PBS Portable Batch System

Administrator Guide

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

Numerical Aerospace Simulation Systems Division
NASA Ames Research Center

Release: 1.1.12
Printed: August 7, 1998

† MRJ Technology Solutions., NASA Contract NAS 2-14303, Moffett Field, CA

Portable Batch System, Rights to Use and Redistribute

This program is derived from prior work some of which was originally written 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.

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

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.

You may note change marks (bars) in the margin of this document. They are in the form of a single hex digit number which represents the last part of the PBS revision number in which the change appeared. For example "9" indicates release 1.1.9, "0" indicates release 1.1.10 and "2" indicates release 1.1.12. After a few (unspecified) revisions, the older change marks will be removed.

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 is called *Mom* as it is the mother of all executing jobs, and *pbs_mom*. 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.

Installation

This section attempts to explain the steps to build and install PBS. PBS installation is 2
accomplished via the GNU autoconf process. This installation procedure requires more man- 2
ual configuration than is ''typical'' for many packages. There are a number of options which 2
involve site policy and therefore cannot be determined automagically.

1.3. Release Contents

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.
- 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.

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 and library are required. If the BaSL scheduler is used, yacc and lex (or GNU bison and flex) are required. Possible sites for Tcl are:

http://sunscript.sun.com/
ftp://ftp.smli.com/pub/tcl
ftp://ftp.sunlabs.com/pub/tcl/

Versions of Tcl previous to 8.0 can no longer be used with PBS. Tcl and Tk version 2
8.0 or greater must be used.

A possible site for yacc and lex is:

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

1.4. Build Overview 2

The normal PBS build procedure is to separate the source from the target. This allows the 2
placement of a single copy of the source on a shared file system from which multiple different 2
target systems can be built. Also, the source can be protected from accidental destruction or 2
modification by making the source read-only. However, if you choose, objects may be made 2
within the source tree. 2

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

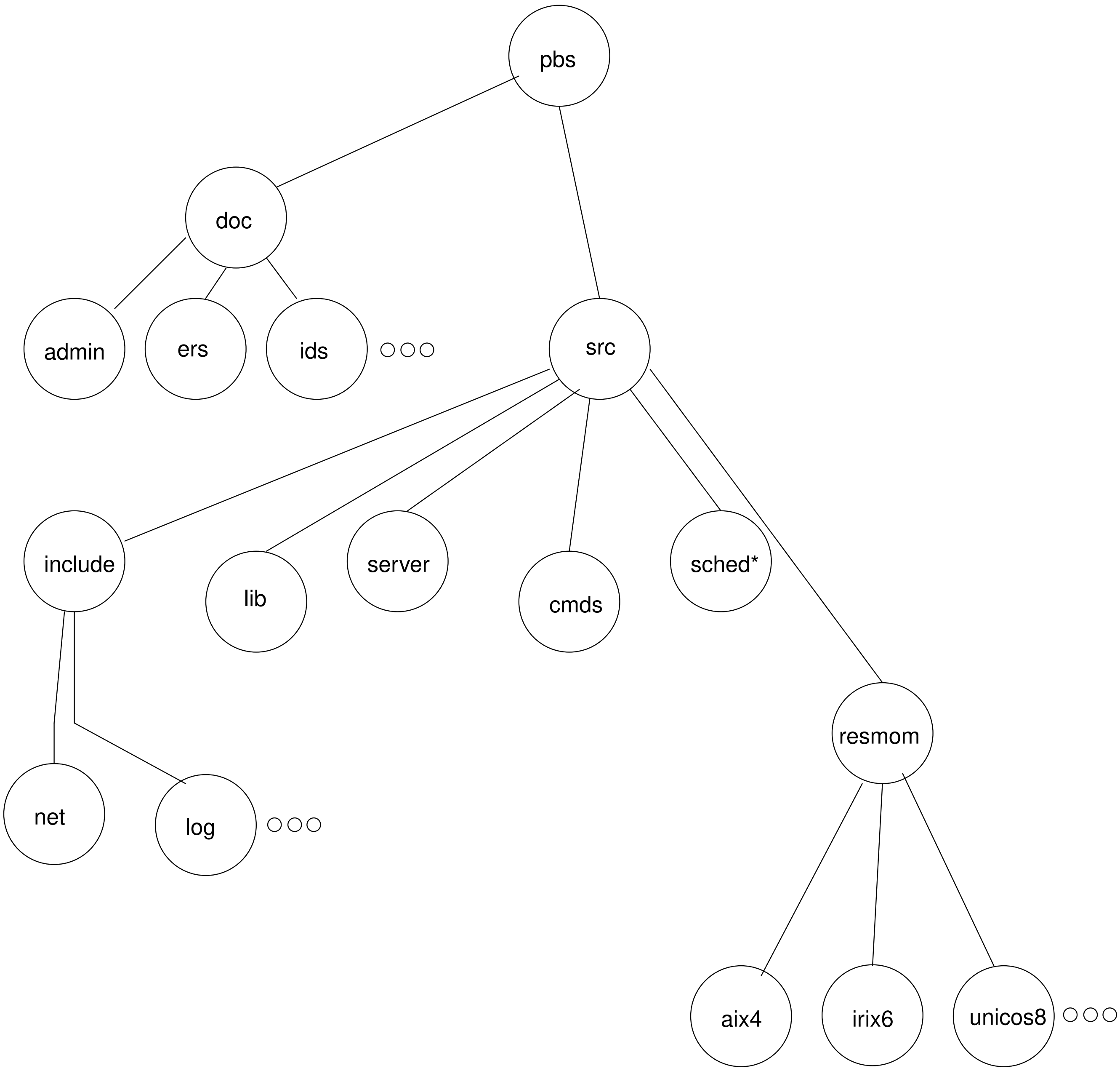

Figure 1−1: Source Tree Structure

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

An overview of the installation steps is listed here. Detailed explanation of symbols will fol- 2
low. It is recommended that you read completely through these instructions before beginning 2
the installation. To install PBS: 2

