs, or to have MOM report her previous jobs to the Server as dead.

14.8.2.2.2 MOM Start Options

(No options)
MOM does not kill any processes. She reports to the Server that her previous jobs are dead.

-r
MOM kills all her jobs. The Server requeues any jobs that can be rerun.

-p
MOM leaves any running jobs in the running state.

Be careful when starting MOM with the -p or -r options on a newly-booted system, because these options depend on the assumption that the PIDs from the previous invocation are still valid.

14.8.2.2.3 Restarting the Server

If the daemon that stops running is the Server, you can restart it according to your needs. The default is for the Server to start up, leaving jobs in their pre-shutdown state. You can choose to preserve running jobs, to purge all jobs, or to re-create the PBS complex (not recommended). In most cases, you should allow the Server to be restarted by the PBS start script.

14.8.2.2.4 Server Start Options

(No options) or
-t warm
The Server leaves all jobs in their pre-crash states. The Server starts a normal scheduling cycle, without giving spe-
cial precedence to jobs that were killed and requeued upon shutdown.

-t hot

The Server leaves all running jobs running. The Server attempts to start any jobs that were killed and requeued upon shutdown before starting any new jobs.

-t cold

The Server purges all jobs, but retains its configuration information.

-t create

Not recommended. The Server starts everything fresh. It purges all jobs, and deletes all information about the PBS complex, including attribute settings and queues.

14.8.2.2.5 Restarting the Scheduler

If the Scheduler was killed, you must restart it using this command:

    pbs_sched

If the Scheduler crashed, it will restart itself. If the Scheduler crashed due to a SIGSEGV or a SIGBUS, it will wait until five minutes have passed before restarting itself. You can restart by hand it at any time.

14.8.2.3 Filesystem Runs Out of Space

If your filesystem has run out of space, the Server may experience errors or may crash. If the Server is still running, you need only to free up enough space. If the Server has crashed, you must restart it. See section 14.8.2.2.3 “Restarting the Server” on page 929.
14.9 Getting Help

If the material in the PBS manuals is unable to help you solve a particular problem, you may need to contact the PBS Support Team for assistance. First, be sure to check the Customer Login area of the PBS Professional website, which has a number of ways to assist you in resolving problems with PBS, such as the Tips & Advice page.

The PBS Professional support team can also be reached directly via email and phone (contact information on the inside front cover of this manual).
Appendix A: License Agreement

CAUTION!

PRIOR TO INSTALLATION OR USE OF THE SOFTWARE YOU MUST CONSENT TO THE FOLLOWING SOFTWARE LICENSE TERMS AND CONDITIONS BY CLICKING THE “I ACCEPT” BUTTON BELOW. YOUR ACCEPTANCE CREATES A BINDING LEGAL AGREEMENT BETWEEN YOU AND ALTAIR. IF YOU DO NOT HAVE THE AUTHORITY TO BIND YOUR ORGANIZATION TO THESE TERMS AND CONDITIONS, YOU MUST CLICK “I DO NOT ACCEPT” AND THEN HAVE AN AUTHORIZED PARTY IN THE ORGANIZATION THAT YOU REPRESENT ACCEPT THESE TERMS.

IF YOU, OR THE ORGANIZATION THAT YOU REPRESENT, HAS A MASTER SOFTWARE LICENSE AGREEMENT (“MASTER SLA”) ON FILE AT THE CORPORATE HEADQUARTERS OF ALTAIR ENGI-
NEERING, INC. (“ALTAIR”), THE MASTER SLA TAKES PRECEDENCE OVER THESE TERMS AND SHALL GOVERN YOUR USE OF THE SOFTWARE.

MODIFICATION(S) OF THESE SOFTWARE LICENSE TERMS IS EXPRESSLY PROHIBITED. ANY ATTEMPTED MODIFICATION(S) WILL BE NONBINDING AND OF NO FORCE OR EFFECT UNLESS EXPRESSLY AGREED TO IN WRITING BY AN AUTHORIZED CORPORATE OFFICER OF ALTAIR. ANY DISPUTE RELATING TO THE VALIDITY OF AN ALLEGED MODIFICATION SHALL BE DETERMINED IN ALTAIR’S SOLE DISCRETION.

Altair Engineering, Inc. - Software License Agreement

THIS SOFTWARE LICENSE AGREEMENT, including any Additional Terms (together with the “Agreement”), shall be effective as of the date of YOUR acceptance of these software license terms and conditions (the “Effective Date”) and is between ALTAIR ENGINEERING, INC., 1820 E. Big Beaver Road, Troy, MI 48083-2031, USA, a Michigan corporation (“Altair”), and YOU, or the organization on whose behalf you have authority to accept these terms (the “Licensee”). Altair and Licensee, intending to be legally bound, hereby agree as follows:

1. DEFINITIONS. In addition to terms defined elsewhere in this Agreement, the following terms shall have the meanings defined below for purposes of this Agreement:

Additional Terms. Additional Terms are those terms and conditions which are determined by an Altair Subsidiary to meet local market conditions.

