PBS Portable Batch System

Administrator Guide

Albeaus Bayucan
Robert L. Henderson
Lonhyn T. Jasinskyj
Casimir Lesiak
Bhroam Mann
Tom Proett
Dave Tweten †

MRJ Technology Solutions
2672 Bayshore Parkway
Suite 810
Mountain View, CA 94043
http://pbs.mrj.com

Release: 2.1
Printed: June 17, 1999

† Numerical Aerospace Simulation Systems Division, NASA Ames Research Center, Moffett Field, CA

Portable Batch System (PBS) Software License

Acknowledgment: The Portable Batch System Software was originally developed as a joint project between the Numerical Aerospace Simulation (NAS) Systems Division of NASA Ames Research Center and the National Energy Research Supercomputer Center (NERSC) of Lawrence Livermore National Laboratory.

Redistribution of the Portable Batch System Software and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright and acknowledgment notices, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright and acknowledgment notices, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- All advertising materials mentioning features or use of this software must display the following acknowledgment:

  This product includes software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions.

DISCLAIMER OF WARRANTY

THIS SOFTWARE IS PROVIDED BY MRJ TECHNOLOGY SOLUTIONS ("MRJ") "AS IS" WITHOUT WARRANTY OF ANY KIND, AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED.

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW, SHALL MRJ, NASA, NOR THE U.S. GOVERNMENT BE LIABLE FOR ANY DIRECT DAMAGES WHATSOEVER, NOR ANY INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This license will be governed by the laws of the Commonwealth of Virginia, without reference to its choice of law rules.

This product includes software developed by the NetBSD Foundation, Inc. and its contributors.

PBS Revision History

Revision 1.0 June, 1994 — Alpha Test Release
Revision 1.1 March 15, 1995
...
Revision 1.1.7 June 6, 1996
Revision 1.1.8 August 19, 1996
Revision 1.1.9 December 20, 1996
Revision 1.1.10 July 31, 1997
Revision 1.1.11 December 19, 1997
Revision 1.1.12 July 9, 1998
Revision 2.0 October 14, 1998
Revision 2.1 May 12, 1999

Table of Contents

1. Introduction

This document is intended to provide the system administrator with the information required to build, install, configure, and manage the Portable Batch System. It is very likely that some important tidbit of information has been left out. No document of this sort can ever be complete, and until it has been updated by several different administrators at different sites, it is sure to be lacking.

You are strongly encouraged to read the PBS External Reference Specification, ERS, included with the release. Look for pbs_ers.ps in the src/doc directory.

1.1. What is PBS?

The Portable Batch System, PBS, is a batch job and computer system resource management package. It was developed with the intent to be conformant with the POSIX 1003.2d Batch Environment Standard. As such, it will accept batch jobs, a shell script and control attributes, preserve and protect the job until it is run, run the job, and deliver output back to the submitter.

PBS may be installed and configured to support jobs run on a single system, or many systems grouped together. Because of the flexibility of PBS, the systems may be grouped in many fashions.

1.2. Components of PBS

PBS consist of four major components: commands, the job Server, the job executor, and the job Scheduler. A brief description of each is given here to help you make decisions during the installation process.

Commands
: PBS supplies both command line commands that are POSIX 1003.2d conforming and a graphical interface. These are used to submit, monitor, modify, and delete jobs. The commands can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS. There are three classifications of commands: user commands which any authorized user can use, operator commands, and manager (or administrator) commands. Operator and manager commands require different access privileges.

Job Server
: The Job Server is the central focus for PBS. Within this document, it is generally referred to as *the Server* or by the execution name *pbs_server*. All commands and the other daemons communicate with the Server via an IP network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, protecting the job against system crashes, and running the job (placing it into execution).

Job Executor
: The job executor is the daemon which actually places the job into execution. This daemon, *pbs_mom*, is informally called *Mom* as it is the mother of all executing jobs. Mom places a job into execution when it receives a copy of the job from a Server. Mom creates a new session as identical to a user login session as is possible. For example, if the user's login shell is csh, then Mom creates a session in which .login is run as well as .cshrc. Mom also has the responsibility for returning the job's output to the user when directed to do so by the Server.

Job Scheduler
: The Job Scheduler is another daemon which contains the site's policy controlling which job is run and where and when it is run. Because each site has its own ideas about what is a good or effective policy, PBS allows each site to create its own Scheduler. When run, the Scheduler can communicate with the various Moms to learn about the state of system resources and with the Server to learn about the availability of jobs to

execute. The interface to the Server is through the same API as the commands. In fact, the Scheduler just appears as a batch Manager to the Server.

In addition to the above major pieces, PBS also provides a Application Program Interface, API, which is used by the commands to communicate with the Server. This API is described in the section 3 man pages firnished with PBS. A site may make use of the API to implement new commands if so desired.

1.3. Release Information

This information applies to the 2.1 release of PBS from MRJ Technology Solutions.

1.3.1. Tar File

PBS is provided as a single tar file. The tar file contains:

- This document in both postscript and text form.
- A "configure" script, all source code, header files, and make files required to build and install PBS.
- A full set of documentation sources. These are troff input files. The documentation may also be obtained by registered sites from the PBS web site: http://pbs.mrj.com

When extracting the tar file, a top level directory will be created with the above information there in. This top level directory will be named for the release version and patch level. For example, the directory will be named `pbs_v2.1p13` for release 2.1 patch level 13.

It is recommended that the files be extracted with the -p option to tar to perserve permission bits.

1.3.2. Additional Requirements

PBS uses a configure script generated by GNU autoconf to produce makefiles. If you have a POSIX make program then the makefiles generated by configure will try to take advantage of POSIX make features. If your make is unable to process the makefiles while building you may have a broken make. Should make fail during the build, try using GNU make.

If the Tcl based GUI (xpbs and xpbsmon) or the Tcl based Scheduler is used, the Tcl header file and library are required. If the BaSL Scheduler is used, yacc and lex (or GNU bison and flex) are required. The offical site for Tcl is:

http://www.scriptics.com/
ftp://ftp.scriptics.com/pub/tcl/tcl8_0

Versions of Tcl prior to 8.0 can no longer be used with PBS. Tcl and Tk version 8.0 or greater must be used.

A possible site for yacc and lex is:

prep.ai.mit.edu:/pub/gnu

2. Installation

This section attempts to explain the steps to build and install PBS. PBS installation is accomplished via the GNU autoconf process. This installation procedure requires more manual configuration than is "typical" for many packages. There are a number of options which involve site policy and therefore cannot be determined automagically.

To reach a usable PBS installation, the following steps are required:

1. Read this guide and plan a general configuration of hosts and PBS. See sections 1.2 and 3.0 through 3.2.
2. Decide where the PBS source and objects are to go. See section 2.2.
3. Untar the distribution file into the source tree. See section 2.2.
4. Select "configure" options and run configure from the top of the object tree. See sections 2.2 through 2.4.
5. Compile the PBS modules by typing "make" at the top of the object tree. See sections 2.2 and 2.3.
6. Install the PBS modules by typing "make install" at the top of the object tree. Root privilege is required. See section 2.2.
7. Create a node description file if PBS is managing a complex of nodes or a parallel system like the IBM SP. See Chapter **3. Batch System Configuration**.
8. Bring up and configure the Server. See sections 3.1 and 3.5.
9. Configure and bring up the Moms. See section 3.6.
10. Test by hand scheduling a few jobs. See the qrun(8B) man page.
11. Configure and start a Scheduler program. Set the Server to active by enabling scheduling. See Chapter 4.

2.1. Planning

PBS is able to support a wide range of configurations. It may be installed and used to control jobs on a single (large) system. It may be used to load balance jobs on a number of systems. It may be used to allocated nodes of a cluster or parallel system to parallel and serial jobs. Or it can deal with a mix of the above.

Before going any farther, we need to define a few terms. How PBS uses some of these terms is different than you may expect.

Node
: A computer system with a single Operating System image, a unified virtual memory image, and one or more IP addresses. Frequently, the term *execution host* is used for node. A box like the SGI Origin 2000, with contains multiple processing units running under a single OS copy is one node to PBS *regardless* of SGI's terminology. A box like the IBM SP which contains many units, each with their own copy of the OS, is a collection of many nodes.

  Under PBS, a node may be allocated *exclusively* or *temporarily shared*, or used, but not allocated, as *timeshared*.

Complex
: A collection of nodes managed by one batch system. A complex may be made up of nodes that are allocated to only one job at a time or of nodes that have many jobs executing on each at once or a combination of both.

Cluster
: A complex made up of cluter nodes.

Cluster Node
: A node that is allocated specifically to one job at a time (see *exclusive node*), or a few

jobs (see *temporarily-shared nodes*). This type of node may also be called *space shared*. Hosts that are timeshared among many jobs are called "timeshared."

Exclusive Nodes
: An exclusive node is one that is used by one and only one job at a time. A set of nodes is assigned exclusively to a job for the duration of that job. This is typically done to improve the performance of message passing programs.

Temporarily-shared Nodes
: A *temporarily-shared node* is one which is temporarily shared by multiple jobs. If several jobs request multiple temporarily-shared nodes, some nodes may be allocated commonly to both jobs and some may be unique to one of the jobs. When a node is allocated as a temporarily-shared node, it remains so until all jobs using it are terminated. Then the node may be next allocated again for temporarily-shared use or for exclusive use.

Timeshared
: In our context, to timeshare is to always allow multiple jobs to run concurrently on an execution host or node. A *timeshared node* is a node on which jobs are timeshared. Often the term *host* rather than node is used in conjunction with timeshared, as in *timeshared host*. If the term node is used without the timeshared prefix, the node is a cluster node which is allocated either exclusively or temporarily-shared.

  If a host, or node, is indicated to be timeshared, it will never be allocated (by the Server) exclusively or temporarily-shared.

Load Balance
: A policy wherein jobs are distributed across multiple timeshared hosts to even out the work load on each host. Being a policy, the distribution of jobs across execution hosts is solely a function of the Job Scheduler.

Node Property
: In order to have a means of grouping nodes for allocation, a set of zero or more node properties may be given to each node. The property is nothing more than a string of alphanumeric characters (first character must be alphabetic) without meaning to PBS. You, as the PBS administrator, may chose whatever property names you wish. Your choices for property names should be relayed to the users.

Batch System
: A PBS Batch System consists of one Job Server (pbs_server), one or more Job Schedulers (pbs_sched), and one or more execution servers (pbs_mom). With prior versions of PBS, a Batch System could be set up to support only a cluster of exclusive nodes **or** to support one or more timeshared hosts. There was no support for temporarily-shared nodes. With this release, a PBS Batch System may be set up to feed work to one large timeshared system, multiple time shared systems, a cluster of nodes to be used exclusively or temporarily-shared, or any combination of the preceding.

Batch Complex
: See Batch System.

If PBS is to be installed on one time sharing system, all three daemons may reside on that system; or you may place the Server (pbs_server) and/or the Scheduler (pbs_sched) on a "front end" system. Mom (pbs_mom) must run on every system where jobs are to be executed.

If PBS is to be installed on a collection of time sharing systems, a Mom must be on each and the Server and Scheduler may be installed on one of the systems or on a front end. If you are using the default supplied Scheduler program, you will need to setup a *node* file for the Server in which is named each of the time sharing systems. You will need to append `:ts` to each host name to identify them as time sharing.

The same arrangement applies to a cluster except that the node names in the node file do not have the appended `:ts`.

2.2. Installation Overview

The normal PBS build procedure is to separate the source from the target. This allows the placement of a single copy of the source on a shared file system from which multiple different target systems can be built. Also, the source can be protected from accidental destruction or modification by making the source read-only. However, if you choose, objects may be made within the source tree.

In the following descriptions, the *source tree* is the result of un-tar-ing the tar file into a directory (and subdirectories). A diagram of the source tree is show in figure 2−1.

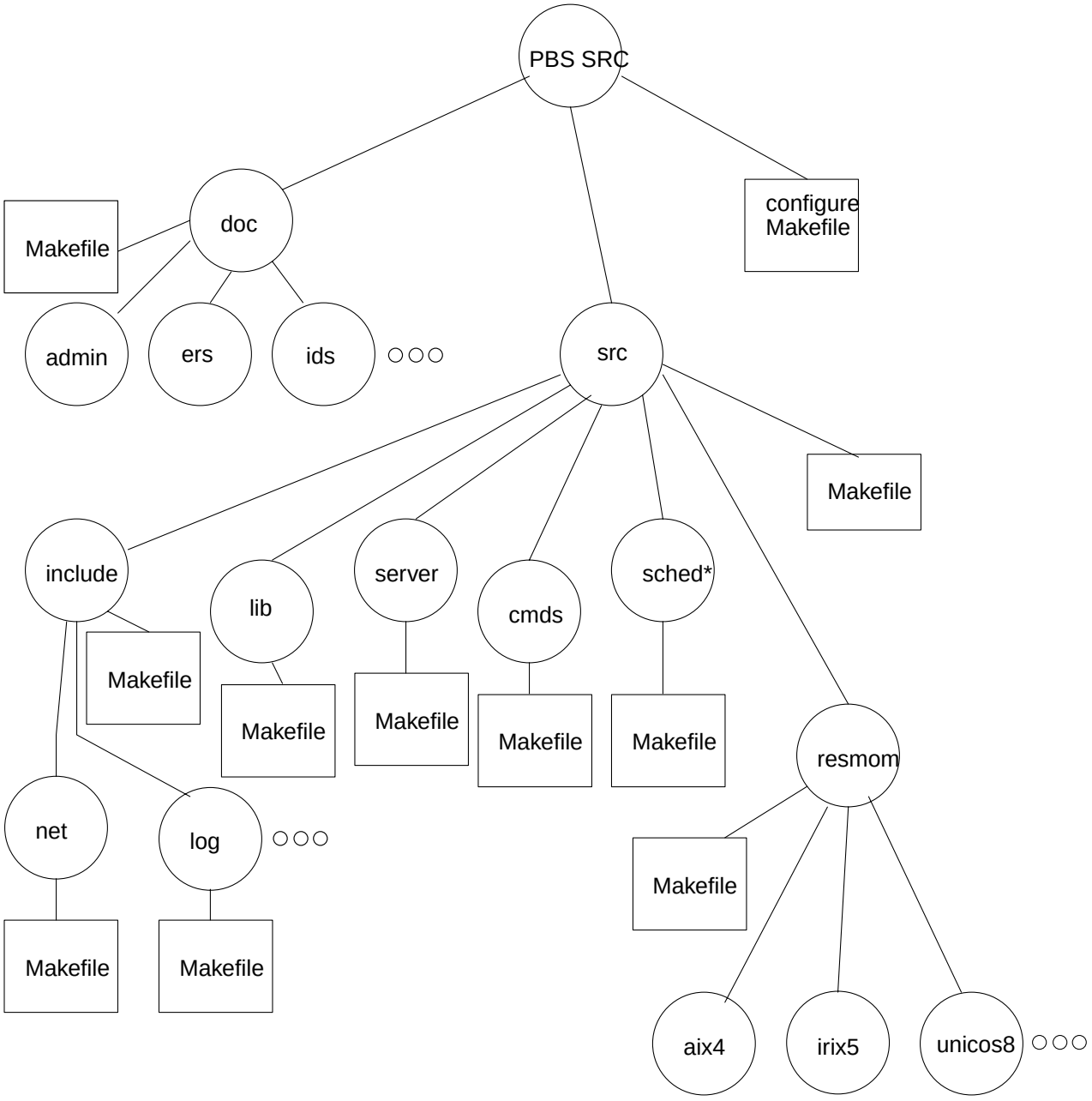

Figure 2−1: Source Tree Structure

The *target tree* is a set of parallel directories in which the object modules are actually compiled. This tree may (and generally should) be separate from the source tree.

An overview of the "configure", compile, installation and batch system configurations steps is listed here. Detailed explanation of symbols will follow. It is recommended that you read completely through these instructions before beginning the installation. To install PBS:

1. Place the tar file on the system where you would like to maintain the source.
2. Untar the tar file. It will untar in the current directory producing a single directory named for the current release and patch number. Under that directory will be several files and subdirectories. This directory and the subdirectories make up the *source tree*. You may write-protect the source tree at this point should you so choose.

   In the top directory are two files, named "Release_Notes" and "INSTALL". The Release_Notes file contains information about the release contents, changes since the last release and points to this guide for installation instructions. The "INSTALL" file consists of standard notes about the use of GNU's configure.
3. If you choose as recomended to have separate build (target) and source trees, then create the top level directory of what will become the *target tree* at this time. The target tree must reside on a file system mounted on the same architecture as the target system for which you are generating the PBS binaries. This may well be the same system as holds the source or it may not. Change directories to the top of the target tree.
4. Make a job Scheduler choice. A unique feature of PBS is its external Scheduler module. This allows a site to implement any policy of its choice. To provide even more freedom in implementing policy, PBS provides three scheduler frameworks. Schedulers may be developed in the C language, the Tcl scripting language, or PBS's very own C language extensions, the **Ba**tch **S**cheduling **L**anguage, or BaSL.

   As distributed, *configure* will default to a C language based scheduler known as *fifo*. This Scheduler can be configured to several common simple scheduling policies, not just first in − first out as the name suggests. When this Scheduler is installed, certain configuration files are installed in PBS_HOME/scheduler_priv/. **You will need to modify these files for your site.** These files are discussed in sections **4.5 QC based Sample Scheduler** and in the section **4.5.1 FIFO Scheduler**.

   To change the selected Scheduler, see the configure options **--set-sched** and **--set-sched-code** in the Features and Package Options section of this chapter. Additional information on the types of schedulers and how to configure fifo can be found in the Scheduling Policies chapter later in this guide.
5. Read setion 2.3, then from within the top of the target tree created in step 3, type the following command

   ```
   {source_tree}/configure [options]
   ```

   Where `{source_tree}` is the full relative or absolute path to the configure script in the source tree. If you are building in the source tree type `./configure [options]` at the top level of the source tree where the configure script is found.

   This will generate the complete target tree starting with the current working directory and a set of header files and make files used to build PBS. Rerunning the configure script will only need to be done if you choose to change options specified on the configure command line. See section **2.3 Build Details** for information on the configure options.

No options are absolutely required, but unless the vendor's C compiler is not ANSI, it is suggested that you use the `--set-cc` option to not use gcc. If you wish to build the GUI to PBS, and the Tcl libraries are not in the normal place, /usr/local/lib, then you will need to specify `--with-tcl=`directory, giving the path to the Tcl libraries.

Running config without any (other) options will produce a working PBS system with the following defaults:

- User commands are installed in /usr/local/bin.
- The daemons and administrative commands are installed in /usr/local/sbin.
- The working directory (PBS_HOME) for the daemons is usr/spool/pbs.
- The Scheduler will be the C based scheduler "fifo".

Because the number of options you select may be large and because each option is very wordy you may wish to create a shell script consisting of the configure command and the selected options.

The documentation is not generated by default. You may make it by specifying the `--enable-docs` option to configure or by changing into the *doc* subdirectory in the target tree and typing make.

In order to build and print PostScript copies of the documentation from the included source, you will need the GNU groff formating package including the "ms" formatting macro package. You may choose to print using different font sets. In the source tree is a file "doc/doc_fonts" which may be edited. Please read the comments in that file. Note that font position 4 is left with the symbol font mounted.

6. After running the configure script, the next step is to compile PBS by typing

```
make
```

from the top of the target tree.

7. To install PBS you must be running with root privileges. As root, type

```
make install
```

from the top of the object tree. This generates the working directory structures required for running PBS and installs the programs in the proper executable directories.

When the working directories are made, they are also checked to see that they have been setup with the correct ownership and permissions. This is performed to ensure that files are not tampered with and the security of PBS compromised. Part of the check is to insure that all parent directories and all files are:

- owned by root (bin, sys, or any uid < 10), **EPERM** returned if not;
- that group ownership is by a gid < 10, **EPERM** returned if not;
- that the directories are not world writable, or where required to be world writable that the sticky bit is set, **EACCESS** returned if not; and
- that the file or directory is indeed a file or directory, **ENOTDIR** returned if not.

The various PBS daemons will also perform similar checks when they are started.