1. Place the tar file on the system where you would like to maintain the source. 2
2. Untar the tar file. It will untar in the current directory producing several files and sub- 2
   directories. This is the *source tree*. You may write protect the source tree at this point 2
   should you so choose. 2

In the top directory are two files, named "How_to_Install" and "INSTALL". The 2
"How_to_Install" file is actually a set of release notes containing information about the 2
release contents and pointing to this guide for installation instructions. The 2
"INSTALL" file consists of standard notes about the use of GNU's configure. 2

3. If you choose to have separate build and src trees create the top level directory of what 2
will become the *target tree* at this time. This need not be within the source tree. It 2
must reside on a system of the same architecture as the target system for which you are 2
generating the PBS binaries. This may well be the same system as holds the source or 2
it may not. Change directories to the top of the target tree. 2

4. Make a job scheduler choice. A unique feature of PBS is its external scheduler module. 2
This allows a site to implement any policy of its choice. To provide even more freedom 2
in implementing policy, PBS provides three scheduler frameworks. Schedulers may be 2
developed in the C language, the Tcl scripting language, or PBS's very own C language 2
extensions, the **Ba**tch **S**cheduling **L**anguage, or BaSL. 2

As distributed, *configure* will default to a C language based scheduler known as *fifo*. 2
This scheduler can be configured to several common simple scheduling policies, not just 2
first in – first out as the name suggests.. When this scheduler is installed, certain con- 2
figuration files are installed in PBS_HOME/scheduler_priv/. **You will need to modify** 2
these files for your site. These files are discussed in chapter 4.5 ``C based Sample 2
Scheduler'', in the section ``FIFO Scheduler''. 2

To change the selected scheduler, see the configure options **--set-sched** and **--set-** 2
sched-code in the Features and Package Options section of this chapter. Additional 2
information on the types of scheduler and how to configure fifo can be found in the 2
Scheduling Policies chapter later in this guide. 2

5. From within the top of the target tree, type the following command 2

```
{source_tree}/configure [options]
```
2

Where `{source_tree}` is the full relative or absolute path to the configure script in 2
the source tree. If you are building in the source tree type `./configure` at the top level 2
of the source tree where the configure script is found. 2

This will generate the complete target tree and a set of header files and makefiles used 2
to build PBS. *Note,* unlike in prior versions of PBS, the top two subdirectories will be 2
`doc` and `src` rather than `doc` and `obj`; this is a result of the one-for-one copy of the tree 2
made by configure. This step will only need to be redone if you choose to change options 2
specified on the configure command line. See section **2.3 Build Details** for information 2
on the configure options. **There are certain options which you must specify**. 2

Because the number of options may be large and each option is very wordy you may 2
wish to create a shell script consisting of the configure command and the selected 2
options. 2

6. The next step is to compile PBS by typing 2

```
make
```
2

from the top of the target tree. 2

7. The documentation is not generated by default. You may make it by specifying the 2
`--enable-docs` option to configure or by changing into the *doc* subdirectory in the tar- 2
get tree and typing make. 2

In order to build and print PostScript copies of the documentation from the included 2
source, you will need the GNU groff formating package including the ``ms'' formatting 2
macro package. You may choose to print using different font sets. In the source tree is 2
a file ``doc/doc_fonts'' which may require editing. Please read the comments in that file. 2
Note that font position 4 is left with the symbol font mounted. 2

8. To install PBS you must be running with root privileges. As root, type 2

```
make install
```
2

This generates the working directory structures required for running PBS and installs the programs in the proper executable directories. You must be root to perform this step.

When the directories are made, they are also checked to ensure that they have been setup with the correct ownership and permissions. This is performed to insure 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.

9. The three daemons, pbs_server, pbs_sched and pbs_mom must be run by root in order to function. Typically they should be started at system boot time. To have a fully functional system, each of the daemons will require certain configuration information. This is explained in detail in this guide in **chapter 2. Batch System Configuration**.

   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.

   A. **One time only**, start pbs_server with the "-t create" option. 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. For man pages on pbs_server and pbs_mom, see ERS chapter 6. For man pages on the scheduler, see ERS chapter 7.

   B. Start the execution server, pbs_mom. No options or arguments are required. Mom will look for a configuration file in PBS_HOME/mom_priv/config. If it is not found, Mom will continue to function, but certain requests may be made of Mom from the local system only. See the pbs_mom(8) man page for more information on the config file.

   C. Start the selected job scheduler, pbs_sched.

      a. For C language based schedulers, such as the default, see the man page pbs_sched_cc(8) for more detail.

      b. 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`.