Documentation. Documentation provided by Altair or its resellers on any media for use with the Products.

Execute. To load Software into a computer’s RAM or other primary memory for execution by the computer.

Global Zone: Software is licensed based on three Global Zones: the Americas, Europe and Asia-Pacific. When Licensee has Licensed Workstations located in multiple Global Zones, which are connected to a single License (Network) Server, a premium is applied to the standard Software License pricing for a single Global Zone.
ISV/Independent Software Vendor. A software company providing its products, (“ISV Software”) to Altair's Licensees through the Altair License Management System using Altair License Units.

License Log File. A computer file providing usage information on the Software as gathered by the Software.

License Management System. The license management system (LMS) that accompanies the Software and limits its use in accordance with this Agreement, and which includes a License Log File.

License (Network) Server. A network file server that Licensee owns or leases located on Licensee's premises and identified by machine serial number and/or HostID on the Order Form.

License Units. A parameter used by the LMS to determine usage of the Software permitted under this Agreement at any one time.

Licensed Workstations. Single-user computers located in the same Global Zone(s) that Licensee owns or leases that are connected to the License (Network) Server via local area network or Licensee's private wide-area network.

Maintenance Release. Any release of the Products made generally available by Altair to its Licensees with annual leases, or those with perpetual licenses who have an active maintenance agreement in effect, that corrects programming errors or makes other minor changes to the Software. The fees for maintenance and support services are included in the annual license fee but perpetual licenses require a separate fee.

Order Form. Altair's standard form in either hard copy or electronic format that contains the specific parameters (such as identifying Licensee's contracting office, License Fees, Software, Support, and License (Network) Servers) of the transaction governed by this Agreement.

Products. Products include Altair Software, ISV Software, and/or Suppliers' software; and Documentation related to all of the forgoing.

Proprietary Rights Notices. Patent, copyright, trademark or other proprietary rights notices applied to the Products, packaging or media.

Software. The Altair software identified in the Order Form and any Updates or Maintenance Releases.
**Subsidiary.** Subsidiary means any partnership, joint venture, corporation or other form of enterprise in which a party possesses, directly or indirectly, an ownership interest of fifty percent (50%) or greater, or managerial or operational control.

**Suppliers.** Any person, corporation or other legal entity which may provide software or documents which are included in the Software.

**Support.** The maintenance and support services provided by Altair pursuant to this Agreement.

**Templates.** Human readable ASCII files containing machine-interpretable commands for use with the Software.

**Term.** The term of licenses granted under this Agreement. Annual licenses shall have a 12-month term of use unless stated otherwise on the Order Form. Perpetual licenses shall have a term of twenty-five years. Maintenance agreements for perpetual licenses have a 12-month term.

**Update.** A new version of the Products made generally available by Altair to its Licensees that includes additional features or functionalities but is substantially the same computer code as the existing Products.

2. **LICENSE GRANT.** Subject to the terms and conditions set forth in this Agreement, Altair hereby grants Licensee, and Licensee hereby accepts, a limited, non-exclusive, non-transferable license to: a) install the Products on the License (Network) Server(s) identified on the Order Form for use only at the sites identified on the Order Form; b) execute the Products on Licensed Workstations in accordance with the LMS for use solely by Licensee's employees, or its onsite Contractors who have agreed to be bound by the terms of this Agreement, for Licensee's internal business use on Licensed Workstations within the Global Zone(s) as identified on the Order Form and for the term identified on the Order Form; c) make backup copies of the Products, provided that Altair's and its Suppliers' and ISV's Proprietary Rights Notices are reproduced on each such backup copy; d) freely modify and use Templates, and create interfaces to Licensee's proprietary software for internal use only using APIs provided that such modifications shall not be subject to Altair's warranties, indemnities, support or other Altair obligations under this Agreement; and e) copy and distribute Documentation inside Licensee's organization exclusively for use by Licensee's employees and its onsite Contractors who have agreed to be bound by the terms of this Agreement. A copy of the License Log File shall be made available to Altair automatically on no less than a monthly basis. In
the event that Licensee uses a third party vendor for information technology (IT) support, the IT company shall be permitted to access the Software only upon its agreement to abide by the terms of this Agreement. Licensee shall indemnify, defend and hold harmless Altair for the actions of its IT vendor(s).

3. **RESTRICTIONS ON USE.** Notwithstanding the foregoing license grant, Licensee shall not do (or allow others to do) any of the following: a) install, use, copy, modify, merge, or transfer copies of the Products, except as expressly authorized in this Agreement; b) use any back-up copies of the Products for any purpose other than to replace the original copy provided by Altair in the event it is destroyed or damaged; c) disassemble, decompile or “unlock”, reverse translate, reverse engineer, or in any manner decode the Software or ISV Software for any reason; d) sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the Products or Licensee's rights under this Agreement; e) allow use outside the Global Zone(s) or User Sites identified on the Order Form; f) allow third parties to access or use the Products such as through a service bureau, wide area network, Internet location or time-sharing arrangement except as expressly provided in Section 2(b); g) remove any Proprietary Rights Notices from the Products; h) disable or circumvent the LMS provided with the Products; or (i) link any software developed, tested or supported by Licensee or third parties to the Products (except for Licensee's own proprietary software solely for Licensee's internal use).