8. If you have more than one host in your PBS cluster, you need to create a node file for the Server. Create the file `{PBS_HOME}/server_priv/nodes`. It should contain one line per node on which a Mom is to be run. The line should consist of the short host name, without the domain name parts. For example if you have three nodes: larry.stooge.com, curley.stooge.com, and moe.stooge.com; then the node file should contain

```
larry
curley
moe
```

If the nodes are timesharing nodes which will be load balanced, append `:ts` to the name of each node, as in

```
larry:ts
curley:ts
moe:ts
```

9. The three daemons, pbs_server, pbs_sched and pbs_mom must be run by root in order to function. Typically in a production system, they are started at system boot time out of the boot /etc/rc* files. This first time, you will start the daemons by hand. It does not

matter what the current working directory is when a daemon is started. The daemon will place itself in its own directory {PBS_HOME}/*_priv, where * is either serv, resmom, or sched.

Note that not all three daemons must be or even should be present on all systems. In the case of a large system, all three may be present. In the case of a cluster of workstations, you may have the Server (pbs_server) and the Scheduler (pbs_sched) on one system only and a copy of Mom (pbs_mom) on each node where jobs may be executed. At this point, it is assumed that you plan to have all three daemons running on one system.

To have a fully functional system, each of the daemons will require certain configuration information. Except for the node file, the Server's configuration information is provided via the qmgr command after the Server is running. The configuration information for Mom and the Scheduler is provided via a config file located in {PBS_HOME}/mom_priv or {PBS_HOME}/sched_priv. This is explained in detail in this guide in **Chapter 3. Batch System Configuration**.

A. Before starting the execution server(s), Mom(s), on each execution host, you will need to create her config file. To get started, the following lines are sufficient:

```
$logevent 0x1ff
$clienthost server-host
```

where *server-host* is the name of the host on which the Server is running. This is not requried if the Server and this Mom are on the same host. Create the file {PBS_HOME}/mom_priv/config and copy the above lines into it. See the pbs_mom(8) man page and section **3.6 Configuring the Execution Server** for more information on the config file.

Start the execution server, pbs_mom,

```
{sbindir}/pbs_mom
```

No options or arguments are required. See the pbs_mom(8) man page.

B. **The first time only**, start pbs_server with the "-t create" option,

```
{sbindir}/pbs_server -t create
```

See the ERS for command details. This option causes the Server to initialize various files. This option will not be required after the first time unless you wish to clear the Server database and start over. See the pbs_server(8) man page for more information. A copy of the section 8 man pages can be found in the External Reference Spec, ERS, Chapter 6.

C. Start the selected job Scheduler, pbs_sched.

i. For C language based schedulers, such as the default fifo Scheduler, options are generally required. To run the Scheduler, type

```
{sbindir}/pbs_sched
```

See the man page pbs_sched_cc(8) for more detail.

ii. For the BaSL Scheduler, the scheduling policy is written in a specialized batch scheduling language that is similar to C. The scheduling code, containing BaSL constructs, must first be converted into C using the **basl2c** utility. This is done by setting the configure option **--set-sched-code=file** where *file* is the relative (to src/scheduler.basl/samples) or absolute path of a basl source file. The file name should end in `.basl`. A good sample program is "fifo_byqueue.basl" that can schedule jobs on a single-server, single-execution host environment, or a single-server, multiple-node hosts environment. Read the header of this sample Scheduler for more information about the algorithm used.

The Scheduler configuration file is an important entity in BaSL because it is where the list of servers and host resources reside. Execute the basl based Scheduler by typing:

```
{sbindir}/pbs_sched -c config_file
```

The Scheduler searches for the config file in PBS_HOME/*sched_priv* by default. More information can be found in the man page pbs_sched_basl(8).

iii. The Tcl Scheduler requires the Tcl code policy module. Samples of Tcl scripts may be found in *src/scheduler.tcl/sample_scripts*

For the Tcl based Scheduler, the Tcl body script should be placed in PBS_HOME/sched_priv/*some_file* and the Scheduler run via

```
{sbindir}/pbs_sched -b PBS_HOME/sched_priv/some_file
```

More information can be found in the man page pbs_sched_tcl(8).

10. Log onto the system as root and define yourself to pbs_server as a manager by typing:

```
# qmgr
Qmgr: set server managers=your_name@your_host
```

Information on qmgr can be found in the qmgr(8) man page and on-line help is available by typing help within qmgr.

From this point, you no longer need root privilege. Note, *your_hos*t can be any host on which PBS' qmgr command is installed. You can now configure and manage a remote batch system from the comfort of your own workstation.

Now you need to define at least one queue. Typically it will be an execution queue unless you are using this Server purely as a gateway. You may chose to establish queue minimum, maximum, and/or default resource limits for some resources. For example, to establish a minimum of 1 second, a maximum of 12 cpu hours, and a default of 30 cpu minutes on a queue named "dque"; issue the following commands inside of qmgr:

```
Qmgr: create queue dque queue_type=e
Qmgr: s q dque resources_min.cput=1,resources_max.cput=12:00:00
Qmgr: s q dque resources_default.cput=30:00
Qmgr: s q dque enabled=true, started=true
```

You may also wish to increase the system security by restricting from where the Server may be contacted. To restrict services to your domain, give the following qmgr directives:

```
Qmgr: set server acl_hosts=*.your_domain
Qmgr: set server acl_host_enable=true
```

Last, activate the Server − Scheduler interaction, i.e. the scheduling of jobs by pbs_sched, by issuing:

```
Qmgr: s s scheduling=true
```

When the attribute **scheduling** is set to true, the Server will call the the job Scheduler, if false the job Scheduler is not called. The value of **scheduling** may also be specified on the pbs_server command line with the −a option.

2.3. Build Details

While the overview gives sufficient information to build a basic PBS system, there are lots of options available to you and custom tailoring that should be done.

2.3.1. Configure Options

The following is detailed information on the options to the configure script.

2.3.1.1. Generic Configure Options

The following are generic configure options that do not affect the functionality of PBS.

--cache-file=file
Cache the system configuration test results in file.
Default: config.cache

--help
Prints out information on the available options.

--no-create
Do not create output files.

--quiet, --silent
Do not print "checking" messages.

--version
Print the version of autoconf that created configure.

--enable-depend-cache
This turns on configure's ability to cache *makedepend* information across runs of configure. This can be bad if the user makes certain configuration changes in rerunning configure, but it can save time in the hands of experienced developers.
Default: disabled

2.3.1.2. Directory and File Names

These options specify where PBS objects will be placed.

--prefix=PREFIX
Install files in subdirectories of PREFIX directory.
Default: **/usr/local**

--exec-prefix=EPREFIX
Install architecture dependent files in subdirectories of EPREFIX.
Default: see PREFIX

--bindir=DIR
Install user executables (commands) in subdirectory DIR.
Default: EPREFIX/bin (**/usr/local/bin**)

--sbindir=DIR
Install System Administrator executables in subdirectory DIR. This includes certain administrative commands and the daemons.
Default: EPREFIX/sbin (**/usr/local/sbin**)

--libdir=DIR
Object code libraries are placed in DIR. This includes the PBS API library, libpbs.a.
Default: PREFIX/lib (**/usr/local/lib**)

--includedir=DIR
C language header files are installed in DIR.
Default: PREFIX/include (**/usr/local/include**)

--mandir=DIR
Install man pages in DIR.
Default: PREFIX/man (**/usr/local/man**)

--srcdir=SOURCE_TREE
PBS sources can be found in directory SOURCE_TREE.
Default: location of the *configure* script.

--x-includes=DIR
X11 header files are in directory DIR.
Default: attempts to autolocate the header files

--x-libraries
X11 libraries are in directory DIR.
Default: attempts to autolocate the libraries

2.3.1.3. Features and Package Options

In general, these options take the following forms:

--disable-FEATURE	Do not compile for FEATURE, same as --enable-FEATURE=no
--enable-FEATURE	Compile for FEATURE
--with-PACKAGE	Compile to include PACKAGE
--without-PACKAGE	Do not compile to include PACKAGE, same as with-PACKAGE=no
--set-OPTION	Set the value of OPTION

For PBS, the recognized --enable/disable, --with/without, and --set options are:

--enable-docs
Build (or not build) the PBS documentation. To do so, you will need the following GNU utilities: groff, gtbl and gpic. Even if this option is not set, the man pages will still be installed.
Default: disabled

--enable-server
Build (or not build) the PBS job server, pbs_server. Normally all components (Commands, Server, Mom, and Scheduler) are built.
Default: enabled

--enable-mom
Build (or not build) the PBS job execution daemon, pbs_mom.
Default: enabled

--enable-clients
Build (or not build) the PBS commands.
Default: enabled

--with-tcl=DIR_PREFIX
Use this option if you wish Tcl based PBS features compiled and the Tcl libraries are not in /usr/local/lib. These Tcl based features include the GUI interface, xpbs. If the following option, --with-tclx, is set, use this option only if the Tcl libraries are not co-located with the Tclx libraries. When set, DIR_PREFIX must specify the absolute path of the directory containing the Tcl Libraries.
Default: if --enable-gui is enabled, then with, Tcl utilities are built; otherwise, without, Tcl utilities are not built.

--with-tclx=DIR_PREFIX
Use this option if you wish the Tcl based PBS features to be based on Tclx. This option implies --with-tcl.
Default: Tclx is not used.

--enable-gui
Build the xpbs GUI. Only valid if --with-tcl is set.
Default: enabled

--set-cc[=ccprog]
Specify which C compiler should be used. This will override the CC environment setting. If only --set-cc is specified, then CC will be set to cc.
Default: gcc (after all, configure is from GNU also)

--set-cflags[=FLAGS]
Set the compiler flags. This is used to set the CFLAGS variable. If only --set-cflags is specified, then CFLAGS is set to "". This must be set to -64 to build 64 bit objects under Irix 6, e.g. --set-cflags=-64. Note, multiple flags, such as -g

and -64 should be enclosed in quotes, e.g. `--set-cflags='-g -64'`
Default: CFLAGS is set to a best guess for the system type.

--enable-debug
Builds PBS with debug features enabled. This allows the daemons to remain attached to standard output and produce vast quantities of messages.
Default: disabled

--set-tmpdir=DIR
Set the tmp directory in which pbs_mom will create temporary scratch directories for jobs. Used on Cray systems only.
Default: **/tmp**

--set-server-home=DIR
Sets the top level directory name for the PBS working directories, PBS_HOME. This directory **MUST reside on a file system which is local to the host** on which any of the daemons are running. That means you must have a local file system on any system where a pbs_mom is running as well as where pbs_server and/or pbs_sched is running. PBS uses synchronous writes to files to maintain state. We recommend that the file system has the same mount point and path on each host, that enables you to copy daemons from one system to another rather than having to build on each system.
Default: **/usr/spool/pbs**

--set-server-name-file=FILE
Set the file name which will contain the name of the default Server. This file is used by the commands to determine which Server to contact. If FILE is not an absolute path, it will be evaluated relative to the value of --set-server-home, PBS_HOME.
Default: server_name

--set-default-server=HOSTNAME
Set the name of the host that clients will contact when not otherwise specified in the command invocation. It must be the primary network name of the host.
Default: the name of the host on which PBS is being compiled.

--set-environ=PATH
Set the path name of the file containing the environment variables used by the daemons and passed to the jobs. For AIX based systems, we suggest setting this option to /etc/environment. Relative path names are interpreted relative to the value of --set-server-home, PBS_HOME.
Default: the file pbs_environment in the directory PBS_HOME.

For a discussion of this file and the environment, see section **6.1.1. Internal Security**. You may edit this file to modify the path or add other environmental variables.

--enable-plock-daemons=WHICH
Enable daemons to lock themselves into memory to improve performance. The argument WHICH is the logical-or of 1 for pbs_server, 2 for pbs_sheduler, and 4 for pbs_mom (7 is all three daemons). This option is recommended for Unicos systems. It must **not** be used for AIX systems.
Default: disabled.

Note, this feature uses the plock() system call which is not available on Linux and bsd derived systems. Before using this feature, check that plock(3) is avaible on the system.

--enable-syslog
Enable the use of syslog for error reporting. This is in addition to the normal PBS logs.

Default: disabled.

--set-sched=TYPE
Set the Scheduler (language) type. If set to c, a C based Scheduler will be compiled. If set to tcl, a Tcl based Scheduler will be used. If set to basl, a BAtch Scheduler Language Scheduler will be generated. If set to no, no Scheduler will be compiled, jobs will have to be run by hand.
Default: c

--set-sched-code=PATH
Sets the name of the file or directory containing the source for the Scheduler. This is only used for C and BaSL Schedulers, where --set-sched is set to either c or basl. For C Schedulers, this should be a directory name. For BaSL Schedulers, it should be file name ending in .basl. If the path is not absolute, it will be interpreted relative to SOURCE_TREE/src/schedulers.SCHED_TYPE/samples. For example, if --set-sched is set to basl, then set --set-sched-code to fifo_byqueue.basl.
Default: fifo (C based Scheduler)

--enable-tcl-qstat
Builds qstat with the Tcl interpreter extensions. This allows site and user customizations. Only valid if --with-tcl is already present.
Default: disabled

--set-tclatrsep=CHAR
Set the character to be used as the separator character between attribute and resource names in Tcl/Tclx scripts.
Default: "."

--set-mansuffix=CHAR
Set the character to be used as the man page section suffix letter. For example, the qsub man page is installed as man1/qsub.1B. To install without a suffix, --set-mansuffix="".
Default: "B"

--set-qstatrc-file=FILE
Set the name of the file that qstat will use if there is no *.qstatrc* file in the user's home directory. This option is only valid when --enable-tcl-qstat is set. If FILE is a relative path, it will be evaluated relative to the PBS Home directory, see --set-server-home.
Default: PBS_HOME/qstatrc

--with-scp
Directs PBS to attempt to use the *Secure Copy Program*, *scp*, when copying files to or from a remote host. This applies for delivery of output files and stage-in/stage-out of files. If scp is to used and the attempt fails, PBS will then attempt the copy using rcp in case that scp did not exist on the remote host.

For local delivery, "/bin/cp -r" is always used. For remote delivery, a varient of rcp is required. The program must always provide a non-zero exit status on any failure to deliver files. This is not true of all rcp implementation, hence a copy of a known good rcp is included in the source, see mom_rcp. More information can be found in section **7.5 Delivery of Output Files**.
Default: sbindir/pbs_rcp (from the mom_rcp source directory) is used, where sbindir is the value from --sbindir.

--enable-shell-pipe
When enabled, pbs_mom passes the name of the job script to the top level shell via a pipe. If disabled, the script file is the shell's standard input file. See section **7.3 Shell Invocation** for more information.
Default: enabled

--enable-sp2

Turn on special features for the IBM SP. This option is only valid when the PBS machine type is aix4. The PBS machine type is automatically determined by the configure script.
Default: disabled

With PSSP software before release 3.1, access to two IBM supplied libraries, libjm_client.a and libSDR.a, are required. These libraries are installed when the ssp.clients fileset in installed, and PBS will expect to find them in the normal places for libraries.

With PSSP 3.1 and later, libjm_client.a and libSDR.a are not required, instead libswitchtbl.a is used to load and unload the switch. See the discussion under the sub-section **IBM SP** in the section **2.4 Machine Dependent Build Instructions**.

--enable-nodemask

Build PBS with support for SGI Origin2000 nodemask. Requires Irix 6.x.
Default: disabled

--enable-srfs

This option enables support for Session Reservable File Systems. It is only valid on Cray systems with the NASA modifications to support Session Reservable File System, SRFS.
Default: disabled

--enable-array

Setting this under Irix 6.x forces the use of SGI Array Session tracking. Enabling this feature is recommanded if MPI jobs use the Array Services Daemon. The PBS machine type is set to irix6array. Disabling this option forces the use of POSIX session ids. See section **2.4.5 SGI Systems Running IRIX 6**.
Default: Autodetected by existence and content of /etc/config/array.

2.3.2. Make File Targets

The follow target names are applicable for make:

all	The default target, it compiles everything.
build	Same as all.
depend	Builds the header file dependency rules.
install	Installs everything.
clean	Removes all object and executable program files in the current subtree.
distclean	Leaves the object tree very clean. It will remove all files that were created during a build.

possible to compile or install a piece, such as Mom, by changing to the appropriate subdirectory and typing “make” or “make install”.

2.4. Machine Dependent Build Instructions

There are a number of possible variables that are only used for a particular type of machine. If you are not building for one of the following types, you may ignore this section.

2.4.1. Cray Systems

2.4.1.1. Cray C90, J90, and T90 Systems

On the traditional Cray systems such as the C90, PBS supports Unicos versions 8, 9 and 10.

Because of the fairly standard usage of the symbol **TARGET** within the PBS makefiles, when building under Unicos you cannot have the environment variable TARGET defined. Otherwise, it is changed by Unicos's make to match the makefile value, which confuses the compiler. If set, type `unsetenv TARGET` before making PBS.

If your system supports the Session Reservable File System enhancement by NASA, run configure with the `--enable-srfs` option. If enabled, the Server and Mom will be compiled to have the resource names *srfs_tmp*, *srfs_big*, *srfs_fast*, and *srfs_wrk*. These may be used from **qsub** to request SRFS allocations. The file `/etc/tmpdir.conf` is the configuration file for this. An example file is:

```
# Shell environ var     Filesystem
TMPDIR
BIGDIR                  /big/nqs
FASTDIR                 /fast/nqs
WRKDIR                  /big/nqs
```

The directory for TMPDIR will default to that defined by JTMPDIR in Unicos's /usr/include/tmpdir.h.

Without the SRFS mods, Mom under Unicos will create a temporary job scratch directory. By default, this is placed in /tmp. The location can be changed via `--set-tmpdir=DIR`.

2.4.1.2. Unicos 10 with MLS

If you are running Unicos MLS, required in Unicos 10.0 and later, the following action is required after the system is built and installed. Mom updates **ue_batchhost** and **ue_batchtime** in the UDB for the user. In an MLS system, Mom must have the security capability to write the protected UDB. To grant this capability, change directory to wherever pbs_mom has been installed and type:

```
spset -i 16 -j daemon -k exec pbs_mom
```

You, the administrator, must have capabilities **secadm** and **class 16** to issue this command. You use the setucat and setucls commands to get to these levels if you are authorized to do so. The UDB **reclsfy** permission bit gives a user the proper authorization to use the spset command.

WARNING

There has been only limited testing in the weakest of MLS environments, problems may appear because of differences in your environment.

2.4.1.3. Cray T3E Systems

For Cray T3E systems, **TBD**.

2.4.2. Digital Unix

The following is the recommend value for CFLAGS when compiling PBS under Digital UNIX 4.0D: `--set-cflags="-g-std0"` that is s-t-d-zero.

2.4.3. HPUX

The following is the recommend value for CFLAGS when compiling PBS under HP UX: `--set-cflags="-g-Ae"`

2.4.4. IBM Workstations

PBS supports IBM workstations running AIX 4.x. When man pages are installed in *mandir*, the default man page file name suffix, "**B**", must be removed. Currently, this must be done by hand. For example, change man3/qsub.3B to man3/qsub.3.

Do not use the configure option `--enable-plock`. It will crash the system by using up all of memory.

2.4.5. IBM SP

Every thing under IBM Workstation section above applies to the IBM SP. Be sure to read the section **3.2 Multiple Execution Systems** before configuring the Server.

Set special SP-2 option, `--enable-sp2`, to compile special code to deal with the SP high speed switch.

If the library *libswitchtbl.a* is not detected, it is assumed that you are running with PSSP software prior to 3.1. In this case, the IBM poe command sets up the high speed switch directly and PBS interfaces with the IBM Resource (Job) Manager to track which nodes jobs are using. PBS requires two libraries, libjm_client.a and libSDR.a, installed with the ssp.clients fileset.

If the library libswitchtbl.a is detected, it is assumed you are running with PSSP 3.1 or later software. PBS takes on the responsibility of loading the high speed switch tables to provide node connectivity.

> **Important Note**
> At this time, only the switch Window ID of zero is used, limiting usage of any node to a single task regardless of the number of processor in the node.