         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:

         ```
         pbs_sched -c config_file
         ```

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

c. 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

```
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
```

From this point, you no longer need root privilege. Note, "your_host" 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 ‘‘qname’’; issue the following commands inside of qmgr:

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

Lastly, allow the 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 be specified on the pbs_server command line with the −a option.

1.5. 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.

1.5.1. 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.

1.5.2. Configure Options

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

1.5.2.1. Generic Configure Options

The following are generic configure options that do not effect PBS.

--cache-file=`file`

Cache the system configuration test results in `file`.

Default: config.cache 2

--help 2
Prints out information on the available options. 2

--no-create 2
Do not create output files. 2

--quiet, --silent 2
Do not print ‘‘checking’’ messages. 2

--version 2
Print the version of autoconf that created configure. 2

--enable-depend-cache 2
This turns on configure’s ability to cache *makedepend* information across runs of config- 2
ure. This can be bad if the user makes certain configuration changes in rerunning con- 2
figure, but it can save time in the hands of experienced developers. 2
Default: disabled 2

1.5.2.2. Directory and File Names 2

These options specify where PBS objects will be placed. 2

--prefix=`PREFIX` 2
Install files in subdirectories of `PREFIX` directory. 2
Default: /usr/local 2

--exec-prefix=`EPREFIX` 2
Install architecture dependent files in subdirectories of `EPREFIX`. 2
Default: see PREFIX 2

--bindir=`DIR` 2
Install user executables (commands) in subdirectory `DIR`. 2
Default: EPREFIX/bin (/usr/local/bin) 2

--sbindir=`DIR` 2
Install System Administrator executables in subdirectory `DIR`. This includes certain 2
administrative commands and the daemons. 2
Default: EPREFIX/sbin (/usr/local/sbin) 2

--libdir=`DIR` 2
Object code libraries are placed in `DIR`. This includes the PBS API library, libpbs.a. 2
Default: PREFIX/lib (/usr/local/lib) 2

--includedir=`DIR` 2
C language header files are installed in `DIR`. 2
Default: PREFIX/include (/usr/local/include) 2

--mandir=`DIR` 2
Install man pages in `DIR`. 2
Default: PREFIX/man (/usr/local/man) 2

--srcdir=`SOURCE_TREE` 2
PBS sources can be found in directory `SOURCE_TREE`. 2
Default: location of *configure* script. 2

1.5.2.3. Features and Package Options 2

In general, these options take the following forms: 2

`--disable-`FEATURE	Do not compile for FEATURE, same as --enable-FEATURE=no	22
`--enable-`FEATURE	Compile for FEATURE	22
`--with-`PACKAGE	Compile to include PACKAGE	22

`--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-isode=`type`
: Use the ISO Data Exchange protocols for backward compatibility.
This will be removed in release 2.0.
If `type` is set to:

  yes, all, or 0
  : then ISODE/ASN.1 protocol is used for commands and server to server communication

  old or 1
  : then ISODE/ASN.1 protocol will be recognized by the server to support old commands.

  no or 2
  : then ISODE/ASN.1 will not be used at all and Data is Strings (DIS) will be used for all communications.

  Default: no - only DIS is used.

--set-isode-top=`DIR`
: Specify the directory under which the ISODE libraries, header files, and executables can be found. They are expected in subdirectories lib, include and bin, respectively.
This will be removed in release 2.0.
Default: search in /usr/local/isode and /usr/local.

--with-tcl=`DIR_PREFIX`
: Use this option if you wish Tcl based PBS features compiled. These 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: 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 local file system. PBS uses synchronous writes to files to maintain state.
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.
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 containing the environment variables used by the daemons and passed to the jobs. Use --without-environ if no environment is required. 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.

--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 should **not** be used for AIX systems.
Default: disabled.

--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 `loadave2.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-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 chapter 6.4 ''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 rather than placing the script file as the shell's standard input. This does result in a second shell being created to process the script. However, a command in the script that reads from standard input will not be able to access the script contents, which can cause some shells to by-pass commands in the script.
Default: enabled

--enable-sp2
Turn on special features for the IBM SP. This requires access to a special IBM supplied library, libSDR.a. This library is not provided by IBM except on special request. 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

--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 enables the use of SGI Array Session tracking. Enabling

this feature is recommanded if MPI jobs use the Array Services Daemon. The PBS 2
machine type is set to irix6array. 2
Default: disabled 2

1.6. 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.

1.6.1. Cray Systems

1.6.1.1. Cray C90, J90, and T90 Systems

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

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`.

1.6.1.2. 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.

1.6.1.3. Cray T3E Systems

For Cray T3E systems, **TBD**.

1.6.2. 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.

1.6.3. IBM SP

Every thing under IBM Workstation section above applies to the IBM SP. Be sure to read the section **3.1 Nodes** before configuring the Server.

Set special SP-2 code to interface with the IBM Resource (Job) Manager with `--enable-sp2`.

1.6.4. 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.

1.6.5. SGI Workstations Running IRIX 6

If built for Irix 6.x, pbs_mom will track which processes are part of a PBS job using POSIX session numbers. This method is fine for workstations and multiprocessor boxes not running the SGI Array Services Daemon (arrayd) and not using SGI's mpirun command. The PBS machine type (MACH) is set to `irix6`.