4. **OWNERSHIP AND CONFIDENTIALITY.** Licensee acknowledges that all applicable rights in patents, copyrights, trademarks, service marks, and trade secrets embodied in the Products are owned by Altair and/or its Suppliers or ISVs. Licensee further acknowledges that the Products, and all copies thereof, are and shall remain the sole and exclusive property of Altair and/or its Suppliers and ISVs. This Agreement is a license and not a sale of the Products. Altair retains all rights in the Products not expressly granted to Licensee herein. Licensee acknowledges that the Products are confidential and constitute valuable assets and trade secrets of Altair and/or its Suppliers and ISVs. Licensee agrees to take the same precautions necessary to protect and maintain the confidentiality of the Products as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement. Licensee shall promptly notify Altair in the event any unauthorized person obtains access to the Products. If Licensee is required by any governmental authority or court of law to disclose Altair's or its ISV's
or its Suppliers' confidential information, then Licensee shall immediately
notify Altair before making such disclosure so that Altair may seek a pro-
tective order or other appropriate relief. Licensee's obligations set forth in
Section 3 and Section 4 of this Agreement shall survive termination of this
Agreement for any reason. Altair's Suppliers and ISVs, as third party bene-
ficiaries, shall be entitled to enforce the terms of this Agreement directly
against Licensee as necessary to protect Supplier's intellectual property or
other rights.

Altair and its resellers providing support and training to licensed end users
of the Products shall keep confidential all Licensee information provided to
Altair in order that Altair may provide Support and training to Licensee.
Licensee information shall be used only for the purpose of assisting Lic-
ensee in its use of the licensed Products. Altair agrees to take the same pre-
cautions necessary to protect and maintain the confidentiality of the
Licensee information as it does to protect its own information of a confi-
dential nature but in any event, no less than a reasonable degree of care,
and shall not disclose or make them available to any person or entity except
as expressly provided in this Agreement.

5. MAINTENANCE AND SUPPORT. Maintenance. Altair will
provide Licensee, at no additional charge for annual licenses and for a
maintenance fee for paid-up licenses, with Maintenance Releases and
Updates of the Products that are generally released by Altair during the
term of the licenses granted under this Agreement, except that this shall not
apply to any Term or Renewal Term for which full payment has not been
received. Altair does not promise that there will be a certain number of
Updates (or any Updates) during a particular year. If there is any question
or dispute as to whether a particular release is a Maintenance Release, an
Update or a new product, the categorization of the release as determined by
Altair shall be final. Licensee agrees to install Maintenance Releases and
Updates promptly after receipt from Altair. Maintenance Releases and
Updates are subject to this Agreement. Altair shall only be obligated to pro-
vide support and maintenance for the most current release of the Software
and the most recent prior release. Support. Altair will provide support via
telephone and email to Licensee at the fees, if any, as listed on the Order
Form. If Support has not been procured for any period of time for paid-up
licenses, a reinstatement fee shall apply. Support consists of responses to
questions from Licensee's personnel related to the use of the then-current
and most recent prior release version of the Software. Licensee agrees to
provide Altair with sufficient information to resolve technical issues as
may be reasonably requested by Altair. Licensee agrees to the best of its
abilities to read, comprehend and follow operating instructions and procedures as specified in, but not limited to, Altair's Documentation and other correspondence related to the Software, and to follow procedures and recommendations provided by Altair in an effort to correct problems. Licensee also agrees to notify Altair of a programming error, malfunction and other problems in accordance with Altair's then current problem reporting procedure. If Altair believes that a problem reported by Licensee may not be due to an error in the Software, Altair will so notify Licensee. Questions must be directed to Altair's specially designated telephone support numbers and email addresses. Support will also be available via email at Internet addresses designated by Altair. Support is available Monday through Friday (excluding holidays) from 8:00 a.m. to 5:00 p.m. local time in the Global Zone where licensed, unless stated otherwise on the Order Form.

**Exclusions.** Altair shall have no obligation to maintain or support (a) altered, damaged or Licensee-modified Software, or any portion of the Software incorporated with or into other software not provided by Altair; (b) any version of the Software other than the current version of the Software or the immediately prior release of the Software; (c) problems caused by Licensee's negligence, abuse or misapplication of Software other than as specified in the Documentation, or other causes beyond the reasonable control of Altair; or (d) Software installed on any hardware, operating system version or network environment that is not supported by Altair. Support also *excludes* configuration of hardware, non-Altair Software, and networking services; consulting services; general solution provider related services; and general computer system maintenance.