With PSSP 3.1, two additional items of information must be passed to the job, the switch window id, and a *job key* which authorizes a process to use the switch. As poe does not pass this information to the processes it creates, an underhanded method had to be created to present them to the job. Two new programs are compiled and installed into the `bindir` directory, *pbspoe* and *pbspd*.

pbspoe is a wrapper around the real poe command. pbspoe must be used by the user in place of the real poe. pbspoe modifies the command arguments and invokes the real poe, which is assumed to be in /usr/lpp/ppe.poe/bin. If a user specifies:

```
pbspoe a.out args
```

it is converted to the effective command:

```
/usr/lpp/ppe.poe/bin/poe pbspd job_key a.out args -hfile $PBS_NODE-
FILE
```

PBS_NODEFILE of course contains the nodes allocated by pbs. The pbs_mom on those nodes have loaded the switch table with the user's uid, the job key, and a window id of zero.

pbspd places the job key into the environment as **MP_PARTITION**, and the window id as **MP_MPI_NETWORK**. pbspd then exec-s a.out with the remaining arguments.

If the user specified a command file to pbspoe with *-cmdfile file*, then pbspoe prefixes each line of the command file with **pbspd job_key** and copies it into a temporary file. The temporary file is passed to poe instead of the user's file.

pbspoe also works with /usr/lpp/ppe.poe/bin/pdbx and /usr/lpp/ppe.poe/bin/xpdbx. This substitution is done to make the changes as tranparent to the user as possible.

> **Note**
> Not all poe arguments or capabilities are supported. For example, poe job steps are not supported.

For transparent usage, it is **necessary** that after PBS is installed that you perform these additional steps:

1. Remove IBM's poe, pdbx, and xpbsdx from /usr/bin or any directory in the user's normal path. Be sure to leave the commands in /usr/lpp/ppe.poe/bin which should not be in the user's path, or if in the user's path must be after /usr/bin.
2. Create a link named /usr/bin/poe pointing to `{bindir}/pbspoe`. Also make links for /usr/bin/pdbx and /usr/bin/xpbdx which point to `{bindir}/pbspoe.`.
3. Be sure that pbspd is installed in a directory in the user's normal path on each and every node.

2.4.6. SGI Workstations Running IRIX 5

If, and only if, your system is running Irix 5.3, you will need to add `-D_KMEMUSER` to **CFLAGS** because of a quirk in the Irix header files.

2.4.7. SGI Systems Running IRIX 6

If built for Irix 6.x, pbs_mom will track which processes are part of a PBS job in one of two ways depending on the existence of the Array Services Daemon, arrayd, as determined by /etc/config/array. If the daemon is not configured to run, pbs_mom will use POSIX session numbers. This method is fine for workstations and multiprocessor boxes not using SGI's mpirun command. The PBS machine type (PBS_MACH) is set to `irix6`. This mode can also be forced by setting `--disable-array`.

Where arrayd and mpirun are being used, the tasks of a parallel job are started through requests to arrayd and hence are not part of the job's POSIX session. In order to relate processes to the job, the SGI Array Session Handle (ASH) must be used. This feature is enabled when /etc/config/array contains `on` or may be forced by setting the configure option `--enable-array`. The PBS machine type (PBS_MACH) is set to `irix6array`

IRIX 6 supports both 32 and 64 bit objects. In prior versions, PBS was typically built as a 32 bit object. Irix 6.4 introduced system supported checkpoint/restart; PBS will include support for checkpoint/restart if the file */usr/lib64/libcpr.so* is detected during the build process. To interface with the SGI checkpoint/restart library, PBS must be built as a 64 bit object. Add `-64` to the **CFLAGS**. This can be done via the configure option `--set-cflags=-64`

WARNING

Because of changes in structure size, PBS will not be able to recover any server, queue, or job information recorded by a PBS built with 32 bit objects, or vice versa. Please read section 6.5 of the Admin Guide entitled *Installing an Updated Batch System* for instructions on dealing with this incompatibility.

If libcpr.so is not present, PBS may be built as either a 32 bit or a 64 bit object. To build as 32 bit, add `-n32` instead of -64 to **CFLAGS**.

2.4.8. FreeBSD and NetBSD

There is a problem with FreeBSD up to at least version 2.2.6. It is possible to lose track of which session a set of processes belongs to if the session leader exits. This means that if the top shell of a job leaves processes running in the background and then exits, Mom will not be able to find them when the job is deleted. This should be fixed in a future version.

2.4.9. Linux

Redhat version 4.x and 5.x are supported.

2.4.10. SUN Running SunOS

The native SunOS C compiler is not ANSI and cannot be used to build PBS. GNU gcc is recommended.

3. Batch System Configuration

Now that the system has been built and installed, the work has just begun. The Server and Moms must be configured and the scheduling policy must be implemented. These items are closely coupled. Managing which and how many jobs are scheduled into execution can be done in several methods. Each method has an impact on the implementation of the scheduling policy and server attributes. An example is the decision to schedule jobs out of a single pool (queue) or divide jobs into one of multiple queues each of which is managed differently. More on this type of discussion is covered under the Chapter **4. Scheduling Policies**.

3.1. Single Execution System

If you are installing PBS on a single system, you are ready to configure the daemons and start worrying about your scheduling policy. We still suggest that you read section **3.2.3 Where Jobs May Be Run** and then continue with section **3.3 Network Addresses**. No nodes file is needed.

If you wish, the PBS Server and Scheduler, pbs_server and pbs_sched, can run on one system and jobs execute on another. This is trivial case of multiple execution systems discussed in the next section. We suggest that you read it. If you are running the default Scheduler, fifo, you will need a nodes file with one entry, the name of the host with Mom on it, appendix with `:ts`. If you write your own Scheduler, it can told in ways other than the nodes file on which host jobs should be run.

3.2. Multiple Execution Systems

If you are running on more than a single computer, you will need to install the execution daemon (pbs_mom) on each system where jobs are expected to execute. If you are running the default scheder, fifo, you will need a nodes file with one entry for each execution host. The entry is the name of the host with Mom on it, appendix with `:ts`. Again, if you write your own Scheduler, it can be told in ways other than the Server's nodes file on which hosts jobs could be run.

3.2.1. Installing Mulitple Moms

There are four ways in which a Mom may be installed on each of the various execution hosts.

1. The first method is to do a full install of PBS on each host. While this works, it is a bit wasteful.
2. The second way is to rerun configure with the following options: `--disable-server --set-sched=no`. You may also choose to `--disable-clients`, but users often use the PBS commands within a job script. You will then need to recompile and then do an install on each execution host.
3. The third way is to do an install of just Mom (and maybe the commands) on each system. If the system will run the same binaries as where PBS was compiled, cd down to src/mom and `make install` as root. To install the commands cd ../cmds and again `make install`. If the system requries recompiling, do so at the top level to recompile the libraries and then proceed as above.
4. The fourth requires that the the system be able to execute the existing binaries and that the directories *sbindir* and *bindir* in which the PBS daemons and commands were placed in the initial full build be available on each host. These directories, unlike the PBS_HOME directory can be on a network file system.

If the target tree is accessible on the host, as root execute the following commands on each execution host:

```
sh {target_tree}/buildutils/pbs_mkdirs mom
sh {target_tree}/buildutils/pbs_mkdirs aux
sh {target_tree}/buildutils/pbs_mkdirs default
```

This will build the required portion of PBS_HOME on each host.

If the target tree is not accessible, copy the pbs_mkdirs shell script to each execution host and again as root, execute it with the above operands.

You will now need to declare the execution hosts to the pbs_server daemon as explained in the next section.

3.2.2. Declaring Nodes

In PBS systems prior to release 2.0, the system had to be specifically built and configured to support exclusive nodes. A single pbs_mom was given a list of nodes. When jobs were run, the top level shell for all jobs ran on the host where that Mom lived. Mom allocated nodes to the job and provided the job with a file containing the list of nodes allocated to it. If the PBS batch system was supporting one or more timeshared hosts, only the Job Scheduler knew of those hosts. It directed where the Server should send the job for execution.

In this version of PBS allocation of nodes is handled by the Server instead of Mom. Each node to be used by a job must have its own copy of Mom running on it. If only timeshared hosts are to be served by the PBS batch system, then as before, the Job Scheduler must direct where the job should be run. If unspecified, the Server will execute the job on the host where it is running. See the next section for full details.

If nodes are to be allocated *exclusively* or *temporarily-shared*, a list of the nodes must be specified to the Server. This list may also contain timeshared nodes. Nodes marked as timeshared will be listed by the Server in a node status report along with the other nodes. However, the Server will **not attempt to allocate them** to jobs. The presence of timeshared nodes in the list is solely as a convenience to the Job Scheduler and other programs, such as xpbsmon.

The node list is given to the Server in a file named `nodes` in the Server's home directory `PBS_HOME/server_priv`. This is a simple text file with the specification of a single node per line in the file. The format of each line in the file is:

```
node_name[:ts] [property ...]
```

The node name is the network name of the node (host name). The optional `:ts` appended to the name indicates that the node is a timeshared node. Zero or more properties may be specified. Each item on the line must be separated by white space. Comment lines may be included if the first non-white space character is the pound sign '#'. Properties are arbitrary strings make up of alphanumber characters whose first character must be alphabetic.

This is the same format used by Mom in earlier versions with the addition of the :ts suffix. The following is an example of a possible nodes file:

```
# The first set of nodes are cluster nodes.
# Note that the properties are provided to group
# certain nodes together.
curly stooge odd
moe   stooge even
larry stooge even
harpo marx odd
groucho marx odd
chico marx even
# And for fun we throw in one timeshared node.
```

```
chaplin:ts
```

After the pbs_server is started with a nodes file containing at least one node definition, the list of nodes may be altered via the qmgr command.

Add nodes: **Qmgr:** `create node` *node_name* `[attributes=values]`
where attributes are `state`, `properties`, and `ntype`. The possible values for the attributes are:

state which can take the values: `free`, `down`, or `offline`.

properties
which can be any string or comma separated strings which must be enclosed in quotes. For example: `properties="green,blue,yellow"`

ntype what can take the values: **Qmgr:** `cluster` **or** `time-shared`**.**

Delete nodes:
Qmgr: `delete node` *node_name*

Modify nodes:
`set node` *node_name* `[attributes=values]`
where attributes are the same as for create.

3.2.3. Where Jobs May Be Run

Where jobs may be or will be run is determined by an interaction between the Scheduler and the Server. This interaction is effected by the existence of the *nodes* file.

3.2.3.1. No Node File

If a nodes file does not exist, the Server only directly knows about its own host. It assumes that jobs may be executed on it. When told to run a job without a specific execution host named, it will default to its own host. Otherwise, it will attempt to execute the job where directed in the Run Job request. Typically the job Scheduler will know about other hosts because it was written that way at your site. The Scheduler will direct the Server where to run the job.

The default fifo Scheduler depends on the existence of a node file if more than one host is to be scheduled. Any or all of the nodes contained in the file may be time shared hosts with the appended ":ts".

3.2.3.2. Node File Exists

If a nodes file exists, then the following rules come into play

1. If a specific host is named in the Run Job request and the host is specified in the nodes file as a *timeshared* host, the Server will attempt to run the job on that host.
2. If a specific host is named in the Run Job request and the named node is not in the nodes file as a timeshared host or if there are multiple nodes named in the Run Job request, then the Server attempts to allocate the named *cluster* node or nodes to the job. All of the named nodes must appear in the Server's nodes file. If the allocation succeeds, the job is run directly on the first of the nodes allocated.
3. If no location was specified on the Run Job request, but the job requests nodes, then cluster nodes which match the request are allocated if possible. If the allocation succeeds, the job is run on the node allocated to match the first specification in the node request. Note, the Scheduler may modify the job's original node request, see the job attribute **neednodes**.

4. If the server attribute **default_node** is is set, its valuse is used. If this matches the name of a time-shared node, the job is run on that node. If the value of default_node can be mapped to a set of one or more free cluster nodes, they are allocated to the job.
5 If default_node is not set, and at least one time-shared node is defined, that node is used. If more than one is defined, one is selected for the job, but which is not really predictable.
6. The last choice is to act as if the job has requested `1#shared`. The job has allocated to it any existing job-shared node, or if none exist, then a free node is allocated as job-shared.

What all the above means can be boiled down into the following set of guidelines:

- If the batch system consists of a single timeshared host on which the Server and Mom are running, no problem − all the jobs run there. The Scheduler only needs to say which job it wants run.
- If you are running a timeshared complex with *one* or more back-end hosts, where Mom is on a different host than is the Server, then load balancing jobs across the various hosts is a matter of the Scheduler determining on which host to place the selected job. This is done by querying the resource monitor side of Mom using the resource monitor API - the addreq() and getreq() calls. The Scheduler tells the Server where to run each job.
- If your cluster is made up of cluster nodes and you are running distributed (multiple node) jobs, as well as serial jobs, the Scheduler typically uses the *Query Resource* or *Avail* request to the Server for each queued job under consideration. The Scheduler then selects one of the jobs that the Server replied could run, and directs that the job should be run. The Server will then allocate the nodes to the job. By setting the Server attribute **default_node** set to one temporarily-shared node, `1#shared`, jobs which do not request nodes will be placed together on a few temporarily-shared nodes.
- If you have a batch system supporting both cluster nodes and one timeshared node, the situation is like the above, only you may wish to change **default_node** to point to the timeshared host. Jobs that do not ask for nodes will end up running on the timeshared host.
- If you have a batch system supporting both cluster nodes and multiple time shared hosts, you have a complex system which requires a smart Scheduler. The Scheduler must recognize which jobs request nodes and use the *Avail* request to the Server. It must also recognize which jobs are to be load balanced among the timeshared hosts, and provide the host name to the Server when directing that the job be run. The supplied *fifo* Scheduler has this capability.

3.3. Network Addresses and Ports

PBS makes use of fully qualified host names for identifying the jobs and their location. A PBS batch system is known by the host name on which the Server, pbs_server, is running. The name used by the daemons, or used to authenicate messages is the **canonical** host name. This name is taken from the primary name field, `h_name`, in the structure returned by the library call gethostbyaddr(). According to our understanding of the IETF RFCs, this name must be fully qualified and consistent for any IP address assigned to that host.

The three daemons and the commands will attempt to use /etc/services to identify the standard port numbers to use for communication. The port numbers need not be below the magic 1024 number. The service names that should be added to /etc/services are

```
pbs             15001/tcp           # pbs server (pbs_server)
pbs_mom         15002/tcp           # mom to/from server
```

```
pbs_resmom    15003/tcp          # mom resource management requests
pbs_resmom    15003/udp          # mom resource management requests
pbs_sched     15004/tcp          # scheduler
```

The numbers listed are the default number used by this version of PBS. If you change them, be careful to use the same numbers on all systems. Note, the name *pbs_resmom* is a carry-over from early versions of PBS when separate daemons for job execution (pbs_mom) and resource monitoring (pbs_resmon). The two functions were combined into pbs_mom though the term "resmom" might be found referring to the combined functions.

If the services cannot be found in /etc/services, the PBS components will default to the above listed numbers.

If the Server is started with an non-standard port number, see -p option in the pbs_server(8) man page, the Server "name" becomes *host_name.domain:port*, where port is the numeric port number being used. See the discussion of **Alternate Test Systems**, section 6.4.

3.4. Starting Daemons

All three of the daemon processes, Server, Scheduler and Mom, must run with the real and effective uid of root. Typically, the daemons are started from the systems boot files, e.g. /etc/rc.local. However, it is recommended that the Server be brought up "by hand" the first time and configured before being run at boot time.

3.4.1. Starting Mom

Mom should be started at boot time. Typically there are no requried options. It works best if Mom is started before the Server so she will be ready to respond to the Server's "are you there?" ping. Start Mom with the line

```
{sbindir}/pbs_mom
```

in the /etc/rc2 or equivalent boot file.

If Mom is taken down and the host system continues to run, Mom should be restarted with the `-r` option. This directs Mom to kill off any jobs which were left running. See the ERS for a full explanation.

By default, Mom will only accept connections from a privileged port on her system, either the port associated with "localhost" or the name returned by gethostname(2). If the Server or Scheduler are running on a different host, the host name(s) must be specified in Mom's configuration file. See the -c option on the pbs_mom(8B) man page and in the Admin Guide, see sections **3.6 Configurating the Execution Server, pbs_mom** for more information on the configuration file.

Should you wish to make use of the prologue and/or epilogue script features, please see section 6.2 "Job Prologue/Epilogue Scripts".

3.4.2. Starting the Server

The initial run of the Server or any first time run after recreating the home directory must be with the `-t create` option. This option directs the Server to create a new server database. This is best done by hand. If a database is already present, it is discarded after receiving a positive validation response. At this point it is necessary to configure the Server. See the section **3.5 Server Configuration**. The create option leaves the Server in a "idle" state. In this state the Server will not contact the Scheduler and jobs are not run, except manually via the qrun(1B) command. Once the Server is up, it can be placed in the "active" state by setting the Server attribute `scheduling` to a value of true:

```
qmgr -c "set server scheduling=true"
```

The value of `scheduling` is retained across Server terminations/starts.

After the Server is configured it may be placed into service. Normally it is started in the system boot file via a line such as:

```
{sbindir}/pbs_server
```

The `-t start_type` option may be specified where `start_type` is one of the options specified in the ERS (and the pbs_server man page). The default is `warm`. Another useful option is the `-a true|false` option. This turns on|off the invocation of the PBS job Scheduler.

3.4.3. Starting the Scheduler

The Scheduler should also be started at boot time. Start it with an entry in the /etc/rc2 or equivalent file:

```
{sbindir}/pbs_sched [options]
```

There are no required options for the default fifo scheduler. Typically the only required option for the BaSL based Scheduler is the `-c config_file` option specifying the configuration file. For the Tcl based Scheduler, the option is used to specify the Tcl script to be called.

3.5. Configuring the Job Server, pbs_server

Server management consist of configuring the Server attributes and establishing queues and their attributes. Unlike Mom and the Job Scheduler, the Job Server (pbs_server) is configured while it is running, except for the nodes file. Configuring server and queue attributes and creating queues is done with the qmgr(1B) command. This must be either as root or as a user who has been granted PBS Manager privilege as shown in the last step in the **Build Overview** section of this guide. Exactly what needs to be set depends on your scheduling policy and how you chose to implement it. The system needs at least one queue established and certain server attributes initialized.

The Server attributes are discussed in section 2.4 of the ERS. The following are the "minimum required" server attributes and the recommended attributes. For the sake of examples, we will assume that your site is a sub-domain of a large network and all hosts at your site have names of the form:

host.**foo.bar.com**

and the batch system consists of a single large machine named **big.foo.bar.com**.

3.5.1. Server Configuration

The following attributes are required or recommended. They are set via the *set server* (s s) subcommand to the *qmgr* (1B) command.

Not all of the Server attributes are discussed here, only what is needed to get a reasonable system up and running. See the *pbs_server_attributes* man page for a complete list.

3.5.1.1. Required Server Attributes

`default_queue`
: Declares the default queue to which jobs are submitted if a queue is not specified on the qsub(1B) command. The queue must be created first. Example:

```
Qmgr: c q dque queue_type=execution
Qmgr: s s default_queue=dque
```

3.5.1.2. Recommended Server Attributes

`acl_hosts` A list of hosts from which jobs may be submitted. For example, if you wish to allow all the systems on your sub-domain plus one other host, boss, at headquarters to submit jobs, then set:

```
Qmgr: s s acl_hosts=*.foo.bar.com,boss.hq.bar.com
```

`acl_host_enable`
: Enables the Server's host access control list, see above.

Qmgr: `s s acl_host_enable=true`

`default_node`
Defines the node on which jobs are run if not otherwise directed. Please see section 3.2.3 **Where Jobs May be Run** for a discussion of how to set this attibute depending on your system. The default value (also the value assumed if the attribute is unset) is `1#shared`.

Qmgr: `s s default_node=big`

Note, the value may be specified as either `big` or `big.foo.bar.com`. If there is a node file, the value must match exactly the name specified in the node file. I.e. big in both places or big.foo.bar.com in both places.

`managers` Defines which users, at a specified host, are granted batch system administrator privilege. For example, to grant privilege to "me" at all systems on the subdomain and "sam" only from this system, big, then:

Qmgr: `s s managers=me@*.foo.bar.com,sam@big.foo.bar.com`

`operators` Defines which users, at a specified host, are granted batch system operator privilege. Specified as are the managers.