Where arrayd and mpirun is 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 by setting `--enable-array`. The PBS machine type (MACH) is set to `irix6array`

IRIX 6 supports both 32 and 64 bit objects. In prior versions of PBS, 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 made 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 chapter 3.8 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**.

1.6.6. 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.

1.6.7. Linux

Redhat version 4.x is supported. Version 5.x is not currently supported; there are system header file differences.

1.6.8. SUN Running SunOS

The native SunOS C compiler is not ANSI and cannot be used to build PBS. GNU gcc is rec- 0
ommended. 0

1.7. 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.

1.7.1. 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.

1.7.2. 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 `15000`, 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).

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.

1.8. Site Modifiable Source Files

It is safe to skip this section untill 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, in 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 replace 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, 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.

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.

2. Batch System Configuration

Now that the system has been built and installed, the work has just begun. The scheduling policy must be implemented and the Server and Moms configured. 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 sections **Server Management** and **Scheduling Policies**.

2.1. Nodes

Before proceeding, you will need to make some decisions about how the overall batch system is to be arranged. To do that, we need to define some terms:

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. 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* , *shared* , or *timeshared* .

Cluster

A collection of nodes managed by one batch system. A cluster 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 Node

To confuse the issue, we use the term *cluster node* to identify a node that is allocated specifically to one job (see *exclusive node*), or a few jobs (see *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.

Shared Nodes

A *shared node* is one which is shared by multiple jobs. If several jobs which requested multiple shared nodes are running, some nodes may be common to the subset allocated to each job and some may be unique. When a node is allocated as a shared node, it remains so until all jobs using it are terminated. Then the node may be next allocated for shared or exclusive use.

Timeshare 2

In our context, to always allow multiple jobs to run concurrently on an execution host or 2
node. Often the term host rather than node is used in conjunction with timeshare, as in 2
timeshared host. If the term node is used without the timeshare prefix, the node is one 2
which is allocated either exclusively or shared. 2

If a host, or node, is indicated to be timeshared, it will never be allocated (by the server) 2
exclusively or shared. 2

Load Balance 2

A policy wherein jobs are distributed across multiple timeshared hosts to even out the 2
work load on each host. Being a policy, the distribution of jobs across execution hosts is 2
solely a function of the Job Scheduler. 2

Node Property 2

In order to have a means of grouping nodes for allocation, a set of zero or more node 2
properties may be given to each node. The property is nothing more than a string of 2
printable characters without meaning to PBS. You, as the PBS administrator, may 2
chose whatever property names you wish. Your choices for property names should be 2
relayed to the users. 2

Batch System 2

A PBS Batch System consists of one Job Server (pbs_server), one or more Job Sched- 2
ulers (pbs_sched), and one or more execution servers (pbs_mom). With prior versions of 2
PBS, a Batch System could be set up to support only a cluster of exclusive nodes **or** to 2
support one or more timeshared hosts. There was no support for shared nodes. With 2
this release, a PBS Batch System may be set up to feed work to one large timeshared 2
system, multiple time shared systems, a cluster of nodes to be used exclusively or 2
shared, or any combination of the preceding. 2

2.1.1. Defining Nodes 2

In prior PBS systems, nodes were only specified when the system was built and configured to 2
support exclusive nodes. A single pbs_mom was given a list of nodes. When jobs were run, 2
the top level shell for all jobs ran on the host where that Mom lived. Mom allocated nodes to 2
the job and provided the job a file containing the list of nodes allocated to it. If the PBS batch 2
system was supporting one or more timeshared hosts, only the Job Scheduler knew of those 2
hosts. It directed where the server should send the job for execution. 2

In this version of PBS, allocation of nodes is handled by the Server instead of Mom. Each 2
node to be used by a job must have its own copy of Mom running on it. If only timeshared 2
hosts are to be served by the PBS batch system, then as before, the Job Scheduler must 2
direct where the job should be run. If unspecified, the Server will execute the job on the host 2
where it is running. See the next section for full details. 2

If nodes are to be allocated *exclusively* or *shared*, a list of the nodes must be specified to the 2
server. This list may also contain timeshared nodes. Nodes marked as timeshared will be 2
listed by the server in a node status report along with the other nodes. However, the server 2
will **not attempt to allocate them** to jobs. The presence of timeshared nodes in the list is 2
solely as a convenience to the Job Scheduler and other programs, such as xpbsmon. 2

The node list is given to the server in a file named `nodes` in the server's home directory 2
`PBS_HOME/server_priv`. This is a simple text file with the specification of a single node 2
per line in the file. The format of each line in the file is: 2

```
node_name[:ts] [property ...]
```
2

The node name is the network name of the node (host name). The optional `:ts` appended to 2
the name indicates that the node is a timeshared node. Zero or more properties may be spec- 2
ified. Each item on the line must be separated by white space. Comment lines may be 2
included if the first non-white space character is the pound sign '#'. 2