**6. WARRANTY AND DISCLAIMER.** Altair warrants for a period of ninety (90) days after Licensee initially receives the Software that the Software will perform under normal use substantially as described in then current Documentation. Supplier software included in the Software and ISV Software provided to Licensee shall be warranted as stated by the Supplier or the ISV. Copies of the Suppliers' and ISV's terms and conditions of warranty are available on the Altair Support website. Support services shall be provided in a workmanlike and professional manner, in accordance with the prevailing standard of care for consulting support engineers at the time and place the services are performed.

**ALTAIR DOES NOT REPRESENT OR WARRANT THAT THE PRODUCTS WILL MEET LICENSEE'S REQUIREMENTS OR THAT THEIR OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT IT WILL BE COMPATIBLE WITH ANY PARTICULAR HARDWARE OR SOFTWARE. ALTAIR**
EXCLUDES AND DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES NOT STATED HEREIN, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. THE ENTIRE RISK FOR THE PERFORMANCE, NON-PERFORMANCE OR RESULTS OBTAINED FROM USE OF THE PRODUCTS RESTS WITH LICENSEE AND NOT ALTAIR. ALTAIR MAKES NO WARRANTIES WITH RESPECT TO THE ACCURACY, COMPLETENESS, FUNCTIONALITY, SAFETY, PERFORMANCE, OR ANY OTHER ASPECT OF ANY DESIGN, PROTOTYPE OR FINAL PRODUCT DEVELOPED BY LICENSEE USING THE PRODUCTS.

7. INDEMNITY. Altair will defend and indemnify, at its expense, any claim made against Licensee based on an allegation that the Software infringes a patent or copyright (“Claim”); provided, however, that this indemnification does not include claims which are based on Supplier software or ISV software, and that Licensee has not materially breached the terms of this Agreement, Licensee notifies Altair in writing within ten (10) days after Licensee first learns of the Claim; and Licensee cooperates fully in the defense of the claim. Altair shall have sole control over such defense; provided, however, that it may not enter into any settlement binding upon Licensee without Licensee's consent, which shall not be unreasonably withheld. If a Claim is made, Altair may modify the Software to avoid the alleged infringement, provided however, that such modifications do not materially diminish the Software's functionality. If such modifications are not commercially reasonable or technically possible, Altair may terminate this Agreement and refund to Licensee the prorated license fee that Licensee paid for the then current Term. Perpetual licenses shall be pro-rated over a 36-month term. Altair shall have no obligation under this Section 7, however, if the alleged infringement arises from Altair's compliance with specifications or instructions prescribed by Licensee, modification of the Software by Licensee, use of the Software in combination with other software not provided by Altair and which use is not specifically described in the Documentation, and if Licensee is not using the most current version of the Software, if such alleged infringement would not have occurred except for such exclusions listed here. This section 7 states Altair's entire liability to Licensee in the event a Claim is made. No indemnification is made for Supplier and/or ISV Software.
8. LIMITATION OF REMEDIES AND LIABILITY. Licensee's exclusive remedy (and Altair's sole liability) for Software that does not meet the warranty set forth in Section 6 shall be, at Altair's option, either (i) to correct the nonconforming Software within a reasonable time so that it conforms to the warranty; or (ii) to terminate this Agreement and refund to Licensee the license fees that Licensee has paid for the then current Term for the nonconforming Software; provided, however that Licensee notifies Altair of the problem in writing within the applicable Warranty Period when the problem first occurs. Any corrected Software shall be warranted in accordance with Section 6 for ninety (90) days after delivery to Licensee. The warranties hereunder are void if the Software has been improperly installed, misused, or if Licensee has violated the terms of this Agreement.

Altair's entire liability for all claims arising under or related in any way to this Agreement (regardless of legal theory), shall be limited to direct damages, and shall not exceed, in the aggregate for all claims, the license and maintenance fees paid under this Agreement by Licensee in the 12 months prior to the claim on a prorated basis, except for claims under Section 7. ALTAIR AND ITS SUPPLIERS AND ISVS SHALL NOT BE LIABLE TO LICENSEE OR ANYONE ELSE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING HEREUNDER (INCLUDING LOSS OF PROFITS OR DATA, DEFECTS IN DESIGN OR PRODUCTS CREATED USING THE SOFTWARE, OR ANY INJURY OR DAMAGE RESULTING FROM SUCH DEFECTS, SUFFERED BY LICENSEE OR ANY THIRD PARTY) EVEN IF ALTAIR OR ITS SUPPLIERS OR ITS ISVS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Licensee acknowledges that it is solely responsible for the adequacy and accuracy of the input of data, including the output generated from such data, and agrees to defend, indemnify, and hold harmless Altair and its Suppliers and ISVs from any and all claims, including reasonable attorney's fees, resulting from, or in connection with Licensee's use of the Software. No action, regardless of form, arising out of the transactions under this Agreement may be brought by either party against the other more than two (2) years after the cause of action has accrued, except for actions related to unpaid fees.