`resources_cost`
If you are planning to use the "synchronous job starts" feature across multiple execution hosts, you may wish to establish arbitrary costs for various resources on each system. See the ERS section on **Synchronize Job Starts** (section 3.2.2).

`resources_defaults`
This attribute establishes the resource limits assigned to jobs that were submitted without a limit and for which there are no queue limits. It is important that a default value be assigned for any resource requirement used in the scheduling policy. See the *pbs_resources_** man page for your system type (* is irix6, linux, solaris5, ...).

Qmgr: `s s resources_defaults.cput=5:00`
Qmgr: `s s resources_defaults.mem=4mb`

`resources_max`
This attribute sets the maximum amount of resources which can be used by a job entering any queue on the Server. This limit is checked only if there is not a queue specific resources_max attribute defined for the specific resource.

`system_cost` See resources_cost.

3.5.2. Queue Configuration

There are two types of queues defined by PBS, routing and execution. A routing queue is a queue used to move jobs to other queues which may even exist on different PBS Servers. Routing queues are similar to the old NQS pipe queues. 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.

A Server may have multiple queues of either or both types. A Server must have at least one queue defined. Typically it will be an execution queue; jobs cannot be executed while residing in an routing queue.

Queue attributes fall into three groups: those which are applicable to both types of queues, those applicable only to execution queues, and those applicable only to routing queues. If an "execution queue only" attribute is set for a routing queue, or vice versa, it is simply ignored by the system. However, as this situation might indicate the administrator made a mistake, the Server will issue a warning message about the conflict. The same message will be issued if the queue type is changed and there are attributes that do not apply to the new type.

Not all of the Queue Attributes are discussed here, only what is needed to get a reasonable system up and running. See the *pbs_queue_attributes* man page for a complete list.

3.5.2.1. Required Attributes for All Queues

`queue_type` Must be set to either `execution` or `routing` (e or r will do). The queue type must be set before the queue can be enabled. If the type conflicts with certain attributes which are valid only for the other queue type, the set request will be rejected by the Server.

Qmgr: `s q dque queue_type=execution`

`enabled` If set to true, jobs may be enqueued into the queue. If false, jobs will not be accepted.

Qmgr: `s q dque enabled=true`

`started` If set to true, jobs in the queue will be processed, either routed by the Server if the queue is a routing queue or scheduled by the job Scheduler if an execution queue.

Qmgr: `s q dque started=true`

3.5.2.2. Required Attributes for Routing Queues

`route_destinations`

List the local queues or queues at other Servers to which jobs in this routing queue may be sent. For example:

Qmgr: `s q routem route_destinations=dque,overthere@another.foo.bar.com`

3.5.2.3. Recommended Attributes for All Queues

`resources_max`

If you chose to have more than one execution queue based on the size or type of job, you may wish to establish maximum and minimum values for various resource limits. This will restrict which jobs may enter the queue. A routing queue can be established to "feed" the execution queues and jobs will be distributed by those limits automatically.

A resources_max value defined for a specific resource at the queue level will override the same resource resources_max defined at the Server level. Therefore, it is possible to define a higher as well as a lower value for a queue limit than the Server's corresponding limit. If there is no maximum value declared for a resource type, there is no restriction on that resource. For example:

```
s q dque resources_max.cput=2:00:00
```

places a restriction that no job requesting more than 2 hours of cpu time will be allowed in the queue. There is no restriction on the memory, **mem**, limit a job may request.

`resources_min`

Defines the minimum value of resource limit specified by a job before the job will be accepted into the queue. If not set, there is no minimum restriction.

3.5.2.4. Recommended Attributes for Execution Queues

`resources_default`

Defines a set of default values for jobs entering the queue that did not specify certain resource limits. There is a corresponding server attribute which sets a default for all jobs.

The limit for a specific resource usage is established by checking various job, queue, and server attributes. The following list shows the attributes and their

order of precedence:

1. The job attribute Resource_List, i.e. what was requested by the user.
2. The queue attribute resources_default.
3. The Server attribute resources_default.
4. The queue attribute resources_max.
5. The Server attribute resources_max.

* Under Unicos, a user supplied value must be within the system's User Data Base, UDB, limit for the user. If the user does not supply a value, the lower of the defaulted value from the above list and the UDB limit is used.

Please note, an *unset* resource limit for a job is treated as an *infinite* limit.

3.5.2.5. Selective Routing of Jobs into Queues

Often it is desirable to route jobs to various queues on a Server, or even between Servers, based on the resource requirements of the jobs. The queue *resources_min* and *resources_max* attributes discussed above make this selective routing possible. As an example, let us assume you wish to establish two execution queues, one for short jobs of less than 1 minute cpu time, and the other for long running jobs of 1 minute or longer. Call them **short** and **long**. Apply the resources_min and resources_max attribute as follows:

```
Qmgr: set queue short resources_max.cput=59
Qmgr: set queue long  resources_min.cput=60
```

When a job is being enqueued, it's requested resource list is tested against the queue limits: `resources_min <= job_requirement <= resources_max`. If the resource test fails, the job is not accepted into the queue. Hence, a job asking for 20 seconds of cpu time would be accepted into queue **short** but not into queue **long**. Note, if the min and max limits are equal, only that exact value will pass the test.

You may wish to set up a routing queue to feed jobs into the queues with resource limits. For example:

```
Qmgr: create queue feed queue_type=routing
Qmgr: set queue feed route_destinations="short,long"
Qmgr: set server default_queue=feed
```

A job will end up in either **short** or **long** depending on its cpu time request.

You should always list the destination queues in order of the most restrictive first as the first queue which meets the job's requirements will be its destination (assuming that queue is enabled). Extending the above example to three queues:

```
Qmgr: set queue short resources_max.cput=59
Qmgr: set queue long  resources_min.cput=1:00,resources_max.cput=1:00:00
Qmgr: create queue verylong queue_type=execution
Qmgr: set queue feed route_destinations="short,long,verylong"
```

A job asking for 20 minutes (20:00) of cpu time will be placed into queue **long**. A job asking for 1 hour and 10 minutes (1:10:00) will end up in queue **verylong** by default.

Caution, if a test is being made on a resource as shown with cput above, and a job does not specify that resource item (it does not appear in the `-l resource=value` list on the qsub command, the test will pass. In the above case, a job without a cpu time limit will be allowed into queue **short**. For this reason, together with the fact that an unset limit is considered to be an infinite limit, you may wish to add a default value to the queues or to the Server. Either

```
Qmgr: set queue short resources_default.cput=40
```

or

```
Qmgr: set server resources_default.cput=40
```

will see that a job without a cpu time specification is limited to 40 seconds. A resources_default attribute at a queue level only applies to jobs in that queue. Be aware of

two facts:

1. If a default value is assigned, it is done so after the tests against min and max.
2. Default values assigned to a job from a queue `resources_default` are not carried with the job if the job moves to another queue. Those resource limits becomes unset as when the job was specified. If the new queue specifies default values, those values are assigned to the job while it is in the new queue.
3. Server level default values are applied if there is no queue level default.

In the above example, a default attibute should be applied to either at the server level or at the routing queue level. or

Minimum and maximum queue limits work with numeric valued resources, including time and size values. Generally, they do not work with string valued resources because of character comparison order. However, setting the min and max to the same value to force an exact match will work even for string valued resources. For example,

```
Qmgr: set queue big resources_max.arch=unicos8
Qmgr: set queue big resources_min.arch=unicos8
```

can be used to limit jobs entering queue **big** to those specifying `arch=unicos8`. Again, remember that if `arch` is not specified by the job, the tests pass automatically and the job will be accepted into the queue.

It is possible to set limits on queues (and the Server) as to how many nodes a job can request. The *nodes* resource itself is a text string and difficult to limit. However, two additional Read-Only resources exist for jobs. They are *nodect* and *neednodes*. Nodect (node count) is set by the Server to the integer number of nodes desired by the user as declared in the "nodes" resource specification. That declaration is parsed and the resulting total number of nodes is set in nodect. This is useful when an administrator wishes to place an integer limit, *resources_min* or *resources_max*, on the number of nodes used by a job entering a queue.

Based on the earlier example of declaring nodes, if a user requested the following nodes, see section **7.2 Parallel Jobs** for more information:

```
3:marx+2:stooge
```

nodect would be set to 5 (3+2). Neednodes is initially set by the Server to the same value as nodes. Neednodes may be modified by the job Scheduler for special policies. The contents of neednodes determines which nodes are actually assigned to the job. Neednodes is visible to the administrator but not to an unprivileged user.

If you wish to set up a queue default value for "nodes" (a value to which the resource is set if the user does not supply one), corresponding default values must be set for "nodect" and "neednodes". For example

```
Qmgr: set queue foo resources_default.nodes=1
Qmgr: set queue foo resources_default.nodect=1
Qmgr: set queue foo resources_default.neednodes=1
```

Minimum and maximum limits are set for "nodect" only. For example:

```
Qmgr: set queue foo resources_min.nodect=1
Qmgr: set queue foo resources_max.nodect=15
```

Minimum and maximum values must **not** be set for nodes or neednodes as those are string values.

3.5.3. Recording Server Configuration

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

```
qmgr -c "print server" > /tmp/server.con
```

will record in the file server.con the qmgr subcommands required to recreate the current configuration including the queues. The commands could be feed back into qmgr via standard input:

```
qmgr < /tmp/server.con
```

3.6. Configuring the Execution Server, pbs_mom

Mom is configured via a configuration file which she reads at initialization time and when sent the SIGHUP signal. This file is described in the pbs_mom(8) man page as well as in the following section.

If the -c option is not specified when Mom is run, she will open *PBS_HOME/mom_priv/config* if it exists. If it does not, Mom will continue anyway. This file may be placed elsewhere or given a different name, in which case pbs_mom must be started with the -c option.

The file provides several types of run time information to pbs_mom: static resource names and values, external resources provided by a program to be run on request via a shell escape, and values to pass to internal set up functions at initialization (and re-initialization).

Each item type is on a single line with the component parts separated by white space. If the line starts with a hash mark (pound sign, #), the line is considered to be a comment and is skipped.

3.6.1. Access Control and Initialization Values

An initialization value directive has a name which starts with a dollar sign ($) and must be known to Mom via an internal table. Currently the entries in this table are:

clienthost A *$clienthost* entry causes a host name to be added to the list of hosts which will be allowed to connect to Mom as long as it is using a privileged port. For example, here are two lines for the configuration file which will allow the hosts "fred" and "wilma" to connect:

```
$clienthost      fred
$clienthost      wilma
```

Two host names are always allowed to connect to pbs_mom, "localhost" and the name returned to pbs_mom by the system call gethostname(). These names need not be specified in the configuration file. The hosts listed as "clienthosts" comprise a "sisterhood" of hosts. Any one of the sisterhood will accept connections from a Scheduler [Resource Monitor (RM) requests] or Server [jobs to execute] from within the sisterhood. They will also accept Internal Mom (IM) messages from within the sisterhood. For a sisterhood to be able to communicate IM messages to each other, they must all share the same RM port.

For a Scheduler to be able to query resource information from a Mom, the Scheduler's host must be listed as a *clienthost*.

If the Server is provided with a nodes file, the IP addresses of the hosts (nodes) in the file will be forwarded by the Server to the Mom on each host listed in the node file. These hosts need not be in the various Mom's configuration file as they will be added internally when the list is received from the Server. The Server's host must be either the same host as the Mom or be listed as a clienthost entry in each Mom's config file.

restricted A *$restricted* host entry causes a host name to be added to the list of hosts which will be allowed to connect to Mom without needing to use a privilaged port. These names allow for wildcard matching. For example, here is a configuration file line which will allow queries from any host from the domain "ibm.com".

```
$restricted      *.ibm.com
```

Connections from the specified hosts are restricted in that only internal queries may be made. No resources from a config file will be reported and no control requests can be issued. This is to prevent any shell commands from being run by a non-root process.

This type of entry is typically used to specify hosts on which a monitoring tool, such as xpbsmon, can be run. Xpbsmon will query Mom for general resource information.

logevent A *$logevent* entry sets the mask that determines which event types are logged by pbs_mom. For example:

```
$logevent 0x1ff

$logevent 255
```

The first example would set the log event mask to 0x1ff (511) which enables logging of all events including debug events. The second example would set the mask to 0x0ff (255) which enables all events except debug events. The values of events are listed in section **6.3 Use and Maintenace of Logs** in this guide.

cputmult A *$cputmult* entry sets a factor used to adjust cpu time used by a job. This is provided to allow adjustment of time charged and limits enforced where the job might run on systems with different cpu performance. If Mom's system is faster than the reference system, set cputmult to a decimal value greater than 1.0. If Mom's system is slower, set cputmult to a value between 1.0 and 0.0. The value is given by

```
value = speed_of_this_system / speed_of_reference_system
```

For example:

```
$cputmult 1.5
```

or

```
$cputmult 0.75
```

wallmult A *$wallmult* entry sets a factor used to adjust wall time usage by to job to a common reference system. The factor is used for walltime calculations and limits in the same way as cputmult is used for cpu time.

usecp If Mom is to move a file to a host other than her own, Mom normally uses scp or rcp to transfer the file. This applies to stage-in/out and delivery of the job's standard output/error. [Please study the -o and -e option to qsub, qsub(1) man page, and section **3.3.5 Job Exit** of the ERS, to understand the naming convention for standard output and error files.] The destination is recorded as `hostx:/full/path/name`. So if `hostx` is not the same system on which Mom is running, then she uses scp or rcp; if it is the same system, then Mom uses /bin/cp.

If the destination file system is NFS mounted amoung all the systems in the PBS environment (cluster), then a cp may work better than s/rcp. One or more *$usecp* directives in the config file can be used to inform Mom on which file systems a cp command can be used instead of s/rcp. The $usecp entry has the form:

```
$usecp host_specification:path_prefix substitute_prefix
```

The *host_specification* is either a fully qualified host−domain name or a wild carded host−domain specification as used in the Server's host ACL attribute. The *path_prefix* is a leading component of the fully qualified path for the NFS files as visible on the specified host. The *substitute_prefix* is the initial components of the path to the same files on Mom's host. If different mount points are used, the path_prefix and the substitute_prefix will be different. If the same mount points are used for the cross mounted file system, then the two prefixes will be the same.

When given a file destination, Mom will:

1. Match the host_spec against her host name. If they match, Mom will use the cp command to move the file. If the hostspec is `localhost`, then Mom will also use cp.
2. If the match in step one fails, Mom will match the host portion of the destination against each $usecp host_specification in turn. If the host matches, Mom matches the *path_prefix* against the initial segment of the destination name. If this matches, Mom will discard the host name, replace the initial

segment of the path that matched against *path_prefix* with the *substitute_prefix* and use cp for the resulting destination.

3. If the host is neither the local host nor does it match any of the usecp directives, them Mom will use the rcp command to move the file.

For example, a user on host **myworkstation.company.com** submits a job while her current working directory is **/u/wk/her_home/proj**. The destination for her output would be given by PBS as `myworkstation.company.com:/u/wk/her_home/proj/123.OU` The job runs on host **pool2.company.com** which has the user's home file system cross mounted as **/r/home/her_home**, then either of the following entries in the config file on pool2

```
$usecp myworkstation.company.com:/u/wk/ /r/home/
$usecp *.company.com:/u/wk/  /r/home/
```

will result in a cp copy to `/r/home/her_home/proj/123.OU` instead of an rcp to `myworkstation.company.com:/u/wk/her_home/proj/123.OU`.

Note that the destination is matched against the $usecp entries in the order in the config file. The first match of host and file prefix determines the substitution. Therefore, if you have the same file system mounted on /foo on HostA and on /bar on every other host, then the entries for pool1 should be in the following order

```
$usecp HostA.company.com:/foo /bar
$usecp     *.company.com:/bar /bar
```

3.6.2. Static Resources

For static resource names and values, the configuration file contains a list of resource name/value pairs, one pair per line and separated by white space. An Example of static resource names and values could be the number of tape drives of different types and could be specified by

```
tape3480      4
tape3420      2
tapedat       1
tape8mm       1
```

The names can be anything and are not restricted to actual hardware. For example the entry `pong  1` could be used to indicate to the Scheduler that a certain piece of software is available on this system.

3.6.3. Shell Commands

If the first character of the value portion of a name/value pair is the exclamation mark (!), the entire rest of the line is saved to be executed through the services of the **system**(3) standard library routine. The first line of output from the shell command is returned as the response to the resource query.

The shell escape provides a means for the resource monitor to yield arbitrary information to the Scheduler. Parameter substitution is done such that the value of any qualifier sent with the resource query, as explained below, replaces a token with a percent sign (%) followed by the name of the qualifier. For example, here is a configuration file line which gives a resource name of "escape":

```
escape      !echo %xxx %yyy
```

If a query for "escape" is sent with no qualifiers, the command executed would be "echo %xxx %yyy". If one qualifier is sent, "escape[xxx=hi there]", the command executed would be "echo hi there %yyy". If two qualifiers are sent, "escape[xxx=hi][yyy=there]", the command executed would be "echo hi there". If a qualifier is sent with no matching token in the command line, "escape[zzz=snafu]", an error is reported.

Another example would allow the Scheduler to have Mom query the existence of a file. The following entry would be placed in Mom's config file:

```
file_exists !if test -f %file; then echo yes; else echo no; fi
```

The the query string "file_exists[file=/tmp/lockout]" would return "yes" if the file exists and "no" if it did not.

Another possible use of the shell command configuration entry is to provide a means by which the use of floating software licenses may be tracked. If a program can be written to query the license server, the number of available licenses could be returned to tell the Scheduler if it is possible to run a job that needs a certain licensed package. [You get the fun and games of writing this program.]

3.6.4. Examples of Config File

For the following examples, we will assume your site is "The Widget Company" and your domain name is "widget.com". The following is an example of a config file for pbs_mom where the batch system is a single large system. We want to log most records and specify that the system has 1 8mm tape drives.

```
$logevent 0x0ff
tape8mm 1
```

If the Scheduler for the large system happened to be on a front end machine, named fe.widget.com, then you would want to allow it to access Mom, so the config file becomes:

```
$logevent 0x0ff
$clienthost fe.widget.com
tape8mm 1
```

Now the center has expanded to two large systems. The new system has two tape drives and is 30% faster than the old system. You wish to charge the users the same regardless of where their job runs. Basing the charges on the old system, you will need to multiple the time used on the new system by 1.3 to charge the same as on the old system. The config file for the "old" system stays the same. The config file for the "new" system is:

```
$logevent 0x0ff
$clienthost fe.widget.com
$cputmult 1.3
$wallmult 1.3
tape8mm 2
```

Now you have put together a cluster of PCs running Linux named "bevy", as in a bevy of PCs. The Scheduler and Server is running on *bevyboss.widget.com* which also has the user's home file systems mounted as /u/home/... The nodes are named *bevy1.widget.com*, *bevy2.widget.com*, etc. The user's home file systems are NFS mounted as /r/home/... Your personal workstation, adm.widget.com, is where you plan to run *xpbsmon* to monitor the cluster. The config file for each Mom would look like:

```
$logevent 0x1ff
$clienthost bevyboss.widget.com
$restricted adm.widget.com
$usecp bevyboss.widget.com:/u/home /r/home
```

3.7. Configurating the Scheduler, pbs_sched

The configuration required for a Scheduler depends on the Scheduler itself. If you are starting with the delivered *fifo* Scheduler, please jump ahead to section 4.5.1 "FIFO Scheduler" in this guide.

4. Scheduling Policies

PBS provides a separate process to schedule which jobs should be placed into execution. This is a flexible mechanism by which you may implement a very wide variety of policies. The Scheduler uses the standard PBS API to communicate with the Server and an additional API to communicate with the PBS resource monitor, **pbs_mom**. Should the provided Schedulers be insufficient to meet your site's needs, it is possible to implement a replacement Scheduler using the provided APIs which will enforce the desired policies.

The first generation batch system, NQS, and many of the other batch systems use various queue based controls to limit or schedule jobs. Queues would be turned on and off to control job ordering over time or have a limit of the number of running jobs in the queue.