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 space shared (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
```

Unlike the prior NEEDNODE implementation where pbs_mom managed nodes, the node list used by the server cannot be modified while the server is up. The actual node file is only read by the server at start up. If you need to modify the list, edit it and shut down the server with `qterm -t quick`. All running jobs will continue to run. Then, immediately restart the server. The server will read the modified file and have the new node list. No running jobs will be impacted.

2.1.2. 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.

2.1.2.1. No Node File

If a nodes file does not exist, the server only directly knows about its own host. It will assume 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.

2.1.2.2. 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. Otherwise, a set of nodes to allocate is specified by a server attribute, **default_node**. By default, this is one shared node. It may be set to any valid node request or any single timeshared node listed in the nodes file.

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

- If the batch system consists of a single timeshared host, no problem − all the jobs run there. The scheduler only needs to say which job it wants run.

the various hosts is a matter of the scheduler determining on which host to place 2
the selected job. This is done by querying the resource monitor side of Mom using 2
the resource monitor API - the addreq() and getreq() calls. The Scheduler tells the 2
server where to run each job. 2

- If your cluster is running distributed jobs (as well as serial jobs), the Scheduler typ- 2
ically uses the *Query Resource* or *Avail* request to the server for each queued job 2
under consideration. The Scheduler then selects one of the jobs that the Server 2
replied could run, and directs that the job should be run. The Server will then allo- 2
cate the nodes to the job. By leaving the server attribute **default_node** set to one 2
shared node, `1#shared`, jobs which do not request nodes will be placed together on 2
a few nodes running shared jobs. 2
- If you have a batch system supporting both cluster nodes and one timeshared node, 2
the situation is like the above, only you may wish to change **default_node** to point 2
to the timeshared host. 2
- If you have a batch system supporting both cluster nodes and multiple time shared 2
hosts, you have a complex system which requires a smart scheduler. The Sched- 2
uler must recognize which jobs request nodes and use the *Avail* request to the 2
Server. It must also recognize which jobs are to be load balanced among the time- 2
shared hosts, and provide the host name to the Server when directing that the job 2
be run. The supplied *fifo* scheduler has this capability. 2

2.1.3. Specifying Nodes 2

The *nodes* resource is set by the user to declare the node requirements for the job. It is a 2
string of the form 2

    node_spec[+node_spec...] 2

where node_spec is 2

    number | property[:property...] | number:property[:property...] 2

The node_spec may have an optional global modifier appended. This is of the form `#prop-` 2
`erty`. For example: 2

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

or 2

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

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

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

A major use of the global modifier is to provide the *shared* keyword. This specifies that all 2
the nodes are to be shared nodes. The keyword shared is only recognized as such when used 2
as a global modifier. 2

Two additional Read-Only resources exist for jobs. They are *nodect* and *neednodes*. Nodect 2
(node count) is set by the server to the integer number of nodes desired by the user as 2
declared in the ‘‘nodes’’ resource specification. That declaration is parsed and the resulting 2
total number of nodes is set in nodect. This is useful when an administrator wishes to place 2
an integer limit, *resources_min* or *resources_max*, on the number of nodes used by a job 2
entering a queue. 2

Based on the above example, it would be set to 12 (6+3+2+1). Neednodes is initially set by 2
the Server to the same value as nodes. Neednodes may be modified by the job scheduler for 2
special policies. The contents of neednodes determines which nodes are actually assigned to 2
the job. Neednodes is visible to the administrator but not to the user. 2

If you wish to set up a queue default value for ''nodes'' (a value to which the resource is set if **2**
the user does not supply one), corresponding default values must be set for ''nodect'' and **2**
''neednodes''. For example **2**

    Qmgr: set queue foo resources_default.nodes=1 **2**
    Qmgr: set queue foo resources_default.nodect=1 **2**
    Qmgr: set queue foo resources_default.neednodes=1 **2**

Minimum and maximum limits are set for ''nodect'' only. For example: **2**

    Qmgr: set queue foo resources_min.nodect=1 **2**
    Qmgr: set queue foo resources_max.nodect=15 **2**

Minimum and maximum values must **not** be set for nodes or neednodes as those are string **2**
values.

2.2. Network Addresses

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. 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** later in this section.

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   15000/tcp 9
    pbs_dis   15001/tcp 9
    pbs_mom   15002/tcp 15003/udp 15003/tcp 9
    pbs_sched   15004/tcp 9

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.

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

As of release 1.1.9, there is one additional port used. *pbs_dis* is used for requests of the 2
server in the new *Data Is Strings* data encoding. When the ASN.1 encoding is completely 2
removed, in release 2.0, port 15000 will no longer be needed. As of release 1.1.10, port 0
15003/tcp is used by pbs_mom for task management (and alternately for scheduler communi- 0
cation) as well as 15003/udp.

2.3. 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.

2.3.1. Starting the Server

The initial run of the Server or any first time run after recreating the home directory should be with the `-t create` option. This option directs the Server to create a new server database. If one 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 3.4.1 **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. 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.

2.3.2. Starting the Scheduler

The Scheduler should also be started at boot time. Typically the only required option for the BaSL based scheduler is the `-r script_file` option specifying the script file. For the Tcl based scheduler, the option `-b script_file` is used to specify the Tcl script to be called.

2.3.3. Starting MOM

MOM should also be started at boot time. Typically there are no options unless MOM is being restarted on a running system or the pbs_sever or pbs_sched are running on a different host.

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 **2.5 3Configurating 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".

2.4. 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 config- 2
ured while it is running, except for the nodes file. Configuring server and queue attributes 9
and creating queues is done with the qmgr(1B) command. This must be either as root or as a 9
User who has been granted PBS Manager privilege as shown in the last step in the **Build** 9
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. 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**.

2.4.1. 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.

2.4.1.1. 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:

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

2.4.1.2. 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:

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

`acl_host_enable`

Enables the server host access control list, see above.