9. UNITED STATES GOVERNMENT RESTRICTED RIGHTS.
This section applies to all acquisitions of the Products by or for the United States government. By accepting delivery of the Products except as provided below, the government or the party procuring the Products under government funding, hereby agrees that the Products qualify as “commer-
cial” computer software as that term is used in the acquisition regulations applicable to this procurement and that the government's use and disclosure of the Products is controlled by the terms and conditions of this Agreement to the maximum extent possible. This Agreement supersedes any contrary terms or conditions in any statement of work, contract, or other document that are not required by statute or regulation. If any provision of this Agreement is unacceptable to the government, Vendor may be contacted at Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031; telephone (248) 614-2400. If any provision of this Agreement violates applicable federal law or does not meet the government's actual, minimum needs, the government agrees to return the Products for a full refund.

For procurements governed by DFARS Part 227.72 (OCT 1998), the Software, except as described below, is provided with only those rights specified in this Agreement in accordance with the Rights in Commercial Computer Software or Commercial Computer Software Documentation policy at DFARS 227.7202-3(a) (OCT 1998). For procurements other than for the Department of Defense, use, reproduction, or disclosure of the Software is subject to the restrictions set forth in this Agreement and in the Commercial Computer Software - Restricted Rights FAR clause 52.227-19 (June 1987) and any restrictions in successor regulations thereto.

Portions of Altair's PBS Professional Software and Documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision(c)(1)(ii) of the rights in the Technical Data and Computer Software clause in DFARS 252.227-7013, or in subdivision (c)(1) and (2) of the Commercial Computer Software-Restricted Rights clause at 48 CFR52.227-19, as applicable.

10. CHOICE OF LAW AND VENUE. This Agreement shall be governed by and construed under the laws of the state of Michigan, without regard to that state's conflict of laws principles except if the state of Michigan adopts the Uniform Computer Information Transactions Act drafted by the National Conference of Commissioners of Uniform State Laws as revised or amended as of June 30, 2002 (“UCITA”) which is specifically excluded. This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. Each Party waives its right to a jury trial in the event of any dispute arising under or relating to this Agreement. Each party agrees that money damages may not be an adequate remedy for breach of the provisions of this Agreement, and in the event of such breach,
the aggrieved party shall be entitled to seek specific performance and/or injunctive relief (without posting a bond or other security) in order to enforce or prevent any violation of this Agreement.

11. [RESERVED]

12. Notice. All notices given by one party to the other under the Agreement or these Additional Terms shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All notices shall be deemed given upon actual receipt.

Written notice shall be made to:

**Altair:**  **Licensee Name & Address:**

Altair Engineering, Inc.

1820 E. Big Beaver Rd

Troy, MI 48083

Attn: Tom M. Perring

13. TERM. For annual licenses, or Support provided for perpetual licenses, renewal shall be automatic for each successive year (“Renewal Term”), upon mutual written execution of a new Order Form. All charges and fees for each Renewal Term shall be set forth in the Order Form executed for each Renewal Term. All Software licenses procured by Licensee may be made coterminous at the written request of Licensee and the consent of Altair.

14. TERMINATION. Either party may terminate this Agreement upon thirty (30) days prior written notice upon the occurrence of a default or material breach by the other party of its obligations under this Agreement (except for a breach by Altair of the warranty set forth in Section 8 for which a remedy is provided under Section 10; or a breach by Licensee of Section 5 or Section 6 for which no cure period is provided and Altair may terminate this Agreement immediately) if such default or breach continues for more than thirty (30) days after receipt of such notice. Upon termination of this Agreement, Licensee must cease using the Software and, at Altair's option, return all copies to Altair, or certify it has destroyed all such copies of the Software and Documentation.
15. **GENERAL PROVISIONS. Export Controls.** Licensee acknowledges that the Products may be subject to the export control laws and regulations of the United States and other countries, and any amendments thereof. Licensee agrees that Licensee will not directly or indirectly export the Products into any country or use the Products in any manner except in compliance with all applicable U.S. and other countries export laws and regulations. **Notice.** All notices given by one party to the other under this Agreement shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All notices shall be deemed given upon actual receipt. **Assignment.** Neither party shall assign this Agreement without the prior written consent of other party, which shall not be unreasonably withheld. All terms and conditions of this Agreement shall be binding upon and inure to the benefit of the parties hereto and their respective successors and permitted assigns. **Waiver.** The failure of a party to enforce at any time any of the provisions of this Agreement shall not be construed to be a waiver of the right of the party thereafter to enforce any such provisions. **Severability.** If any provision of this Agreement is found void and unenforceable, such provision shall be interpreted so as to best accomplish the intent of the parties within the limits of applicable law, and all remaining provisions shall continue to be valid and enforceable. **Headings.** The section headings contained in this Agreement are for convenience only and shall not be of any effect in constructing the meanings of the Sections. **Modification.** No change or modification of this Agreement will be valid unless it is in writing and is signed by a duly authorized representative of each party. **Conflict.** In the event of any conflict between the terms of this Agreement and any terms and conditions on a Licensee Purchase Order or comparable document, the terms of this Agreement shall prevail. Moreover, each party agrees any additional terms on any Purchase Order or comparable document other than the transaction items of (a) item(s) ordered; (b) pricing; (c) quantity; (d) delivery instructions and (e) invoicing directions, are not binding on the parties. In the event of a conflict between the terms of this Agreement, and the Additional Terms, the Agreement shall take precedence. **Entire Agreement.** This Agreement, the Additional Terms, and the Order Form(s) attached hereto constitute the entire understanding between the parties related to the subject matter hereto, and supersedes all proposals or prior agreements, whether written or oral, and all other communications between the parties with respect to such subject matter. This Agreement may be executed in
one or more counterparts, all of which together shall constitute one and the same instrument. **Execution.** Copies of this Agreement executed via original signatures, facsimile or email shall be deemed binding on the parties.
Index