While PBS supports multiple queues and the queues have some of the "job scheduling" attributes used by other batch systems, the PBS Server does not by itself run jobs or enforce any of the restrictions implied by these queue attributes. In fact, the Server will happily run a *held* job that resides in a *stopped* queue with a zero limit on running jobs, if it is directed to do so. The direction may come from the operator, administrator, or the Scheduler. In fact, the Scheduler is nothing more than a client with administration privilege.

If you chose to implement your site scheduling policy using a multiple queue − queue control based scheme, you may do so. The Server and queue attributes used to control job scheduling may be adjusted by a client with privilege, such as **qmgr**(8B), or by one of your own creation. However, the controls actually reside in the Scheduler, not in the Server. The Scheduler must check the status of the Server and queues, as well as the jobs, determining the setting of the Server and queue controls. It then must use the settings of those controls in its decision making.

Another approach is the "whole pool" approach, wherein all jobs are in a single pool (single queue). The Scheduler evaluates each job on its merits and decides which, if any, to run. The policy can easily include factors such as time of day, system load, size of job, etc. Ordering of jobs in the queue need not be considered. The PBS team believes that this approach is superior for two reasons:

1. Users are not tempted to lie about their requirements in order to "game" the queue policy.
2. The scheduling can be performed against the complete set of current jobs resulting in better fits against the available resources.

4.1. Scheduler − Server Interaction

In developing a scheduling policy, it may be important to understand when and how the Server and the Scheduler interact. The Server always initiates the scheduling cycle. When scheduling is active within the Server, the Server opens a connection to the Scheduler and sends a command indicating the reason for the scheduling cycle. The reasons or events that trigger a cycle are:

- A job newly becomes eligible to execute. The job may be a new job in an execution queue, or a job in an execution queue that just changed state from held or waiting to queued. [SCH_SCHEDULE_NEW]
- An executing job terminates. [SCH_SCHEDULE_TERM]
- The time interval since the prior cycle specified by the Server attribute **schedule_iteration** is reached. [SCH_SCHEDULE_TIME]
- The Server attribute **scheduling** is set or reset to true. If set true, even if it's value was true, the Scheduler will be cycled. This provides the administrator/operator a means on forcing a scheduling cycle. [SCH_SCHEDULE_CMD]
- If the Scheduler was cycled and it requested one and only one job to be run, then the Scheduler will be recycled by the Server. This event is a bit abstruse. It

exists to "simplify" a Scheduler. The Scheduler only need worry about choosing the one best job per cycle. If other jobs can also be run, it will get another chance to pick the next job. Should a Scheduler run none or more than one job in a cycle it is clear that it need not be recalled until conditions change and one of the above trigger the next cycle. [SCH_SCHEDULE_RECYC]

- If the Server recently recovered, the first scheduling cycle, resulting from any of the above, will be indicated uniquely. [SCH_SCHEDULE_FIRST]

Once the Server has contacted the Scheduler and sent the reason for the contact, the Scheduler then becomes a privileged client of the Server. As such, it may command the Server to perform any action allowed to a manager.

When the Scheduler has completed all activities it wishes to perform in this cycle, it will close the connection to the Server. While a connection is open, the Server will not attempt to open a new connection.

Note, that the Server contacts the Scheduler to begin a scheduling cycl only if scheduling is active in the Server. This is controlled by the value of the Server attribute **scheduling**. If set true, scheduling is active and "qstat -B" will show the Server Status as Active. If scheduling is set false, then the Server will not contact the Scheduler and the Server's status is shown as Idle. When started, the Server will recover the value for **scheduling** as it was set when the Server shut down. The value may be changed in two ways: the -a option on the pbs_server command line, or by setting scheduling to true or false via qmgr.

One point should be clarified about job ordering:

Queues "are" and "are not" FIFOs.

What is meant is that while jobs are ordered first in − first out in the Server and in each queue, that fact does NOT imply that running them in that order is mandated, required, or even desirable. That is a decision left completely up to site policy and implementation. The Server will maintain the order across restarts solely as a aid to sites that wish to use a FIFO ordering in some fashion.

4.2. BaSL Scheduling

The provided BaSL Scheduler uses a C-like procedural language to write the scheduling policy. The language provides a number of constructs and predefined functions that facilitate dealing with scheduling issues. Information about a PBS Server, the queues that it owns, jobs residing on each queue, and the computational nodes where jobs can be run are accessed via the BaSL data types **Server, Que, Job, CNode, Set Server**, **Set Que**, **Set Job**, and **Set CNode**.

The idea is that a site must first write a function (containing the scheduling algorithm) called *sched_main()* (and all functions supporting it) using BaSL constructs, and then translate the functions into C using the BaSL compiler **basl2c**, which would also attach a main program to the resulting code. This main program performs general initialization and housekeeping chores such as setting up local socket to communicate with the Server running on the same machine, cd-ing to the priv directory, opening log files, opening configuration file (if any), setting up locks, forking the child to become a daemon, initializing a scheduling cycle (i.e. get node attributes that are static in nature), setting up the signal handlers, executing global initialization assignment statements specified by the Scheduler writer, and finally sitting on a loop waiting for a scheduling command from the Server. The name of the resulting code is *pbs_sched.c*.

When the Server sends the Scheduler an appropriate scheduling command { SCH_SCHEDULE_NEW , SCH_SCHEDULE_TERM , SCH_SCHEDULE_TIME , SCH_SCHEDULE_RECYC , SCH_SCHEDULE_CMD , SCH_SCHEDULE_FIRST }, the Scheduler wakes up and obtains information about Server(s), jobs, queues, and execution host(s), and

then it calls *sched_main()*. The list of Servers, execution hosts, and host queries to send to the hosts' Moms are specified in the Scheduler configuration file.

Global variables defined in the BaSL program will retain their values in between scheduling cycles while locally-defined variables do not.

4.3. Tcl Based Scheduling

The provided Tcl based Scheduler framework uses the basic Tcl interpreter with some extra commands for communicating with the PBS Server and Resource Monitor. The scheduling policy is defined by a script written in Tcl. A number of sample scripts are provided in the source directory *src/scheduler.tcl/sample_scripts*.

The Tcl based Scheduler works, very generally, in the following way:

1. On start up, the Scheduler reads the initialization script (if specified with the -i option) and executes it. Then, the body script is read into memory. This is the file that will be executed each time a "schedule" command is received from the Server. It then waits for a "schedule" command from the Server.
2. When a schedule command is received, the body script is executed. No special processing is done for the script except to provide a connection to the Server. A typical script will need to retrieve information for candidate jobs to run from the Server using **pbsselstat** or **pbsstatjob**. Other information from the Resource Monitor(s) will need to be retrieved by opening connections with **openrm** and submitting queries with **addreq** and getting the results with **getreq**. The Resource Monitor connections must be closed explicitly with **closerm** or the Scheduler will eventually run out of file descriptors. When a decision is made to run a job, a call to **pbsrunjob** must be made.
3. When the script evaluation is complete, the Scheduler will close the TCP/IP connection to the Server.

4.3.1. Tcl Based Scheduling Advice

The Scheduler does not restart the Tcl interpreter for each cycle. This gives the ability to carry information from one cycle to the next. It also can cause problems if variables are not initialized or "unset" at the beginning of the script when they are not expected to contain any information later on.

System load average is frequently used by a script. This number is obtained from the system kernel by pbs_mom. Most systems smooth the load average number over a time period. If one scheduling cycle runs one or more jobs and the next scheduling cycle occurs quickly, the impact of the newly run jobs will likely not be reflected in the load average. This can cause the load average to shoot way up especially when first starting the batch system. Also when jobs terminate, the delay in lowering the load average may delay the scheduling of additional jobs.

The Scheduler redirects the output from "stdout" and "stderr" to a file. This makes it easy to generate debug output to check what your script is doing. It is advisable to use this feature heavily until you are fairly sure that your script is working well.

4.3.2. Implementing a Tcl Scheduler

The best advice is study the examples found in src/scheduler.tcl/sample_scripts. Then once you have modified or written a scheduler body script and optionally an initialization script, place them in the directory {PBS_HOME}/sched_priv and invoke the Scheduler typing

```
{sbindir}/pbs_sched [-b body_script] [-i init_script]"
```

See the pbs_sched_tcl(8) man page for more information.

4.4. C Based Scheduling

The C based Scheduler is similar in structure and operation to the Tcl Scheduler except that C functions are used rather than Tcl scripts.

1. On start up, the Scheduler calls *schedinit(argc, argv)* one time only to initialize whatever is required to be initialized.
2. When a schedule command is received, the function *schedule(cmd, connector)* is invoked. All scheduling activities occur within that function.
3. Upon return to the main loop, the connection to the Server is closed.

Several working Scheduler code examples are provided in the samples subdirectory. The following sections discuss certain of the sample schedulers including the default scheduler fifo. The sources for the samples are found in *src/scheduler.cc/samples* under the Scheduler type name, for example *src/scheduler.cc/samples/fifo*.

4.4.1. FIFO Scheduler

This Scheduler will provide several simple scheduling policies. It provides the ability to sort the jobs in several different ways, in addition to FIFO order. There is also the ability to sort on user and group priority. Mainly this Scheduler is intended to be a jumping off point for a real Scheduler to be written. A good amount of code has been written to make it easier to change and add to this Scheduler. Check the IDS for a more detailed view of the code.

As distributed, the fifo Scheduler is configured with the following options, see file `PBS_HOME/sched_priv/sched_config`:

- All jobs in a queue will be considered for execution before the next queue is examined.
- The queues are sorted by queue priority.
- The jobs within each queue are sorted by requested cpu time (cput). The shortest job is places first.
- Jobs which have been queued for more than a day will be considered starving and heroic measures will be taken to attempt to run them.
- Any queue whose name starts with "ded" is treated as a dedicated time queue. Jobs in that queue will only be considered for execution if the system is in dedicated time as specified in the `dedicated_time` configuration file. If the system is in dedicated time, jobs not in a "ded" queue will not considered. (See file `PBS_HOME/sched_priv/dedicated_time`)
- Prime time is from 4:00 AM to 5:30 PM. Any holiday is considered non-prime. Standard federal holidays for the year 1998 are included. (See file `PBS_HOME/sched_priv/holidays`)
- A sample `dedicated_time` and resource group file are also included.
- These system resources are checked to make sure they are not exceeded: *mem* (memory requested) and *ncpus* (number of CPUs requested).

4.4.1.1. Installing the FIFO Scheduler

1. As discussed in the build overview, run configure with the following options: `--set-sched=c` and `--set-sched-code=fifo`, which are the default.
2. You may wish to read through the src/scheduler.cc/samples/fifo/config.h file. Most default values will be fine.
3. Build and install PBS
4. Change directory into `PBS_HOME/sched_priv` and edit the scheduling policy config file `sched_config`, or use the default values. This file controls the

scheduling policy (which jobs are run when). The default name of `sched_config` may be changed in config.h. The format of the sched_config file is:

name: value [prime | non_prime | all]

name and value may not contain any white space
value can be: true | false | number | string
any line starting with a '#' is a comment.
a blank third word is equivalent to "all" which is both prime and non-prime

the associated values as shipped as defaults are shown in braces {}:

round_robin
: boolean: If true − run jobs one from each queue in a circular fashion; if false − run as many jobs as possible up to queue/server limits from one queue before processing the next queue. The following server and queue attributes, if set, will control if a job "can be" run: **resources_max**, **max_running**, **max_user_run**, and **max_group_run**. See the man pages pbs_server_attributes and pbs_queue_attributes.
{false all}

by_queue
: boolean: If true − the jobs will be run from their queues; if false − the entire job pool in the Server is looked at as one large queue.
{true all}

strict_fifo
: boolean: If true − will run jobs in a strict FIFO order. This means if a job fails to run for any reason, no more jobs will run from that queue/server that scheduling cycle. If *strict_fifo* is not set, large jobs can be starved, i.e., not allowed to run because a never ending series of small jobs use the available resources. Also see the server attribute **resources_max** in section 3.5.1, and the fifo parameter *help_starving_jobs* below.
{false all}

fair_share
: boolean: This will turn on the fair share algorithm. It will also turn on usage collecting and jobs will be selected using a function of their usage and priority(shares).
{false all}

load_balancing
: boolean: If this is set the Scheduler will load balance the jobs between a list of time-shared hosts (:ts) obtained from the Server (pbs_server). The Server reads the list from its nodes file, see section 3.2.
{false all}

help_starving_jobs
: boolean: This bit will have the Scheduler turn on its rudimentry starving jobs support. Once jobs have waited for the amount of time give by `starve_max`, they are considered starving. If a job is considered starving, then no jobs will run until the starving job can be run. `Starve_max` needs to be set also.

sort_by
string: have the jobs sorted. sort_by can be set to a single sort type or *multi_sort*. If set to *multi_sort*, multiple *key* fields are used. Each *key* field will be a key for the multi sort. The order of the key fields decides which sort type is used first.

Sorts: no_sort, shortest_job_first, longest_job_first, smallest_memory_first, largest_memory_first, high_priority_first, low_priority_first, multi_sort, fair_share, large_walltime_first, short_walltime_first
{shortest_job_first}

no_sort
do not sort the jobs

shortest_job_first
ascending by the cput attribute

longest_job_first
descending by the cput attribute

smallest_memory_first
ascending by the mem attribute

largest_memory_first
descending by the mem attribute

high_priority_first
descending by the job priority attribute

low_priority_first
ascending by the job priority attribute

large_walltime_first
descending by job walltime attribute

cmp_job_walltime_asc
ascending by job walltime attribute

multi_sort
sort on multiple keys.

fair_share
If `fair_share` if given as the sort key, the jobs are sorted based on the values in the resource group file. This is only used if strict priority sorting is needed.

key Sort type as defined above for multiple sorts. Each sorting key is listed on a separate line starting with the word *key*. For example:

```
sort_by: multi_sort
key: sortest_job_first
key: smallest_memory_first
key: high_priority_first
```

log_filter
What event types not to log. The value should be the addition of the event classes which should be filtered (i.e. ORing them together). The numbers are defined in *src/include/log.h*. NOTE: those numbers are in hex and log_filter is in base 10.
{256}

Examples:
To filter PBSEVENT_DEBUG2, PBSEVENT_DEBUG and PBSEVENT_ADMIN
0x100: 256 0x080: 128 0x004: 4= 388
log_filter 388

To filter PBSEVENT_JOB,PBSEVENT_DEBUG and PBSEVENT_SCHED
0x008: 8 0x080: 128 0x040: 64= 200
log_filter 200

dedicated_prefix
The queues with this prefix will be considered dedicated queues. Example: if the dedicated prefix is "ded" then dedicated, ded1, ded5 etc would be dedicated queues

{ded}

starve_max
The amount of time before a job is considered starving. This config variable is not used if `help_starving_jobs` is not set.

The following do not matter if fair share is not turned on (which is is not by default).

half_life
The half life of the fair share usage
{24:00:00}

unknown_shares
The amount of shares for the "unknown" group.
{10}

sync_time
The amount of time between writing the fair share usage data to disk.
{1:00:00}

The policy set by the supplied values in sched_config is:
Jobs are run on the basis of queue priority, both in prime and non-prime time.
Jobs with in each queue are sorted on the basis of smallest (memory) first.
Help for starving jobs will take effect after a job is 24 hours old.

5. If fair share or strict priority is going to be used, the resource group file `{PBS_HOME}/sched_priv/resources_group`, will need to be edited. A sample file was installed. When editing the file, use the following format for each line of the file:

```
# comment
username cresgrp resgrp shares
```

username
string: the username of the user or the group

cresgrp
numeric: an id for the group or user, should be unique for each. For users, the UID works well.

resgrp
string: the name of the parent resource group this user/group is in. The root of the entire tree is called `root` and is added automatically to the tree by the Scheduler.

shares
numeric: The amount of shares(priority) the user/group has in the resource group.

6. If strict priority is wanted, a fair share tree will be needed. A really simple one will suffice. Every user's resgrp will be root. The amount of shares will be their priority. Next, set unknown_shares to one. Everyone who is not in the tree will share the one share between them to make sure everyone in the tree will have priority over them. Lastly, the main sort must be set to *fair_share*. This will sort by the fair share tree which was just set up.

7. Create the holidays file to handle prime time and holidays. The holidays file should use the UNICOS 8 holiday format. The ordering does matter. Any line that begins with a "*" is considered a comment.

   YEAR YYYY
   This is the current year.

   <day> <prime> <nonprime>
   Day can be weekday | saturday | sunday
   prime and nonprime are times when prime or non-prime time start. They can either be HHMM with no colons(:) or the word "all" or "none"

   <day> <date> <holiday>
   day is the day of the year between 1 and 365 date is the calendar date. Ex Jan 1 holiday is the name of the holiday. Ex New Year's Day This is repeated for each company holiday

8. To load balance between timesharing nodes, several things need to happen. First, a nodes file needs to be set up as PBSHOME/server_priv/nodes. (See section 3.2). All timesharing nodes need to be denoted with :ts appended to the hostname. These are the nodes between which the Scheduler will load balance. Secondly, on every node there has to be a Mom. In each of Mom's config files two static values need to be set up. One is for the ideal load and the other for the maximum load. This is done by putting two lines in the config file in the following format: name value. The names will be *ideal_load* and *max_load*. Lastly, turn the load_balancing bit on in the scheduling policy config file. Load balancing will have the job comment changed on running of the job to show where the job was run.

   Example of Mom config file:(64 processor machine)
   ideal_load 60
   max_load 64

9. Space sharing is done automatically if there are both a nodes file and the job requests nodes. Make sure to set up a resources_default.nodes and resources_default.nodect.

10. The Scheduler honors the following attributes/node resources:

Source Object	Attribute/Resource	Comparison
Queue	started	equal true
Queue	queue_type	equal execution
Queue	max_running	ge #jobs running
Queue	max_user_run	ge #jobs running for a user
Queue	max_group_run	ge #jobs running for a group
Job	job state	equal Queued
Server	max_running	ge #jobs running
Server	max_user_run	ge #jobs running for a user
Server	max_group_run	ge #jobs running for a group
Server	resources_available	ge resources requested by job
Server	resources_max	ge resources requested
Node	loadave	less than configured limit
Node	arch	equal type requested
Node	host	equal name requested
Node	ncpus	ge number ncpus requested
Node	physmem	ge amount mem requested

NOTE: if resources_available.res is set, it will be used, if not resources_max.res will be used. If neither are set infinity is assumed.

4.4.1.2. Examples FIFO Configuration Files

The following are just examples and may or may not be what is shipped.