```
s s acl_host_enable=true
```

`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 sub-domain and ''sam'' only from this system, big, then:

```
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.

```
s s resources_defaults.cput=5:00
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.

2.4.2. Queue Configuration

Each server must have at least one queue defined. It may be either a routing queue or an 9
execution queue. Typically it will be an execution queue; jobs cannot be executed while resid- 9
ing in an routing queue. 9

Queue attributes fall into three groups, those which are applicable to both types of queues, 9
those applicable only to execution queues and those applicable only to routing queues. If an 9
''execution queue only'' attribute is set for a routing queue, or vice versa, it is simply ignored 9
by the system. However, as this situation might indicate the administrator made a mistake, 9
the Server will issue a warning message about the conflict. The same message will be issued 9

if the queue type is changed and there are attributes that do not apply to the new type. 9

Not all of the Queue Attributes are discussed here, only what is needed to get a reasonable 9
system up and running. See the *pbs_queue_attributes* man page for a complete list.

2.4.2.1. 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.

```
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.

```
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.

```
s q dque started=true
```

2.4.2.2. Required Attributes for Routing Queues

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

```
s q routem route_destination=dque,overthere@another.foo.bar.com
```

2.4.2.3. 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 resource_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.

2.4.2.4. 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.

2.4.2.5. 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 *resource_min* and *resource_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:

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

When a job is being enqueued, it's requested resource list is tested against the queue limits: `resource_min <= job_requirement <= resource_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. That will place the job into the appropriate queue. For example:

```
create queue feed queue_type=routing
set queue feed route_destination="short,long"
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:

```
set queue short resource_max.cput=59
set queue long  resource_min.cput=1:00,resource_max.cput=1:00:00
create queue verylong queue_type=execution
set queue feed route_destination="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.

One word of caution is required. If a job does not specify a resource list (requirement) for the tested resource. No test is made. 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.

```
set queue short resource_default.cput=40
```

will see that a job without a cpu time specification is limited to 40 seconds if it enters the short 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 `resource_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. 0

Minimum and maximum queue limits work with numerical valued resources, including time 0
and size values. Generally, they do not work with string valued resources because of charac- 0
ter comparison order. However, setting the min and max to the same value to force an exact 0
match will work even for string valued resources. For example, 0

```
set queue big resource_max.arch=unicos8
set queue big resource_min.arch=unicos8
```
0
0

can be used to limit jobs entering queue **big** to those specifying `arch=unicos8`. Again, 0
remember that if `arch` is not specified by the job, no test is performed and it will be accepted.

2.4.3. Recording Server Configuration

Should you wish to record the configuration of a server for re-use, you may used 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
```

2.5. Configuring the Execution Server, pbs_mom 2

Mom is configured via a configuration file which she reads at initialization time and when 2
sent the SIGHUP signal. This file is described in the pbs_mom(8) man page as well as in the 2
following section. 2

If the -c option is not specified when mom is run, she will open *PBS_HOME/mom_priv/con- 2
fig* if it exists. If it does not, mom will continue anyway. This file may be placed else where 2
or given a different name. In which case, pbs_mom must be started with the -c option. 2

The file provides several types of run time information to pbs_mom: static resource names 2
and values, external resources provided by a program to be run on request via a shell escape, 2
and values to pass to internal set up functions at initialization (and re-initialization). 2

Each item type is on a single line with the component parts separated by white space. If the 2
line starts with a hash mark (pound sign, #), the line is considered to be a comment and is 2
skipped. 2

2.5.1. "Access Control and Initialization Values" 2

An initialization value directive has a name which starts with a dollar sign ($) and must be 2
known to MOM via an internal table. Currently the entries in this table are: 2

clienthost 2

A *$clienthost* entry causes a host name to be added to the list of hosts which will be 2
allowed to connect to MOM as long as they are using a privileged port. For example, 2
here are two configuration file lines which will allow the hosts "fred" and "wilma" to 2
connect: 2

```
$clienthost     fred
$clienthost     wilma
```
2
2

Two host names are always allowed to connect to pbs_mom, "localhost" and the name 2
returned to pbs_mom by the system call gethostname(). These names need not be spec- 2
ified in the configuration file. The hosts listed as "clienthosts" comprise a "sisterhood" 2
of hosts. Any one of the sisterhood will accept connections from a scheduler [Resource 2
Monitor (RM) requests] or server [jobs to execute] from within the sisterhood. They will 2
also accept Internal MOM (IM) messages from within the sisterhood. For a sisterhood 2
to be able to communicate IM messages to each other, they must all share the same RM 2
port. 2

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 chapter 6.3 ''Use and Maintenace of Logs''
of 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

$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.