$logevent MOM parameter 894
$restrict_user 683
$restrict_user_exceptions 683
$restrict_user_maxsysid 683
.rhosts 681

A
Accept an Action 464
access
  by group 659
  by user 659
    effect of flatuid 681
control lists 660
from host 660
to a queue 659
to a reservation 659
to the Server 659
Accounting 889

log 886
acl_group_enable
  queue attribute 672
acl_groups
  queue attribute 672
acl_host_enable
  queue attribute 671, 672
  Server attribute 671, 672
acl_hosts
  queue attribute 671, 672
  Server attribute 671, 672
acl_roots 698
acl_user_enable
  queue attribute 671, 672
  Server attribute 671, 672
acl_users
  queue attribute 671, 672
  Server attribute 671, 672
Index

ACLs 660
  default behavior 661
  format 661
  group 662
  host 662
  matching entry 664
  modifying behavior 661
  overrides 680
  removing entity 667
  rules for creating 666
  user 662
  who can create 669
Action 464
Advance and Standing Reservations 283
  advance reservation 283, 320
  aggressive_provision 603
  alarm 481
  alt_id 862
  AOE 600
  aoe resource
    defining 615
  application license
    floating 390
    definition 321
  application licenses 386
    floating externally-managed 390
    floating license PBS-managed 392
  license units and features 386
  overview 356
  per-host node-locked example 395
  types 386
 Associating Vnodes with Multiple Queues 142
  Authorized_Groups reservation attribute 673
  Authorized_Users reservation attribute 673
  Average CPU Usage Enforcement 442
  average_cpufactor 443
  average_percent_over 443
  average_trialperiod 442
  avoid_provision 603
B
  backfill_prime 280
  basic fairshare 192
  batch requests 893
  borrowing vnode 320, 378
  built-in resource 320
C
  checkpoint 913
    preemption via 268
  chunk 320
  comment 70
  Configuration
    Server 17
  Configuring MOM on an Altix 828
  consumable resource 320
  CPU 321
  cpuaverage 442
  cpuset_create_flags 830
  cpuset_destroy_delay 830
  cput 197
  Creating Queues 22
  creation of provisioning hook 619
  credential 693
  csaswitch 838
  current_aoe 616
  custom resource 321
  custom resources
    application licenses 386
    floating managed by PBS 392
Index

overview 356
per-host node-locked 395
types 386
how to use 353
restart steps 367
scratch space
overview 357
Static Host-level 377
Static Server-level 373
cycle harvesting
ideal_load 167
max_load 167

D
decay 197
dedicated time 171
defining aoe resource 615
defining provisioning policy 621
Defining Resources for the Altix 381
degraded reservation 284
department 193
Deprecations 10
DIS 883
DNS 918
Dynamic Fit 238
Dynamic Resource Scripts/Programs 357

E
egroup 193
euser 193
Eligible Wait Time 173
eligible_time 174, 177
enabled 480
euser 193
event 480
event classes 893
exiting 174
express_queue 260

F
fair_share 192
Fairshare 191
fairshare 261
fairshare entities 193
fairshare ID 195
Fairshare Tree 192
fairshare_usage_res 196
Files
pbs.conf 918
Finished jobs 875
flatuid Server attribute 681
floating License
example 390
floating license
definition 321
floating licenses
example of externally-managed 390

G
gethostname 693
global resource 322
Globus 18, 883
configuration 836
MOM 836
PBS_MANAGER_GLOBUS_SERVICE_PORT 883
PBS_MOM_GLOBUS_SERVICE_PORT 883
group
access 659
ACLs 662
Group limit 413
group limit 322