Example of a scheduling config file

```
#     Set the boolean values which define how the scheduling policy finds
#     the next job to consider to run.
round_robin: False   ALL
by_queue: True       prime
by_queue: false      non-prime
strict_fifo: true    ALL
fair_share: True prime
fair_share: false    non-prime

# help jobs which have been waiting too long
help_starving_jobs: true prime
help_starving_jobs: false    non-prime

# Set a multi_sort
# This example will sort jobs first by ascending cpu time requested, and then
# by ascending memory requested, and then finally by descending job priority
#
sort_by: multi_sort
key: shortest_job_first
key: smallest_memory_first
key: high_priority_first

# Set the debug level to only show high level messages.
# Currently this only shows jobs being run
debug_level: high_mess
```

```
# a job is considered starving if it has waited for this long
max_starve:  24:00:00

# If the Scheduler comes by a user which is not currently in the resource group
# tree, they get added to the "unknown" group.  The "unknown" group is in roots
# resource group.  This says how many shares it gets.
unknown_shares: 10

# The usage information needs to be written to disk in case the Scheduler
# goes down for any reason.  This is the amount of time between when the
# usage information in memory is written to disk.  The example syncs the
# information ever hour.
sync_time: 1:00:00

# What events do you not want to log.  The event numbers are defined in
# src/include/log.h.  NOTE: the numbers are in hex, and log_filter is in
# base 10.
# The example is not to log DEBUG2 events, which are the most prolific
log_filter: 256
```

Here is an example of the holidays file

```
* the current year
YEAR    1998

*
* Start and end of prime time
*
*         Prime   Non-Prime
* Day      Start   Start
weekday       0400   1130
saturday      none   all
sunday        none   all


*
* The holidays
*
* Day of     Calendar      Company
* Year       Date          Holiday
*
1             Jan 1         New Year's Day
20            Jan 20        Martin Luther King Day
48            Feb 17        President's Day
146           May 26        Memorial Day
185           Jul 4         Independence Day
244           Sep 1         Labor Day
286           Oct 13        Columbus Day
315           Nov 11        Veteran's Day
331           Nov 27        Thanksgiving
359           Dec 25        Christmas Day
```

Example of the resource group file for fair share

```
#
# the groups "root" and "unknown" are added by the Scheduler
# All the parents must be added for the children.  This is why all the groups
# are added first.  The cresgrp numbers the users have are their UIDs
#

# name          resgrp          child resgrp      shares


grp1            50          root              10
grp2            51          root              20
grp3            52          root              10
grp4            53          grp1              20
grp5            54          grp1              10
grp6            55          grp2              20
usr1            60          root              5
usr2            61          grp1              10
usr3            62          grp2              10
usr4            63          grp6              10
usr5            64          grp6              10
usr6            65          grp6              20
usr7            66          grp3              10
usr8            67          grp4              10
usr9            68          grp4              10
usr10           69          grp5              10
```

Example of strict priority resource group file

```
# this is a strict priority resource group file.  These are people who should
# get priority over everyone else.  The amount of shares is the priority of
# the user.


sally           1000        root              4
larry           1001        root              6
manager         1010        root              100
vp              1016        root              500
ceo             2000        root              10000
```

Example of dedicated file

```
# Format:
#     FROM                TO
# MM/DD/YYYY HH:MM    MM/DD/YYYY HH:MM

04/10/1998     15:30     04/11/1998     23:50
05/15/1998     05:15     05/15/1998     08:30
06/10/1998     23:25     06/10/1998     23:50
```

4.4.2. IBM_SP Scheduler

This is a highly optimized scheduler for the IBM SP series of supercomputers. This scheduler was the first to provide a "dynamic backfill" algorithm for the SP. The algorithm is designed to implement a usage policy comparable to the one found on NAS traditional vector supercomputers. The algorithm primary goals are to minimize the turnaround time for small jobs during Prime-Time hours, and to maintain the highest possible node utilization during NonPrime-Time hours. Scheduling a diverse workload composed of interactive, small debugging, and long batch jobs presents significant difficulties on the SP, due to its limited resource management capabilities, and parallel job scheduling restrictions (only space-sharing, no time-sharing). The space-sharing scheduling algorithm utilized uses a sophisticated Dynamic-Backfilling method to overcome the SP limitations. The algorithm achieves turnaround time for small jobs to 10 - 20 minutes, and maintains node utilization around 75%. See the whitepaper included in the scheduler.cc/samples/ibm_sp directory for a full discussion of the algorithms used.

4.4.2.1. Installing the IBM SP Scheduler

1. As discussed in the build overview, run configure with the following options:
   `--set-sched=cc and --set-sched-code=ibm_sp`
2. Review src/scheduler.cc/samples/ibm_sp/sched_globals.h editing any variables necessary, such as the value of SCHED_DEFAULT_CONFIGURATION.
3. Build and install PBS.
4. Change directory into {PBS_HOME}/sched_priv and edit the scheduler configuration file "config" (see 4.5.2.2). This file controls the scheduling policy used to determine which jobs are run and when. The comments in the config file explain what each option is for. If in doubt, the default option is generally acceptable.

4.4.2.2. Configuring the IBM_SP Scheduler

The ibm_sp scheduler config file contains the following tunable parameters, which control the policy implemented by the scheduler. Comments are allowed anywhere in the file, and begin with a '#' character. Any non-comment lines are considered to be statements, and must conform to the syntax:

```
<option> <argument>
```

Arguments must be one of:

<boolean> A boolean value. Either 0 (false/off) or 1 (true/on)

<domain> A registered domain name, eg. "mrj.com"

<hostname> A hostname registered in the DNS system.

<integer> An integral (typically non-negative) decimal value.

<pathname> A valid pathname (i.e. "/usr/local/pbs/pbs_acctdir").

<real> A real valued number (i.e. the number 0.80).

<string> An uninterpreted string passed to other programs.

<time_spec> A string of the form HH:MM:SS (i.e. 00:30:00).

Below is a listing of the available configuration parameters for this scheduler, and a brief explaination of each. See the README and the actual "config" files for a detailed description.

Parameter	Type	Definition

DEFAULT_ATTR	<string>	Define default node attribute
ENFORCE_ALLOC	<boolean>	Indicate enforcement of allocations
ENFORCE_DEDTIME	<boolean>	Indicate enforcement of dedicated time
LOCAL_DOMAIN	<domain>	Local network domain name
LOWUSAGE_NODEINUSE	<integer>	Threshold where we start to ignore "policy"
MAXJOB_RUNNING	<integer>	Maximum number of jobs allowed per user
MAXJOB_WALLTIME	<integer>	Maximum walltime (seconds) that a job is allowed to run in the 'normal' queue. If the request is over, the job is deleted.
MAX_QUEUED_TIME	<integer>	Seconds to wait before delaying other jobs
MIN_QUEUED_TIME	<integer>	Seconds a short job should remain in the queue.
NODEUSAGE_DECAY	<real>	Decay factor of node/hour usage
NONPRIME_AVAIL	<integer>	Define Non-Prime node high availability
NONPRIME_BATCH_START	<time_spec>	Define start of the NonPrime-Time Batch only period
NONPRIME_BATCH_STOP	<time_spec>	Define end of the NonPrime-Time Batch only period
NONPRIME_SAT_START	<time_spec>	Special case for the interactive period on Saturday
NONPRIME_SAT_STOP	<time_spec>	Special case for the interactive period on Saturday
OVERALLOC_DECAY	<real>	Decay factor for jobs over allocation.
PBS_HOST	<string>	Name of system -- ie, for the whole SP
PBS_HOST_UPPER	<string>	Upper case version of PBS_HOST
PBS_SERVER	<hostname>	Hostname where PBS server is running
PEER_ENABLE	<boolean>	Enable MetaCenter PEER checking -- for PeerScheduler
PERCENT_TO_LETGO	<integer>	Threashold for % of time shift required for a job to be scheduled.
PRIME_32_END	<time_spec>	End of <32 node window
PRIME_32_START	<time_spec>	Jobs <32 nodes can start during prime
PRIME_AVAIL	<integer>	Define Prime node high availability
PRIME_NODE	<integer>	Define Prime Time Node size Threshold
PRIME_TIME_END	<time_spec>	Define end of the Prime-Time period
PRIME_TIME_START	<time_spec>	Define start of the Prime-Time period
QUEUE_DEDTIME	<pathname>	Name of "dedicated time" queue
QUEUE_PBS	<pathname>	Name of primary/default queue)
QUEUE_SPECIAL	<pathname>	Name of "special" queue
RESMON_HOST	<hostname>	Hostname where PBS mom/resmom is running
SCHEDULE_DOWNTIME	:<pathname>	Location of 'schedule' command for scheduled downtime
SCHED_ACCT_DIR	<pathname>	Location of the per-group allocation and usage files
SCHED_DEBUGGING	<pathname>	Location of the scheduler debugging config file
SCHED_DECAY	<pathname>	Location of the scheduler usage decay file
SCHED_MAPFILE	<pathname>	Location of the user mapfile
SCHED_OUTPUT	<pathname>	Location of the scheduler output file
SCHED_STATUS	<pathname>	Location of the scheduler status file
SCHED_TIMEOUT	<integer>	Seconds to wait before timing out a connection
SEEK_WORK_DELAY	<integer>	Seconds to wait before contacting a PEER
SHIFT_NODELIMIT	<integer>	Node watermark limit for the dynamic backfilling
SMALL_QUEUED_TIME	<time_spec>	Treshold to separate a long job from a short job.
TYPE_AVAIL	<integer>	Flag to maintain availability for a specific node request
TYPE_NODEAVAIL	:<string>	Node request to maintain highly available
USE_SITE_MAPFILE	<boolean>	Indicate use of Username Mapfile
WALLTIME0	<time_spec>	Maximum walltime constants for over-allocation jobs
WALLTIME1	<time_spec>	Walltime limit constants for normal jobs
WALLTIME2	<time_spec>	Walltime limit constants for normal jobs
WALLTIME5	<time_spec>	Maximum walltime constants for over-allocation jobs

4.4.3. SGI Origin Scheduler

This is a highly specialized scheduler for managing a cluster of SGI Origin2000 systems, providing integrated support for Array Services (for MPI programs), and NODEMASK (to pin applications via software to dynamically created regions of nodes within the system). The scheduling algorithm includes an implementation of static backfill and dynamically calculates NODEMASKs on a per-job basis. (See the README file in the scheduler.cc/samples/sgi_origin directory for details of the algorithm.)

4.4.3.1. Installing the SGI_ORIGIN Scheduler

1. As discussed in the build overview, run configure with the following options:
   ```
   --set-sched=cc  --set-sched-code=sgi_origin
   ```
   If you wish to enable scheduler use of the NODEMASK facility, then also add the configure option `--enable-nodemask`.
2. Review src/scheduler.cc/samples/sgi_origin/toolkit.h editing any variables necessary, such as the value of SCHED_DEFAULT_CONFIGURATION.
3. Build and install PBS.
4. Change directory into {PBS_HOME}/sched_priv and edit the scheduler configuration file "config" (see 4.5.3.2). This file controls the scheduling policy used to determine which jobs are run and when. The comments in the config file explain what each option is. If in doubt, the default option is generally acceptable.

4.4.3.2. Configuring the SGI Origin Scheduler

The {PBS_HOME}/sched_priv/config file contains the following tunable parameters, which control the policy implemented by the scheduler. Comments are allowed anywhere in the file, and begin with a '#' character. Any non-comment lines are considered to be statements, and must conform to the syntax:

```
<option> <argument>
```

See the README and config files for a description of the options listed below, and the type of argument expected for each of the options. Arguments must be one of:

<boolean>
: A boolean value. The strings "true", "yes", "on" and "1" are all true, anything else evaluates to false.

<hostname>
: A hostname registered in the DNS system.

<integer>
: An integral (typically non-negative) decimal value.

<pathname>
: A valid pathname (i.e. "/usr/local/pbs/pbs_acctdir").

<queue_spec>
: The name of a PBS queue. Either 'queue@exechost' or just 'queue'. If the hostname is not specified, it defaults to the name of the local host machine.

<real>
: A real valued number (i.e. the number 0.80).

<string>
: An uninterpreted string passed to other programs.

<time_spec>
: A string of the form HH:MM:SS (i.e. 00:30:00 for thirty minutes, 4:00:00 for four hours).

<variance>
: Negative and positive deviation from a value. The syntax is '-mm%,+nn%' (i.e.

'-10%,+15%' for minus 10 percent and plus 15% from some value).

Syntactical errors in the configuration file are caught by the parser, and the offending line number and/or configuration option/argument is noted in the scheduler logs. The scheduler will not start while there are syntax errors in its configuration files.

Before starting up, the scheduler attempts to find common errors in the configuration files. If it discovers a problem, it will note it in the logs (possibly suggesting a fix) and exit.

The following is a complete list of the recognized options:

Parameter	Type
AVOID_FRAGMENTATION	<boolean>
BATCH_QUEUES	<queue_spec>[,<queue_spec>...]
DECAY_FACTOR	<real>
DEDICATED_QUEUE	<queue_spec>
DEDICATED_TIME_CACHE_SECS	<integer>
DEDICATED_TIME_COMMAND	<pathname>
ENFORCE_ALLOCATION	<boolean>
ENFORCE_DEDICATED_TIME	<boolean>
ENFORCE_PRIME_TIME	<boolean>
EXTERNAL_QUEUES	<queue_spec>[,<queue_spec>...]
FAKE_MACHINE_MULT	<integer>
HIGH_SYSTIME	<integer>
INTERACTIVE_LONG_WAIT	<time_spec>
MAX_DEDICATED_JOBS	<integer>
MAX_JOBS	<integer>
MAX_QUEUED_TIME	<time_spec>
MAX_USER_RUN_JOBS	<integer>
MIN_JOBS	<integer>
NONPRIME_DRAIN_SYS	<boolean>
OA_DECAY_FACTOR	<real>
PRIME_TIME_END	<time_spec>
PRIME_TIME_SMALL_NODE_LIMIT	<integer>
PRIME_TIME_SMALL_WALLT_LIMIT	<time_spec>
PRIME_TIME_START	<time_spec>
PRIME_TIME_WALLT_LIMIT	<time_spec>
SCHED_ACCT_DIR	<pathname>
SCHED_HOST	<hostname>
SCHED_RESTART_ACTION	<string>
SERVER_HOST	<hostname>
SMALL_JOB_MAX	<integer>
SMALL_QUEUED_TIME	<time_spec>
SORT_BY_PAST_USAGE	<boolean>
SPECIAL_QUEUE	<queue_spec>
SUBMIT_QUEUE	<queue_spec>
SYSTEM_NAME	<hostname>
TARGET_LOAD_PCT	<integer>
TARGET_LOAD_VARIANCE	<variance>
TEST_ONLY	<boolean>
WALLT_LIMIT_LARGE_JOB	<time_spec>
WALLT_LIMIT_SMALL_JOB	<time_spec>

See the following files for detailed explaination of these options:

src/scheduler.cc/samples/sgi_origin/README
src/scheduler.cc/samples/sgi_origin/config

4.4.4. Multitask Scheduler

This scheduler provides support for "multi-tasking" (ie timesharing of CPU and memory resources). Orginally written for the SGI PowerChallenge, and later ported to the Origin 2000, this scheduler should work for most shared-memory multiprocessor (SMP) systems.

4.4.4.1. Installing the Multitask Scheduler

1. As discussed in the build overview, run configure with the following options:
   `--set-sched=cc --set-sched-code=multitask`
2. Review src/scheduler.cc/samples/multitask/toolkit.h editing any variables necessary, such as the value of SCHED_DEFAULT_CONFIGURATION.
3. Build and install PBS.
4. Change directory into PBS_HOME/sched_priv and edit the scheduler configuration file "config". This file controls the scheduling policy used to determine which jobs are run and when. The comments in the config file explain what each option is for. If in doubt, the default option is generally acceptable.

4.5. Scheduling and File Staging

A decision must be made about when to begin to stage-in files for a job. The files must be available before the job executes. The amount of time that will be required to copy the files is unknown to PBS, that being a function of file size and network speed. If file in-staging is not started until the job has been selected to run when the other required resources are available, either those resources are "wasted" while the stage-in occurs, or another job is started which takes the resources away from the first job, and might prevent it from running. If the files are staged in well before the job is otherwise ready to run, the files may take up valuable disk space need by running jobs.

PBS provides two ways that file in-staging can be initiated for a job. If a run request is received for a job with a requirement for staging-in files, the staging in operation is begun and when completed, the job is run. Or, a specific stage-in request may be received for a job, see pbs_stagein(3B), in which case the files are staged in but the job is not run. When the job is run, it begins execution immediately because the files are already there.

In either case, if the files could not be staged-in for any reason, the job is placed into a wait state with a "execute at" time **PBS_STAGEFAIL_WAIT**, 30 minutes in the future. A mail message is sent to the job owner requesting that s/he look into the problem. The reason the job is changed into wait state is to prevent the Scheduler from constantly retrying the same job which likely would keep on failing.

Figure 5.0 in appendix B of the ERS shows the (sub)state changes for a job involving file in staging. The Scheduler may note the substate of the job and chose to perform pre-staging via the pbs_stagein() call. The substate will also indicate completeness or failure of the operation. The Scheduler developer should carefully chose a stage-in approach based on factors such as the likely source of the files, network speed, and disk capacity.

5. GUI System Administrator Notes

Currently, PBS provides two GUIs: xpbs and xpbsmon.

5.1. xpbs

xpbs provides a user-friendly point-and-click interface to the PBS commands. The xpbs(1) man page provides full information on configuring and running xpbs. Some of that information is repeated here. To run **xpbs** as a regular, non-privileged user, type:

```
setenv DISPLAY <display_host>:0"
xpbs
```

To run **xpbs** with the additional purpose of terminating PBS Servers, stopping and starting queues, or running/rerunning jobs, then run:

```
xpbs -admin
```

Running **xpbs** will initialize the X resource database from various sources in the following order:

1. The **RESOURCE_MANAGER** property on the root window (updated via xrdb) with settings usually defined in the .Xdefaults file
2. Preference settings defined by the system administrator in the global xpbsrc file
3. User's ~/.xpbsrc file - this file defines various X resources like fonts, colors, list of PBS hosts to query, criteria for listing queues and jobs, and various view states. See XPBS Preferences section below for a list of resources that can be set.

The system administrator can specify a global resources file, *{libdir}/xpbs/xpbsrc*, which is read by the GUI if a personal *.xpbsrc* file is missing. Keep in mind that within an Xresources file (Tk only), later entries take precedence. For example, suppose in your *.xpbsrc* file, the following entries appear in order:

```
xpbsrc*backgroundColor: blue
*backgroundColor: green
```

The later entry "green" will take precedence even though the first one is more precise and longer matching.

The things that can be set in the personal preferences file are fonts, colors, and favorite Server host(s) to query.

5.1.1. XPBS Preferences

The resources that can be set in the X resources file, ~/.xpbsrc, are:

*serverHosts
: list of server hosts (space separated) to query by **xpbs**.

*timeoutSecs
: specify the number of seconds before timing out waiting for a connection to a PBS host.

*xtermCmd
: the xterm command to run driving an interactive PBS session.

*labelFont
: font applied to text appearing in labels.

*fixlabelFont
: font applied to text that label fixed-width widgets such as listbox labels. This must be a fixed-width font.

*textFont
: font applied to a text widget. Keep this as fixed-width font.

*backgroundColor
the color applied to background of frames, buttons, entries, scrollbar handles.

*foregroundColor
the color applied to text in any context (under selection, insertion, etc...).

*activeColor
the color applied to the background of a selection, a selected command button, or a selected scroll bar handle.

*disabledColor
color applied to a disabled widget.

*signalColor
color applied to buttons that signal something to the user about a change of state. For example, the color of the button when returned output files are detected.

*shadingColor
a color shading applied to some of the frames to emphasize focus as well as decoration.

*selectorColor
the color applied to the selector box of a radiobutton or checkbutton.

*selectHosts
list of hosts (space separated) to automatically select/highlight in the HOSTS listbox.

*selectQueues
list of queues (space separated) to automatically select/highlight in the QUEUES listbox.

*selectJobs
list of jobs (space separated) to automatically select/highlight in the JOBS listbox.

*selectOwners
list of owners checked when limiting the jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Owners: <list_of_owners>". See -u option in **qselect(1B)** for format of <list_of_owners>.

*selectStates
list of job states to look for (do not space separate) when limiting the jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Job_States: <states_string>". See -s option in **qselect(1B)** for format of <states_string>.

*selectRes
list of resource amounts (space separated) to consult when limiting the jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Resources: <res_string>". See -l option in **qselect(1B)** for format of <res_string>.

*selectExecTime
the Execution Time attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Queue_Time: <exec_time>". See -a option in **qselect(1B)** for format of <exec_time>.

*selectAcctName
the name of the account that will be checked when limiting the jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Account_Name: <account_name>". See -A option in **qselect(1B)** for format of <account_name>.

*selectCheckpoint
the checkpoint attribute relationship (including the logical operator) to consult

when limiting the list of jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Checkpoint: <checkpoint_arg>". See -c option in **qselect(1B)** for format of <checkpoint_arg>.

*selectHold
: the hold types string to look for in a job when limiting the jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Hold_Types: <hold_string>". See -h option in **qselect(1B)** for format of <hold_string>.

*selectPriority
: the priority relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Priority: <priority_value>". See -p option in **qselect(1B)** for format of <priority_value>.

*selectRerun
: the rerunnable attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Rerunnable: <rerun_val>". See -r option in **qselect(1B)** for format of <rerun_val>.

*selectJobName
: name of the job that will be checked when limiting the jobs appearing on the Jobs listbox in the main **xpbs** window. Specify value as "Job_Name: <jobname>". See -N option in **qselect(1B)** for format of <jobname>.

*iconizeHostsView
: a boolean value (true or false) indicating whether or not to iconize the HOSTS region.

*iconizeQueuesView
: a boolean value (true or false) indicating whether or not to iconize the QUEUES region.

*iconizeJobsView
: a boolean value (true or false) indicating whether or not to iconize the JOBS region.