2.5.2. "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
```
2
2
2

The names can be anything and are not restricted to actual hardware. For example the 2
entry `pong 1` could be used to indicate to the scheduler that a certain piece of software is 2
available on this system. 2

2.5.3. "Shell Commands" 2

If the first character of the value portion of a name/value pair is the exclamation mark (!), the 2
entire rest of the line is saved to be executed through the services of the **system**(3) standard 2
library routine. The first line of output from the shell command is returned as the response 2
to the resource query. 2

The shell escape provides a means for the resource monitor to yield arbitrary information to 2
the scheduler. Parameter substitution is done such that the value of any qualifier sent with 2
the resource query, as explained below, replaces a token with a percent sign (%) followed by 2
the name of the qualifier. For example, here is a configuration file line which gives a 2
resource name of "escape": 2

```
escape     !echo %xxx %yyy
```
2

If a query for "escape" is sent with no qualifiers, the command executed would be "echo %xxx 2
%yyy". If one qualifier is sent, "escape[xxx=hi there]", the command executed would be "echo 2
hi there %yyy". If two qualifiers are sent, "escape[xxx=hi][yyy=there]", the command exe- 2
cuted would be "echo hi there". If a qualifier is sent with no matching token in the command 2
line, "escape[zzz=snafu]", an error is reported. 2

A possible use of the shell command configuration entry is to provide a means by which the 2
use of floating software licenses may be tracked. If a program can be written to query the 2
license server, the number of available licenses could be returned to tell the scheduler if it is 2
possible to run a job that needs a certain licensed package. [You get the fun and games of 2
writing this program.] 2

2.6. Configurating the Scheduler, pbs_sched 2

The configuration required for a scheduler depends on the scheduler itself. If you are start- 2
ing with the delivered *fifo* scheduler, please jump ahead to section 4.5.1 ‘‘FIFO Scheduler’’ in 2
this guide. 2

2.7. Alternate Test Systems 2

Alternate or test copies of the various daemons may be run through the use of the command 2
line options which set their home directory and service port. For example, the following com- 2
mands would start the three daemons with a home directory of `/tmp/altpbs` and four ports 2
around 13000. 2

```
pbs_server -t create -d /tmp/altpbs -p 13000 -M 13002  -S 13004
pbs_mom -d /tmp/altpbs -M 13002 -R 13003 -T 13005
pbs_sched -d /tmp/altpbs -S 13004 -r script_file
```
2
2
2

The home directories must be pre-built. The easiest method is to alter the 2
PBS_SERVER_HOME variable by use of the `--set-server-home` option to configure, 2
rerun configure and remake PBS. 2

Jobs may be directed to the test system by using the `server:port` syntax on the -q option. 2
Status is also obtained using the `:port` syntax: For example, to submit a job to the default 2
queue on the above test server, request the status of the test server, and request the status of 2
jobs at the test server: 2

```
qsub -q @host:13000 job
```
2

```
qstat -Bf host:13000
qstat @host:13000
```

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.

2.8. 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_SERVER_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_SERVER_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_SERVER_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_SERVER_HOME will have changed and have been made during testing. If not, build a (temporary) server directory tree by changing PBS_SERVER_HOME using --set-server-home and typing `buildutils/pbs_mkdirs server` 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 13000 -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:13000 < /tmp/config
   qenable queue1@host:13000
   qenable queue2@host:13000
   ```
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:13000 job
   ```

```
qstat @host:13000
```

If all is going well, move the remaining jobs a queue at a time:

```
qmove queue1@host:13000 `qselect -qqueue1`
qstat queue1@host:13000
qmove queue2@host:13000 `qselect -qqueue2`
qstat queue2@host:13000
```

7. At this point, all of the jobs should be under control of the new server and located in the new server home. If the new server 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 (15000), you may attempt, *unsuccessfully*, to use the following command:

```
qmove queue@host 123.job.host:13000
```

However, that will only move the job to the end of the queue it is already in. The server receiving the move request (13000), 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:15000 123.job.host:13000
```

3. 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 APIs to accomplish your heart's desires.

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.

3.1. 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. The server opens a connection to the scheduler and sends a command indicating the reason for the scheduling cycle. The reason or event that triggers 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.

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.

3.2. 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.

3.3. Tcl Based Scheduling

The provided Tcl based Scheduler 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 it exists) 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.

3.3.1. 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.

3.4. 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)* to initialize whatever is required to be initialized. This is called one time only.
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 sample.scheds subdirectory.

3.5. C Based Sample Schedulers

The following sections discuss simple C based sample schedulers supplied with PBS. The sources for the samples are found in *src/scheduler.cc/samples* under the scheduler type name, for example *src/scheduler.cc/samples/fifo* .

3.5.1. 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 to:

- All jobs in a queue will considered for execution before the next queue is examined. (See file `PBS_HOME/sched_priv/sched_config`)
- The queues are sorted by queue priority. (See file `PBS_HOME/sched_priv/sched_config`)
- The jobs within each queue are sorted by requested cpu time (cput). The shortest job is places first. (See file `PBS_HOME/sched_priv/sched_config`)
- 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` & `PBS_HOME/sched_priv/sched_config`)
- 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.

- The system resources which are checked to make sure they are not exceeded: *mem* (memory requested) and *ncpus* (number of CPUs requested).