H
half_life 197
help, getting 931
History jobs 875
Index

hook
  creation of provisioning 619
  provisioning 600
Hooks 461
  alarm 481
  Creating empty hooks 474
  Deleting 475
  enabled 480
  Enabling and disabling 482
  event 480
  Exporting 477
  Importing 475
  order 481
  Overview of creating 473
  Setting order of execution 483
  Setting timeout 484
  setting trigger events 481
  Type 479
  User 480
hooks
  log event class objects 505
host
  access 660
  ACLs 662
host ACLs 662
host.equiv 681
Hostbased Authentication 694
HPC Basic Profile (HPCBP) 579
HPC Basic Profile Server 580
HPCBP 579
  Enable HPCBP MOM 591
  OGSA HPC Profile 597
  Self-signed certificate 591
  Service account 586
  Staging Protocol 588
  Steps to configure PBS 585
  User credentials 589
HPCBP MOM 580, 581
  Configuration Variables 582
  Vnode Attributes 583
hpcbp_enable 584
hpcbp_stage_protocol 584
hpcbp_user_name 583
hpcbp_webservice_address 583
I
  ideal_load
    cycle harvesting 167
  indirect resource 322, 378
  ineligible_time 174
  InfiniBand 788
  initial_time 174
  instance 284
  instance of a standing reservation 284
J
  Job attributes
    alt_id 862
  Job History 874
    Changing Settings 877
    Configuring 876
    Enabling 876
    Setting Duration 876
  Job Submission Description Language (JSDL) 580
  job that can never run 914
  job_history_duration 876
  job_history_enable 876
L
  license
    application
      floating 390
      floating
        definition 321
  Limit 412
    for all users 412
    Generic group limit 413
    Generic user limit 412
    group limit 413
## Index

<table>
<thead>
<tr>
<th>Individual group limit 413</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual user limit 413</td>
</tr>
<tr>
<td>user limit 413</td>
</tr>
<tr>
<td>limit 322</td>
</tr>
<tr>
<td>cput 438</td>
</tr>
<tr>
<td>file size 438</td>
</tr>
<tr>
<td>for all users 323</td>
</tr>
<tr>
<td>generic group limit 321</td>
</tr>
<tr>
<td>generic user limit 321</td>
</tr>
<tr>
<td>group limit 322</td>
</tr>
<tr>
<td>individual group limit 322</td>
</tr>
<tr>
<td>individual user limit 322</td>
</tr>
<tr>
<td>pcput 438</td>
</tr>
<tr>
<td>pmem 438</td>
</tr>
<tr>
<td>pvmem 438</td>
</tr>
<tr>
<td>user limit 324</td>
</tr>
<tr>
<td>walltime 438</td>
</tr>
</tbody>
</table>

**Limits**
- Attributes 420
- Generic and Individual 416
- group 409
- Limits For All Users 417
- Resource usage 409
- Scope 414
- Setting limits 422
- user 409
- load_balancing 165
- LoadLeveler 792
- local resource 323
- Location of MOM’s configuration files 54
- log event classes 893
- log events
  - MOM 894
  - Scheduler 894
  - Server 894
- log_events Server attribute 894
- log_filter scheduler parameter 894
- logging
  - hooks log event class objects 505

**M**
- Manager
  - privilege 658
- managers Server attribute 658
- managing vnode 323, 378
- master provisioning script 600, 617
- master script 600, 617
- matching ACL entry 664
- max_concurrent_provision 621
- max_group_res 434
- max_group_run 434, 435
- max_group_run_soft 434
- max_load
  - cycle harvesting 167
- max_queuable 434
- max_queued 421
- max_queued_res 421
- max_run 420
- max_run_res 420
- max_run_res_soft 420
- max_running 433, 435
- max_user_res 434, 435
- max_user_res_soft 434, 435
- max_user_run 434, 435
- max_user_run_soft 434, 435
- memory-only vnode 323
- modifyjob 513
- MOM
  - log events 894
  - restarting 929
- movejob 513
- MPI, LAPI 792
- MPI_USE_IB 788
- MPICH 780
- mpiexec 785
- mpirun 780
- MPT 788
- Multihost Placement Sets 240
Index

N
Natural Vnode 42
natural vnode 42
ncpus*walltime 197
netwins 793
calculating 795
Network Windows 793
default 796
non-consuming resource 323
nonprimetime_prefix 280
normal_jobs 261
nrt_adapter_resources 797
nrt_load_table 797
nrt_status_adapter 797
nrtUnload_window 797

O
Operator
privilege 657
operators Server attribute 657
order 481

P
P4 780
password
valid 701, 914
single-signon 701
Windows 700, 701
pbs Module 464
pbs.conf 910, 913, 918
pbs_accounting_workload_mgnt 837
pbs_attach 781
PBS_BATCH_SERVICE_PORT 883
PBS_BATCH_SERVICE_PORT_DIS 883
PBS_CONF_SYSLOG 884, 900
PBS_CONF_SYSLOGSEVR 884, 901
PBS_CPUSETDEDICATED 787
PBS_ENVIRONMENT 883
PBS_EXEC 720, 883
PBS_EXEC/bin 782
PBS_GET_IBWINS 793
PBS_HOME 720, 883
PBS_HOME/sched_priv/
resource_group 192
PBS_HOME/sched_priv/
sched_config 192
PBS_HOME/sched_priv/usage 201
PBS_ibm_ib_default_networks 794
PBS_ibm_ib_max_instances 794
pbs_iif 693, 910, 918
PBS_LOCALLOG 883, 900
PBS_MANAGER_GLOBUS_SERVICE_PORT 883
PBS_MANAGER_SERVICE_PORT 883
pbs_mkdirs 927
PBS_MOM_GLOBUS_SERVICE_PORT 883
PBS_MOM_HOME 720, 883
PBS_MOM_SERVICE_PORT 883
PBS_MPI_DEBUG 787
PBS_MPI_SGIARRAY 787
pbs_mpirun 780
PBS_PRIMARY 720, 884
pbs_probe 927
PBS_RCP 884
pbs_rcp 688
pbs_rsub 673
PBS_SCHEDULER_SERVICE_PORT 884
PBS_SCP 688, 884
PBS_SECONDARY 720, 884
PBS_SERVER 720, 884
PBS_START_MOM 720, 884
PBS_START_SCHED 720, 884
PBS_START_SERVER 720, 884
PBScrayhost 331
Index