*iconizeInfoView
: a boolean value (true or false) indicating whether or not to iconize the INFO region.

*jobResourceList
: a curly-braced list of resource names as according to architecture known to xpbs. The format is as follows:
{ <arch-type1> resname1 resname2 ... resnameN }
{ <arch-type2> resname1 resname2 ... resnameN }
. . .
{ <arch-typeN> resname1 resname2 ... resnameN }

5.1.2. XPBS and PBS Commands

xpbs calls PBS commands as follows:

Command Button	PBS Command
detail (Hosts)	qstat -B -f <selected server_host(s)>
terminate	qterm <selected server_host(s)>
detail (Queues)	qstat -Q -f <selected queue(s)>
stop	qstop <selected queue(s)>
start	qstart <selected queue(s)>

enable	qenable <selected queue(s)>
disable	qdisable <selected queue(s)>
detail (Jobs)	qstat -f <selected job(s)>
modify	qalter <selected job(s)>
delete	qdel <selected job(s)>
hold	qhold <selected job(s)>
release	qrls <selected job(s)>
run	qrun <selected job(s)>
rerun	qrerun <selected job(s)>
rerun	qrerun <selected job(s)>
signal	qsig <selected job(s)>
msg	qmsg <selected job(s)>
move	qmove <selected job(s)>
order	qorder <selected job(s)>

5.2. xpbsmon

xpbsmon is the node monitoring GUI for PBS. It is used for displaying graphically information about execution hosts in a PBS environment. Its view of a PBS environment consists of a list of sites where each site runs one or more Servers, and each Server runs jobs on one or more execution hosts (nodes).

The system administrator needs to define the sites information in a global X resources file, *$PBS_LIB/xpbsmon/xpbsmonrc*, which is read by the GUI if a personal *.xpbsmonrc* file is missing. A default xpbsmonrc file usually would have been created already upon install, defining (under *sitesInfo resource) a default site name, list of Servers that run on a site, set of nodes (or execution hosts) where jobs on a particular Server run, and the list of queries that are communicated to each node's **pbs_mom**. If node queries have been specified, the host where xpbsmon is running must have been given explicit permission by the **pbs_mom** daemon to post queries to it. This is done by including a `$restricted` entry in the Mom's config file. See section 3.6 for more information on the restricted entry.

It is not recommended to manually update the *sitesInfo value in the xpbsmonrc file as its syntax is quite cumbersome. The recommended procedure is to bring up xpbsmon, click on "Pref.." button, manipulate the widgets in the Sites, Server, and Query Table dialog boxes, then click "Close" button and save the settings to a *.xpbsmonrc* file. Then copy this file over to *$PBS_LIB/xpbsmon*.

6. Operational Issues

This chapter addresses a few of the "day to day" operational issues which will arise.

6.1. Security

There are three parts to security in the batch system:

Internal security
: Can the daemons be trusted?

Authentication
: How do we believe a client about who it is.

Authorization
: Is the client entitled to have the requested action performed.

6.1.1. Internal Security

An effort has been made to insure the various PBS daemon themselves cannot be a target of opportunity in an attack on the system. The two major parts of this effort is the security of files used by the daemons and the security of the daemons 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. When PBS directories are installed, the make process runs a program to validate ownership and access to the files. This can be rechecked at any time by running `check-tree` in the top level make file. `check-tree` is located in the directory given by the value of bindir in configure. Each daemon also validates the most critical files and directories each time it is started.

A corrupted environment is another source of attack on a system. To prevent this type of attack, each daemon resets its environment when it starts. The source of the environment is a file named by PBS_ENVIRON set by the configure option --set-environ, defaulting to `{PBS_HOME}/pbs_environment`. If it does not already exists, this 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 **TZ** variable. It may be edited to include the other variables required on your system. Please note that **PATH** must be included. The value of **PATH** in pbs_mom's environment, from this file, 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 syntax of an PBS_ENVIRON file entry is either

```
variable_name=value
```

or

```
variable_name
```

In the later case, the value for the variable is obtained from the daemons environment before it is reset.

Other variables for the job's environment may also be obtained from Mom's environment, this list varies by system. If you are interested, see the list in the variable *obtain_vnames* in the source code of src/resmom/*/mom_start.c.

6.1.2. 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 (right or wrong) 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 included

with the request and it must match the IP network layer opinion as to the host's identity.

6.1.3. Host Authorization

Access to the pbs_server from another system may be controlled by an access control list (ACL). See section 10.1.1 of the ERS for details.

Access to pbs_mom is controlled through a list of hosts specified in their configuration files. By default, only "localhost" and the name returned by gethostname(2) are allowed. See the man pages pbs_mom(8B) for more information on the configuration file.

Access to the pbs_sched is not limited other than it must be from a privileged port.

6.1.4. User Authentication

Is the user who he/she claims to be?

The PBS Server authenticates the user name included in a request with the supplied PBS credential. This credential is supplied by **pbs_iff**(1B), see section 10.2 of the ERS.

6.1.5. User Authorization

Is the user entitled to make the request of the Server job under that name?

PBS as shipped assumes a consistent user name space within the set of systems which make up a PBS cluster. Thus if a job is submitted by *UserA@hostA*, PBS will allow the job to be deleted or altered by *UserA@hostB*. The routine *site_map_user*() is called twice. Once to map the name of the requester and again to map the job owner to a name on the Server's (local) system. If the two mapping agree, the requester is considered the job owner. See section 10.1.3 of the ERS. This behavior may be changed by a site by altering the Server routine site_map_user() found in the file src/server/site_map_user.c, see the Internal Design Spec.

Is the user entitled to execute the job under that name?

A user may supply a name under which the job is to executed on a certain system. If one is not supplied, the name of the job owner is chosen to be the execution name. See the *-u user_list* option of the **qsub**(1B) command. Authorization to execute the job under the chosen name is granted under the following conditions:

1. The job was submitted on the Server's (local) host and the submitter's name is the same as the selected execution name.
2. The host from which the job was submitted are declared trusted by the execution host in the /etc/hosts.equiv file or the submitting host and submitting user's name are listed in the execution users' .rhosts file. The system supplied library function, *ruserok*(), is used to make these checks.

If the above are not satisfactory to a site, the routine *site_check_user_map*() in the file src/server/site_check_u.c may be modified. See the IDS for more information.

In addition to the above checks, access to a PBS Server and queues within that Server may be controlled by access control lists. See section 10.1.1 and 10.1.2 of the ERS for more information.

6.1.6. Group Authorization

PBS allows a user to submit jobs and specify under which group the job should be executed. The user specifies a *group_list* attribute for the job which contains a list of `groups@hosts` similar to the user list. See the **group_list** attribute under the -W option of qsub(1B). The PBS Server will ensure that the user is a member of the

specified group by

1. Checking if the group is the user's primary group in the password entry. In this case the user's name does not have to appear in the group entry for his primary group.
2. Checking for the user's name in the specified group entry in /etc/group.

The job will be aborted if both checks fail. The checks are skipped if the user does not supply a group list attribute. In this case the user's primary group from the password file will be used.

When staging files in or out, PBS also uses the selected execution group for the copy operation. This provides normal UNIX access security to the files. Since all group information is passed as a string of characters, PBS cannot determine if a numeric string is intended to be a group name or GID.

Therefore when a group list is specified by the user, PBS places one requirement on the groups within a system. Each and every group in which a user might execute a job MUST have a group name and an entry in /etc/group. If no group lists are ever used, PBS will use the login group and will accept it even if the group is not listed in /etc/group. Note in this case, the **egroup** attribute value is a numeric string representing the user's gid rather than the group "name".

6.1.7. 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 on this or some other system, is listed in the Server attribute **acl_roots**.

6.2. Job Prologue/Epilogue Scripts

PBS provides the ability to run a site supplied script before and/or after each job runs. This provides the capability to perform initialization or cleanup of resources, such as temporary directories or scratch files. The scripts may also be used to write "banners" on the job's output files. When multiple nodes are allocated to a job, these scripts are only run by the "Mother Superior", the pbs_mom on the first node allocated. This is also where the job shell script is run.

If a prologue or epilogue script is not present, Mom continues in a normal manner. If present, the script is run with root privilege. In order to be run, the script must adhere to the following rules:

- The script must be in the *PBS_HOME*/mom_priv directory with the name prologue for the script to be run before the job and the name epilogue for the script to be run after the job.
- The script must be owned by root.
- The script must be readable and executable by root.
- The script cannot be writable by anyone but root.

The script may be a shell script or an executable object file. Typically, a shell script should start with a line of the form: #! interpreter.
See the rules under execve(2) or exec(2) on your system.

6.2.1. Prologue and Epilogue Arguments

When invoked, the prologue is called with the following arguments:

argv[1] is the job id.

argv[2] is the user name under which the job executes.

argv[3] is the group name under which the job executes.

The epilogue is called with the above, plus:

argv[4] is the job name.

argv[5] is the session id. †

argv[6] is the requested resource limits (list). †

argv[7] is the list of resources used.

argv[8] is the name of the queue in which the job resides. †

argv[9] is the account string, if one exists.

For both the prologue and epilogue:

envp The environment passed to the script is null.

cwd The current working directory is the user's home directory.

input When invoked, both scripts have standard input connected to a system dependent file. Currently, for all systems this file is /dev/null.

output With one exception, the standard output and standard error of the scripts are connected to the files which contain the standard output and error of the job. 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.

6.2.2. Prologue Epilogue Time Out

To prevent a bad script or error condition within the script from delaying PBS, Mom places an alarm around the scripts execution. This is currently set to 30 seconds. If the alarm sounds before the scripts has terminated, Mom will kill the script. The alarm value can be changed by changing the define of **PBS_PROLOG_TIME** within src/resmom/prolog.c.

6.2.3. Prologue Error Processing

Normally, the prologue script should exit with a zero exit status. Mom will record in her log any case of a non-zero exit from a script. Exit status values and their impact on the job are:

-4 The script timed out (took too long). The job will be requeued.

-3 The wait(2) call waiting for the script to exit returned with an error. The job will be requeued.

-2 The input file to be passed to the script could not be opened. The job will be requeued.

-1 The script has a permission error, it is not owned by root and or is writable by others than root. The job will be requeued.

0 The script was successful. The job will run.

1 The script returned an exit value of 1, the job will be aborted.

>1 The script returned a value greater than one, the job will be requeued.

The above apply to normal batch jobs. Note, interactive-batch jobs (-I option) cannot be requeued on a non-zero status, the network connection back to qsub is lost and cannot be re-established. Interactive jobs will be aborted on any non-zero prologue exit.

The administrator must exercise great caution in setting up the prologue to prevent jobs from being flushed from the system.

Epilogue script exit values are logged, if non-zero, but have no impact on the state of the job.

6.3. Use and Maintenance of Logs

The PBS system tends to produce lots of log file entries. There are two types of logs, the event logs which record events within each PBS daemon (pbs_server, pbs_mom, and pbs_sched) and the Server's accounting log.

6.3.1. The Daemon Logs

Each PBS daemon maintains an event log file. The details of the log format is covered in section **3.3.8. Event Logging** of the ERS. The Server (pbs_server), Scheduler (pbs_sched), and Mom (pbs_mom) default their logs to a file with the current date as the name in the PBS_HOME/(daemon)_logs directory. This location can be overridden with the "-L pathname" option; pathname must be an absolute path.

If the default log file name is used, no -L option, the log will be closed and reopened with the current date daily. This happens on the first message after midnight. If a path is given with the -L option, the automatic close/reopen does not take place. All daemons will close and reopen the same named log file on receipt of SIGHUP. The pid of the daemon is available in its lock file in its home directory. Thus it is possible to move the current log file to a new name and send SIGHUP to restart the file:

```
cd PBS_HOME/daemon_logs
mv current archive
kill -HUP `cat ../daemon_priv/daemon.lock`
```

The amount of output in the logs depends on the selected events to log and the presence of debug writes, turned on by compiling with -DDEBUG. The Server and Mom can be directed to record only messages pertaining to certain event types. The specified events are logically "or-ed". Their decimal values are:

1 Error Events

2 Batch System/Server Events

4 Administration Events

8 Job Events

16 Job Resource Usage (hex value 0x10)

32 Security Violations (hex value 0x20)

64 Scheduler Calls (hex value 0x40)

128 Debug Messages (hex value 0x80)

256 Extra Debug Messages (hex value 0x100)

Everything turned on is of course 511. 127 is a good value to use. The event logging mask is controlled differently for the Server and Mom. The Server's mask is set via qmgr(1B) setting the **log_events** attribute. This can be done at any time. Mom's mask may be set via her configuration file with a `$logevent` entry, see the -c option on pbs_mom. To change her logging mask, edit the configuration file and send Mom a SIGHUP signal.

The Scheduler, being site written may have a different method of changing its event logging mask, or it may not have the ability at all.

6.3.2. The Accounting Log

The PBS Server daemon maintains an accounting log. The format of the log is described in section **3.3.9 Accounting** of the ERS. The log name defaults to `PBS_HOME/server_priv/accounting/yyyymmdd` where yyyymmdd is the date. The accounting may be placed elsewhere 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, for example

```
pbs_server -A ""
```

then the accounting log will not be opened and no accounting records will be recorded.

The accounting file is changed according to the same rules as the 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 and reopen it upon the receipt of a SIGHUP signal. This allows you to rename the old log and start recording anew on an empty file. For example, if the current date is February 9 the Server will be writing in the file `19990209`. The following actions will cause the current accounting file to be renamed `feb1`, and the Server to close the file and starting writing a new `19990209`.

```
mv 19990201 feb1
kill -HUP 1234     (the Server's pid)
```

6.4. Alternate Test Systems

Alternate or test copies of the various daemons may be run through the use of the command line options which set their home directory and service port. For example, the following commands would start the three daemons with a home directory of `/tmp/altpbs` and four ports around 13001, the Server on 13001, Mom on 13002 and 13003, and the Scheduler on 13004.

```
pbs_server -t create -d /tmp/altpbs -p 13001 -M 13002 -R 13003 -S 13004
pbs_mom -d /tmp/altpbs -M 13002 -R 13003
pbs_sched -d /tmp/altpbs -S 13004 -r script_file
```

The home directories must be pre-built. The easiest method is to alter the **PBS_HOME** variable by use of the `--set-server-home` option to configure, rerun configure and remake PBS.

Jobs may be directed to the test system by using the `server:port` syntax on the -q option. Status is also obtained using the `:port` syntax: For example, to submit a job to the default queue on the above test Server, request the status of the test Server, and request the status of jobs at the test Server:

```
qsub -q @host:13001 job
qstat -Bf host:13001
qstat @host:13001
```

If you or users are using job dependencies on or between test systems, there are minor problems of which you (and the users) need to be aware. The syntax of both the dependency string, `depend_type:job_id:job_id` and the job id `seq_number.host:port` use colons in an indistinguishable manner. The way to work around this is covered in the **Advice for Users** section at the end of this guide.

6.5. Installing an Updated Batch System

Once you have a running batch system, there will come a time when you wish to update it or install a new version. It is assumed that you will wish to build and test the new version using alternative directories and port numbers described above. You may change the location of PBS_HOME for the test version, see configure option `--set-server-home`. Once you are satisfied with the new system, it is suggested that you rebuild the three daemons with PBS_HOME set to directory which will be used in normal operation. Otherwise you will always have to use the -d option when starting the daemons.

When the new batch system is ready to be placed into service, you will wish to move jobs from the old system to the new. The following procedure is suggested. All Servers must be run by root. The qmgr and qmove commands should be run by an batch administrator (likely, root is good).

1. With the old batch system running, disable the queues and stop scheduling by setting "scheduling=false".
2. Backup the pool of jobs in PBS_HOME(old)/server_priv/jobs. Tar may used for this.

Assuming the change is a minor update (change in third digit of the release version number) or a local change where the job structure did not change from the old version to the new, it is likely that you could start the new system in the old HOME and all jobs would be recovered. However if the job structure has changed you will need to *move* the jobs from the old system to the new. The release notes will contain a warning if the job structure has changed or the move is required for other reasons.

To move the jobs, continue with the following steps:

3. It is likely that PBS_HOME will have changed and have been made during testing. If not, build a (temporary) server directory tree by changing PBS_HOME using --set-server-home and typing
   ```
   "buildutils/pbs_mkdirs server"
   ```
   while in the top of the object tree.
4. Start the new PBS Server in its new home. If the new home is different from the directory when it was compiled, use the -d option. Use the -t option if the Server has not been configured for the new directory. Also start with an alternative port using the -p option. Turn off attempts to schedule with the -a option:
   ```
   pbs_server -t create -d new_home -p 13001 -a false
   ```
   Remember, you will need to use the `:port` syntax when commanding the new Server.
5. Duplicate on the new Server the current queues and server attributes (assuming you wish to do so). Enable each queue which will receive jobs at the new Server.
   ```
   qmgr -c "print server" > /tmp/config
   qmgr host:13001  < /tmp/config
   qenable queue1@host:13001
   qenable queue2@host:13001
   ```
6. Now list the jobs at the original Server and move a few jobs one at a time from the old to the new Server:
   ```
   qstat
   qmove queue@host:13001 job
   qstat @host:13001
   ```
   If all is going well, move the remaining jobs a queue at a time:
   ```
   qmove queue1@host:13001 ‘qselect -qqueue1‘
   qstat queue1@host:13001
   qmove queue2@host:13001 ‘qselect -qqueue2‘
   qstat queue2@host:13001
   ```
7. At this point, all of the jobs should be under control of the new Server and located in the new Server's home. If the new Server's home is a temporary directory, shut down the new Server and move everything to the real home using
   ```
   cp -R new_home real_home
   ```
   or, if the real (new) home is already set up,
   ```
   cd new_home/server_priv/jobs
   cp * real_home/server_priv/jobs
   ```
   to copy just the jobs.

At this point, you are ready to bring up and enable the new batch system.

You should be aware of one quirk when using qmove. If you wish to move a job from a Server running on a test port to the Server running on the normal port (15001), you may attempt, *unsuccessfully*, to use the following command:

```
qmove queue@host 123.job.host:13001
```

However, that will only move the job to the end of the queue it is already in. The Server receiving the move request (13001), will compare the destination server name, host, with its own name only, not including the port. Hence it will match and it will not send the job where you intended. To get the job to move to the Server running on the normal port you have to specify that port in the destination:

```
qmove queue@host:15001 123.job.host:13001
```

6.6. Problem Solving

The following is a very incomplete list of possible problems and how to solve them.

6.6.1. Clients Unable to Contact Server

If a client command, qstat, 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 in the "default server file" specified when the commands where built and installed.

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 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 built with a different port number, use that number instead of 15001). pbs_iff should print out a string of garbage characters and exit with a status of 0. The garbage is the encrypted credential which would be used by the command to authenticate the client to the Server. If pbs_iff fails to print the garbage and/or exits with a non-zero status, either the Server is not running or was built with a different encryption system than was pbs_iff.

6.6.2. Nodes Down

The PBS Server determines the state (up or down), by communicating with Mom on the node. The state of nodes may be listed by two commands qmgr and pbsnodes: **Qmgr:** `list nodes @active` or `pbsnodes -a`. A node in PBS may be marked "down" in one of two substates.

If the node is listed as

```
Node lensmen
        state = down, state-unknown
        properties = sparc, mine
        ntype = cluster
```

then the Server has not had contact with Mom since the Server came up. Check to see if a Mom is running on the node. 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 node is still marked "down, state-unknown" after 10+ minutes, either the node 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 node.

If the node is listed as

```
Node lensmen
        state = down
        properties = sparc, mine
        ntype = cluster
```

then the Server has been able to ping Mom on the node in the past, but she has not responded recently. The Server will send a "ping" PBS message to every free node each ping cycle, 10 minutes. If a node does not acknowledge the ping before the next cycle, the Server will mark the node down. On a IBM SP, a node may also be marked down if Mom on the node believes that the node is not connected to the high speed switch. When the Server receives an acknowledgement from Mom on the node, the node will again be marked up (free).

6.6.3. 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) destination host is not trusted and the user does not have a .rhost file, (2) An improper path was specified, (3) a directory is not writable, and (4) the user's .cshrc on the destination host generates output when executed. This are explained fully in the section "Delivery of Output Files" in the next chapter.

6.6.4. 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 OM's log. Typical reasons are a bad user/group account, checkpoint/restart file (Cray), 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.

6.6.5. 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. The fact that it happens even rarely means there is a bug in PBS (*gasp! Oh the horror of it all.*).

If this situation is noted, PBS offers a way out. Use the qsig command to send SIGNULL, signal 0, to the job. If Mom notes there are not any processes then she will force the job into the exiting state.

6.6.6. Dependent Jobs and Test Systems

If you have users running on a test batch system using an alternative port number, -p option to pbs_server, problems may occur with job dependency if the following requirements are not observed:

1. For a test system, the job identifier in a dependency specification must include at least the first part of the host name.
2. The colon in the port number specification must be escaped by a black slash. This is true for both the Server and current server sections.

For example:
`123.test_host\:17000`
`123.old_host@test_host\:17000`
`123.test_host\:17000@diff_test_host\:18000` On a shell line, the back slash itself must be escaped from the shell, so the above become:

```
123.test_host\\:17000
123.old_host@test_host\\:17000
123.test_host\\:17000@diff_test_host\\:18000
```

These rules are not documented on the qsub/qalter man pages since the likely hood of the general user community finding themselves seting up dependencies with jobs on a test system is small and the inclusion would be generally confusing.

6.7. Communication with the User

Users tend to want to know what is happening to their job. PBS provides a special job attribute, *comment*, which is available to the operator, manager, or the Scheduler program. This attribute can be set to a string to pass information to the job owner. It might be used to display information about why the job is not being run or why a hold was placed on the job. Users are able to see this attribute when it is set by using the -f option of the qstat command. A Scheduler program can set the comment attribute via the pbs_alterjob() API. Operators and managers may use the -W option of the qalter command, for example

```
qalter -W comment="some text" job_id
```

7. Advice for Users

The following sections provide information necessary to the general user community concerning use of PBS. Please make this information available.

7.1. Modification of User shell initialization files

A user's job may not run if the user's start-up files (.cshrc, .login, or .profile) contain commands which attempt to set terminal characteristics. Any such activity should be skipped by placing a test of the environment variable **PBS_ENVIRONMENT** (or for NQS compatibility, **ENVIRONMENT**). This can be done as shown in the following sample .login:

```
setenv PRINTER printer_1
setenv MANPATH /usr/man:/usr/local/man:/usr/new/man
if ( ! $?PBS_ENVIRONMENT ) then
    do terminal stuff here
endif
```

If the user's login shell is csh, the following message may appear in the standard output of a job:

```
Warning: no access to tty, thus no job control in this shell
```

This message is produced by many csh versions when the shell determines that its input is not a terminal. Short of modifying csh, there is no way to eliminate the message. Fortunately, it is just an informative message and has no effect on the job.

7.2. Parallel Jobs

If you have set up PBS to manage a cluster of systems or on a parallel system, it is likely with the intent to manage parallel jobs. As discussed in section **2.1 Planning** and **3.2 Multiple Execution Systems**, PBS allocated nodes to one job at a time, called space-sharing. It is important to remember that the entire node is allocated to the job regardless of the number of processors or the amount of memory in the node.

To have PBS allocate nodes to a user's job, the user must specify how many of what type of nodes are required for the job. Then the user's parallel job must execute tasks on the allocated nodes.

7.2.1. How User's Request Nodes

The *nodes* resources_list item is set by the user to declare the node requirements for the job. It is a string of the form

```
-l nodes=node_spec[+node_spec...]
```

where node_spec is

```
number | property[:property...] | number:property[:property...]
```

The node_spec may have an optional global modifier appended. This is of the form `#property`. For example:

```
6+3:fat+2:fat:hippi+disk
```

or

```
6+3:fat+2:fat:hippi+disk#prime.
```

Where fat, hippi, and disk are examples of property names assigned by the administrator in the `{PBS_HOME}/server_priv/nodes` file. The above example translates as the user requesting 6 plain nodes plus 3 "fat" nodes plus 2 nodes that are both "fat" and "hippi" plus one "disk" node, a total of 12 nodes. Where #prime is appended as a global modifier, the global property, "prime" is appended by the Server to each element of the spec. It would be equivalent to

```
6:prime+3:fat:prime+2:fat:hippi:prime+disk:prime .
```

A major use of the global modifier is to provide the *shared* keyword. This specifies that all the nodes are to be temporarily-shared nodes. The keyword shared is only recognized as such when used as a global modifier.

7.2.2. Parallel Jobs and Nodes

PBS provides a means by which a parallel job can spawn, monitor and control tasks on remote nodes. See the man page for tm(3). *Unfortunately*, no vendor has made use of this capability though several contributed to its design. Therefore, spawing the tasks of a parallel job fall to the parallel environment itself. PVM provides one means by which a parallel job spawns processes via the pvmd daemon. MPI typically has a vendor dependent method, often using rsh or rexec.

All of these means are outside of PBS's control. PBS cannot control or monitor resource usage of the remote tasks, only the ones started by the job on Mother Superior. PBS can only make the list of allocated nodes available to the parallel job and hope that the vendor and the user make use of the list and stay within the allocated nodes.

The names of the allocated nodes are place in a file in {PBS_HOME}/aux. The file is owned by root but world readable. The name of the file is passed to the job in the environment variable **PBS_NODEFILE**. For IBM SP systems, it is also in the variable MP_HOSTFILE.

If you are running an open source version of MPI, such as MPICH, then the mpirun command can be modified to check for the PBS environment and use the PBS supplied host file.

7.3. 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 in one of two ways depending on how PBS was installed.

Name of Script on Standard Input

The default method (PBS built with --enable-shell-pipe) is to pass 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:

+ 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.
- An extra shell process is run to process the job script.
- If the script does not include a #! line as the first line, the wrong shell may attempt to interpret the script producing syntax 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.

Script as Standard Input

The alternative method for PBS (built with --disable-shell-invoke), is to open the script file as standard input for the shell. This is equivalent to typing `shell < script`. This also offers advantages and disadvantages:

+ The user's script will always be directly processed by the user's login shell.
+ If the user specifies a non-standard shell (any old program) with the -S option, the script can be read by that program as its input.

- If a command within the job script reads from standard input, it may read lines from the script depending on how far ahead the shell has buffered its input. Any command line so read will not be executed by the shell. A command that reads from standard input with out explicit redirection is generally unwise in a batch job.

The choice of shell invocation methods is left to the site. It is recommended that all PBS execution servers (pbs_mom) within that site be built to use the same shell invocation method.

7.4. Job Exit Status

The exit status of a job is normally the exit status of the shell executing the job script. If a user is using *csh* and has a *.login* file in the home directory, the exit status of csh becomes the exit status of the last command in .logout. This may impact the use of job dependencies which depend on the job's exit status. To preserve the job's status, the user may either remove .logout or add the following two lines to it. Add as the first line:

```
set EXITVAL = $status
```

and as the last executable line:

```
exit $EXITVAL
```

7.5. Delivery of Output Files

To transfer output files or to tranfer staged-in or staged-out files to/from a remote destination, PBS uses either rcp or scp depening on the configuration options. PBS includes the source of a version of the **rcp**(1) command, from the bsd 4.4 lite distribution. The resulting object program, **pbs_rcp**(1B), is used. This version of rcp is provided because it, unlike some rcp implementation, always exits with a non-zero exits status for any error. Thus Mom knows if the file was delivered or not. Fortunately, the secure copy program, scp, is also based on this version of rcp and exits with the proper status code.

Using rcp, the copy of output or staged files can fail for (at least) two reasons.

1. If the user's .cshrc script outputs any characters to standard output, e.g. contains an echo command, **pbs_rcp** will fail. See the section in this document entitled **Modification of User shell initialization files**.
2. The user must have permission to *rsh* to the remote host. Output is delivered to the remote destination host with the remote file owner's name being the job owner's name (job submitter). On the execution host, the file is owned by the user's execution name which may be different. For information, see the `-u user_list` option on the **qsub**(1) command.

   If the two names are identical, permission to rcp may be granted at the system level by an entry in the destination host's */etc/host.equiv* file calling out the execution host.

   If the owner name and the execution name are different or if the destination host's /etc/hosts.equiv file does not contain an entry for the execution host, the user must have an ".rhosts" file in her home directory of the system to which the output files are being returned. The .rhosts must contain an entry for the system on which the job executed with the user name under which the job was executed. It is wise to have two lines, one with just the "base" host name and one with the full *host.domain_name*.

If PBS is built to use the *Secure Copy Program*, scp, then PBS will first try to deliver output or stage-in/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.

For delivery of output files on the local host, PBS uses the **/bin/cp**(1) command. Local and remote Delivery of output may fail for the following additional reasons:

1. A directory in the specified path does not exist.
2. A directory in the specified path is not searchable by the user.
3. The target directory is not writable by the user.

Additional information as to the cause of the delivery problem might be determined from Mom's log file. Each failure is logged. The various error codes are described in requests.c/sys_copy() in the IDS.

7.6. Stage in and Stage out problems

The same requirements and hints discussed above in regard to delivery of output apply to staging files in and out. It may also be useful to note that the stage-in and stage-out option on qsub both take the form

```
local_file@remote_host:remote_file
```

regardless of the direction of transfer. Thus for stage-in, the direction of travel is

local_file <-- remote_host:remote_file

and for stage out, the direction of travel is

local_file --> remote_host:remote_file

Also note that all relative paths are relative to the user's home directory on the respective hosts. PBS uses rcp or scp (or cp if the remote host is the local host) to perform the transfer. Hence, a stage-in is just a

rcp -r remote_host:remote_file local_file

and a stage out is just

rcp -r local_file remote_host:remote_file

As with rcp, the remote_file may be a directory name. Also as with rcp, the local_file specified in the stage in/out directive may name a directory. For stage-in, if remote_file is a directory, then local file must also be a directory. For stage out, if local_file is a directory, then remote_file must also be a directory.

If *local_file* on a stage out directive is a directory , that directory on the execution host, including all files and subdirectories, will be copied. At the end of the job, the directory, including all files and subdirectories, will be deleted. Users should be aware that this may create a problem if multiple jobs are using the same directory.

Stage in presents another problem. Assume the user wishes to stage-in the contents of a single file named *poo* and gives the following stage-in directive:

```
-W stagein=/tmp/bear@somehost:poo
```

If /tmp/bear is an existing directory, the local file becomes /tmp/bear/poo. When the job exits, PBS will determind that */tmp/bear* is a directory and append */poo* to it. Thus */tmp/bear/poo* will be deleted. If however, the user wishes to stage-in the contents of a directory named *cat* and gives the following stage-in directive:

```
-W stagein=/tmp/dog/newcat@somehost:cat
```

where */tmp/dog* is an existing directory, then at job end, PBS will determine that */tmp/dog/newcat* is a directory and append */cat* and then fail on the attempt to delete */tmp/dog/newcat/cat*.

On stage-in when remote_file is a directory, the user should not specify a new directory as local_name. In the above case, the user should go with

```
-W stagein=/tmp/dog@somehost:cat
```

which will produce */tmp/dog/cat* which will match what PBS will try to delete at job's end.

Wildcards should not be used in either the local_file or the remote_file name. PBS does not expand the wildcard character on the local system. If wildcards are used in the remote_file name, since rcp is launched by rsh to the remote system, the expansion will occur. However, at job end, PBS will attempt to delete the file whose name actually contains the wildcard character and will fail to find it. This will leave all the staged in files in place (undeleted).

7.7. Checkpointing MPI Jobs on SGI Systems

Under Irix 6.5 and later, MPI parallel jobs as well as serial jobs can be checkpointed and restarted on SGI systems provided certain criteria are met. SGI's checkpoint system call cannot checkpoint processes that have open sockets. Therefore it is necessary to tell mpirun to not create or to close an open socket to the array services daemon used to start the parallel processes. One of two options to mpirun must be used:

-cpr This option directs mpirun to close its connection to the array services daemon when a checkpoint is to occur.

-miser This option directs mpirun to directly create the parallel process rather than use the array services. This avoids opening the socket connection at all.

The -miser option appears the better choice as it avoids the socket in the first place. If the -cpr option is used, the checkpoint will work, but will be slower because the socket connection must be closed first.

Note, interactive jobs or MPMD jobs (more than one executable program) can not be checkpointed in any case. Both use sockets (and TCP/IP) to communicate, outside of the job for interactive jobs and between programs in the MPMD case.

8. Customizing PBS

Most sites find that PBS works for them with only configuration changes. As their experience with PBS grows, many sites find it useful to customize the supplied Scheduer or to develop one of their own to meet very specific policy requirements. Custom Schedulers have been written in C, BaSL or Tcl.

This section addresses several ways that PBS can be customized for your site. While having the source code is the first step, there are specific actions other than modifying the code you can take.

8.1. Additional Build Options

Two header files within the subdirectory src/include provide additional configuration control over the Server and Mom. The modification of any symbols in the two files should not be undertaken lightly.

8.1.1. pbs_ifl.h

This header file contains structures, symbols and constants used by the API, libpbs.a, and the various commands as well as the daemons. Very little here should ever be changed. Possible exceptions are the following symbols. They must be consistent between all batch systems which might interconnect.

PBS_MAXHOSTNAME
: Defines the length of the maximum possible host name. This should be set at least as large as **MAXHOSTNAME** which may be defined in *sys/params.h*.

PBS_MAXUSER
: Defines the length of the maximum possible user login name.

PBS_MAXGRPN
: Defines the length of the maximum possible group name.

PBS_MAXQUEUENAME
: Defines the length of the maximum possible PBS queue name.

PBS_USE_IFF
: If this symbol is set to zero (0), before the library and commands are built, the API routine pbs_connect() will not attempt to invoke the program **pbs_iff** to generate a secure credential to authenticate the user. Instead, a clear text credential will be generated. This credential is completely subject to forgery and is useful only for debugging the PBS system. You are strongly advised against using a clear text credential.

PBS_BATCH_SERVICE_PORT
: Defines the port number at which the Server listens.

PBS_MOM_SERVICE_PORT
: Defines the port number at which Mom, the execution miniserver, listens.

PBS_SCHEDULER_SERVICE_PORT
: Defines the port number at which the Scheduler listens.

8.1.2. server_limits.h

This header file contains symbol definitions used by the Server and by Mom. Only those that *might* be changed are listed here. These should be changed with care. It is strongly recommended that no other symbols in server_limits.h be changed. If server_limits.h is to be changed, it may be copied into the include directory of the *target* (build) tree and modified before compiling.

NO_SPOOL_OUTPUT
: If defined, directs Mom to not use a spool directory for the job output, but to place

it in the user's home directory while the job is running. This allows a site to invoke quota control over the output of running batch jobs.

PBS_BATCH_SERVICE_NAME
: This is the service name used by the Server to determine to which port number it should listen. It is set to `pbs`, in quotes as it is a character string. Should you wish to assign PBS a service port in `/etc/services`, change this string to the service name assigned. You should also update PBS_SCHEDULER_SERVICE_NAME as required.

PBS_DEFAULT_ADMIN
: Defined to the name of the default administrator, typically "root". Generally only changed to simplify debugging.

PBS_DEFAULT_MAIL
: Set to user name from which mail will be sent by PBS. The default is "adm". This is overridden if the Server attribute *mail_from* is set.

PBS_JOBBASE
: The length of the job id string used as the basename for job associated files stored in the spool directory. It is set to 11, which is 14 minus the 3 characters of the suffixes like `.JB` and `.OU`. Fourteen is the guaranteed length for a file name under POSIX. The actual length that a file name can be depends on the file system and must be determined at run time, but PBS is too lazy to go to that trouble. If the Server and Mom run on a file system that support longer names (most do), then you may up this value so that the names are more readable.

PBS_MAX_HOPCOUNT
: Used to limit the number of hops taken when being routed from queue to queue. It is mainly to detect loops.

PBS_NET_MAX_CONNECTIONS
: The maximum number of open file descriptors and sockets supported by the server.

PBS_NET_RETRY_LIMIT
: The limit on retrying requests to remote servers.

PBS_NET_RETRY_TIME
: The time between network routing retries to remote queues and for requests between the Server and Mom.

PBS_RESTAT_JOB
: To refrain from over burdening any given Mom, the Server will wait this amount of time (default 30 seconds) between asking her for updates on running jobs. In other words, if a user asks for status of a running job more often than this value, the prior data will be returned.

PBS_ROOT_ALWAYS_ADMIN
: If defined (set to 1), "root" is an administrator of the batch system even if not listed in the `managers` attribute.

PBS_SCHEDULE_CYCLE
: The default value for the elapsed time between scheduling cycles with no change in jobs queued. This is the initial value used by the Server, but it can be changed via **qmgr**(1B).

8.2. Site Modifiable Source Files

It is safe to skip this section until you have played with PBS for a while and want to start tinkering.

Dave Tweten of NASA has said, "If it ain't source, it ain't software." This is part of PBS's philosophy that source distribution should be a major part of any software product. Otherwise, the product becomes "hard"−ware. The first example of this philosophy is the PBS job Scheduler. The implementation of the site policy is left to the site. PBS provides three tools for that implementation, the BaSL Scheduler, the Tcl Scheduler, and the C Scheduler.

The philosophy does not stop with the Scheduler. With distribution of the source, a site has the ability to modify any part of PBS as they so choose. Of course, indiscriminate modification is not without dangers. Not the least of which is conflicts with future releases by the developers.

Certain functions of PBS appear to be likely targets of widespread modification by sites for a number of reasons. When identified, the developers of PBS have attempted to improve the easy of modification in these areas by the inclusion of special *site specific modification routines*. These are identified in the IDS under chapter headings of "Site Modifiable Files" in the sections on the Server and Mom. The distributed default version of these files build a private library, libsite.a, which is include in the linking phase for the Server and for Mom. They may be replaced as needed by a site. The procedure is described in the IDS under "libsite.a − Site Modifiable Library" in Chapter 10.

The files include:

Server

site_allow_u.c
: The routine in this file, *site_allow_u()*, provides an additional point at which a user can be denied access to the batch system (server). It may be used instead of or in addition to the Server Acl_User list.

site_alt_rte.c
: The function *site_alt_router()* allows a site to add decision capabilities to job routing. This function is called on a per-queue basis if the queue attribute **alt_router** is true. As provided, site_alt_router() just invokes the default router, *default_router()*.

site_check_u.c
: The routine in this file, *site_check_user_map()*, provides the service of authenticating that the job owner is privileged to run the job under the user name specified or selected for execution on the Server system. Please see the IDS for the default authentication method.

site_map_usr.c
: For sites without a common user name/uid space, this function, *site_map_user()*, provides a place to add a user name mapping function. The mapping occurs at two times. First to determine if a user making a request against a job is the job owner, see "User Authorization". Second, to map the submitting user (job owner) to an execution uid on the local machine.

site_*_attr_*.h
: These files provide a site with the ability to add local attributes to the server, queues, and jobs. The files are installed into the target tree "include" subdirectory during the first make. As delivered, they contain only comments. If a site wishes to add attributes, these files can be *carefully* modified.

  The files are in three groups, by server, queue, and job. In each group are *site_*_attr_def.h* files which are used to defined the name and support functions for the new attribute or attributes, and *site_*_attr_enum.h* files which

insert a enumerated label into the set for the corresponding parent object. For server, queue, node attributes, there is also an additional file that defines if the qmgr(1) command will include the new attribute in the set "printed" with the `print server`, `print queue`, or `print node` sub-commands.

Detailed information on how to modify these files can be found in the IDS under the "Site Modifiable Files" section of the Server, Chapter 5. You should note that just adding attributes will have no effect on how PBS processes jobs. The main usage for new attributes would be in providing new Scheduler controls and/or information. The scheduling algorithm will have to be modified to use the new attributes. If you need Mom to do something different with a job, you will still need "to get down and dirty" with her source code.

Mom

site_mom_chu.c
: If a server is feeding jobs to more than one Mom, additional checking for execution privilege may be required at Mom's level. It can be added in this function *site_mom_chkuser()*.

site_mom_ckp.c
: Provide post-checkpoint, *site_mom_postchk()* and pre-restart *site_mom_prerst()* "user exits" for the Cray and SGI systems.

site_mom_jset.c
: The function *site_job_setup()* allows a site to perform specific actions once the job session has been created and before the job runs.