PBSraylabel_ 331
PBSraynid 331
PBSrayorder 331
PBSrayseg 331
pbsfs 201
pbsnodes 923
pbsrun 790
pbsrun.poe 793
pbsrun_wrap 788
   poe 793
Placement Pool 238
Placement Set 237
poe 793
   permissions 793
   wrapping 793
policy
   defining provisioning 621
preempt_order 259
preempt_prio 259
preempt_queue_prio 259
preempt_sort 259
preemption 258
preemption via checkpoint 268
preemptive scheduling 258
preemptive_sched 259
Primary Server 884
prime_spill 280, 281
primetime_prefix 280
privilege
   manager 658
   operator 657
   user 656
provision_policy 603
provisioning
   creation of hook 619
   defining policy 621
   hook 600
   master script 617
      writing 617
   overview 602
   policy 603
rebooting 602
reservations 606
vnode selection 603
vnode states 608

Q
qdel 911
qhold 914
qmgr 17, 70, 910
qrerun 911
qrls 701, 914
qstat 910, 918, 923
qsub 911
query_other_jobs Server attribute 656
queue 35
   access to a 659
   ACL 661
   attribute
      acl_group_enable 672
      acl_groups 672
      acl_host_enable 671, 672
      acl_hosts 671, 672
      acl_user_enable 671, 672
      acl_users 671, 672
   reservation 675
queue_softlimits 261
Queued jobs 413
queuejob 513
Quick Start Guide x

R
rcp 694, 884
rebooting
   provisioning 602
Reject an Action 464
reservation
   access to a 659
   ACL 661
   advance 283, 320
Index

attribute
   Authorized_Groups 673
   Authorized_Hosts 673
   Authorized_Users 673
control of creation 660
degraded 284
instance 284
queue 675
reservation ID 285
soonest occurrence 284
standing 283
   instance 284
   soonest occurrence 284
reservations
   provisioning 606
Resource 323
resource
   built-in 320
   consumable 320
   custom 321
   indirect 322, 378
   non-consumable 323
   shared 323, 378
resource limits 409
Resource usage limits 409
restart
   custom resources 367
restarting
   MOM 929
   Scheduler 930
   Server 929
restrict_user 683
restrict_user Exceptions 684
restrict_user_maxsysid 684
resv_enable 673
resv_enable Server attribute 660
resvsub 513
roles 655
rsh 694
run_time 174
runjob 513

S
Sales, PBS 923
Sales, support 923
Scheduler
   log events 894
   restarting 930
scp 694, 884
scratch space 357
dynamic
   host-level 384
   server-level 383
static
   host-level 385
   server-level 384
script
   master provisioning 617
   writing provisioning 617
Secondary Server 884
Server
   access to the 659
   ACL 661
   attribute
      acl_host_enable 671, 672
      acl_hosts 671, 672
      acl_user_enable 671, 672
      acl_users 671, 672
      flatuid 681
      log_events 894
      managers 658
      operators 657
      query_other_jobs 656
      resv_enable 660
      log events 894
      parameters 17
      recording configuration 17
      restarting 929
server_softlimits 261
Setting Hook Trigger Events 481
shared resource 323, 378
shares 192
SIGHUP 887
Index

single_signon_password_enable 700, 701
single-signon 701
soonest occurrence 284
sort key 198
ssh 694
standing reservation 283
starving_jobs 260
states
  vnodes and provisioning 608
Static Fit 238
strict_ordering and Backfilling 311
Sun Solaris-specific Memory Enforcement 441
Support team 923, 931
syslog 900

T
Task placement 237
Tips & Advice 931
tree percentage 196
Troubleshooting ProPack4 cpusets 832
Type 479
Type codes 893

U
unknown node 192
unknown_shares 192
Unset Resources 224
usage Limits 409
User 480
user
  access 659
  ACLs 662
  privilege 656
  roles 655
User Guide x, 914
User limit 413
user limit 324
user limits 409

V
Version 909
Virtual Nodes 41
vnode 41
  borrowing 320, 378
  managing 323, 378
  memory-only 323
  selection for provisioning 603
  states and provisioning 608
Vnodes
  hosts and nodes 42

W
Windows
  errors 917
  password 700, 701, 914
  wrapping poe 793
  writing provisioning script 617