3.5.1.1. 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 chapter 3.4, 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 chapter 3.1.
{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:
filter PBSEVENT_DEBUG2 PBSEVENT_DEBUGnPBSEVENT_ADMIN
0x100: 256 0x080: 128 0x004: 4= 388

PBSEVENT_JOBPBSEVENT_DEBUGPBSEVENT_SCHED
0x008: 8 0x080: 128 0x040: 64= 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 will need to be created. Use the following format:

```
# 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 2
numeric: The amount of shares(priority) the user/group has in the resource group. 2

6. If strict priority is wanted, a fair share tree will be needed. A really simple one will suf- 2
fice. Every user's resgrp will be root. The amount of shares will be their priority. Next, 2
set `unknown_shares` to one. Everyone who is not in the tree will share the one share 2
between them to make sure everyone in the tree will have priority over them. Lastly, 2
the main sort must be set to *fair_share*. This will sort by the fair share tree which was 2
just set up. 2

7. Create the holidays file to handle prime time and holidays. The holidays file should use 2
the UNICOS 8 holiday format. The ordering does matter. Any line that begins with a 2
"*" is considered a comment. 2

YEAR YYYY 2
This is the current year. 2

<day> <prime> <nonprime> 2
Day can be weekday | saturday | sunday 2
prime and nonprime are times when prime or non-prime time start. They can ei- 2
ther be HHMM with no colons(:) or the word "all" or "none" 2

<day> <date> <holiday> 2
day is the day of the year between 1 and 365 date is the calendar date. Ex Jan 1 2
holiday is the name of the holiday. Ex New Year's Day This is repeated for each 2
company holiday 2

8. To load balance between timesharing nodes, several things need to happen. First, 2
a nodes file needs to be set up in the PBSHOME/server_priv (See section 3.1). All 2
timesharing nodes need to be denoted with :ts. These are the nodes which the 2
scheduler will load balance between. Secondly, on every node there has to be a 2
mom. In each of mom's config files two static values need to be set up. One for 2
ideal load and the other is maximum load. This is done by putting two lines in the 2
config file like the following format: name value. The name will be *ideal_load* and 2
max_load. Lastly, turn the load_balancing bit on in the scheduling policy config 2
file. Load balancing will have the job comment changed on running of the job to 2
show where the job was run. 2

Example of mom config file:(64 processor machine) 2
ideal_load 60 2
max_load 64 2

9. Space sharing is done automatically if there are both a nodes file and the job re- 2
quests nodes. Make sure to set up a resources_default.nodes and resources_de- 2
fault.nodect. 2

10. The scheduler honors the following attributes/node resources: 2

Queue	:	started	22
Queue	:	queue_type	22
Queue	:	max_running	22
Queue	:	max_user_run	22
Queue	:	max_group_run	22
Job	:	job state	22
Server	:	max_running	22

Server	:	max_user_run	22
Server	:	max_group_run	22
Server	:	resources_available	22
Server	:	resources_max	22
Node	:	loadave	22
Node	:	arch	22
Node	:	ncpus	22
Node	:	physmem	22

NOTE: if resources_available.res is set, it will be used, if not resources_max.res 2
will be used. If neither are set infinity is assumed. 2

3.5.1.2. Examples FIFO Configuration Files 2

The following are just examples and may or may not be what is shipped. 2

<u>Example of a scheduling config file</u> 2

```
#    Set the boolean values which define how the scheduling policy finds    2
#    the next job to consider to run.    2
round_robin: False   ALL    2
by_queue: True       prime    2
by_queue: false      non-prime    2
strict_fifo: true    ALL    2
fair_share: True prime    2
fair_share: false    non-prime    2

# help jobs which have been waiting too long    2
help_starving_jobs: true prime    2
help_starving_jobs: false    non-prime    2

# Set a multi_sort    2
# This example will sort jobs first by ascending cpu time requested, and then    2
# by ascending memory requested, and then finally by descending job priority    2
#    2
sort_by: multi_sort    2
key: shortest_job_first    2
key: smallest_memory_first    2
key: high_priority_first    2

# Set the debug level to only show high level messages.    2
# Currently this only shows jobs being run    2
debug_level: high_mess    2

# a job is considered starving if it has waited for this long    2
max_starve:  24:00:00    2

# If the scheduler comes by a user which is not currently in the resource group    2
# tree, they get added to the "unknown" group.  The "unknown" group is in roots    2
# resource group.  This says how many shares it gets.    2
unknown_shares: 10    2

# The usage information needs to be written to disk in case the scheduler    2
# goes down for any reason.  This is the amount of time between when the    2
```

```
# usage information in memory is written to disk.  The example syncs the    2
# information ever hour.    2
sync_time: 1:00:00    2

# What events do you not want to log.  The event numbers are defined in    2
# src/include/log.h.  NOTE: the numbers are in hex, and log_filter is in    2
# base 10.    2
# The example is not to log DEBUG2 events, which are the most prolific    2
log_filter: 256    2
```

Here is an example of the holidays file 2

```
* the current year    2
YEAR    1998    2

*    2
* Start and end of prime time    2
*    2
*          Prime   Non-Prime    2
* Day      Start   Start    2

weekday      0400   1130    22
saturday     none   all    22
sunday       none   all    22

*    2
* The holidays    2
*    2
* Day of     Calendar     Company    2
* Year       Date         Holiday    2
*    2

1         Jan 1         New Year's Day    22
20        Jan 20        Martin Luther King Day    22
48        Feb 17        President's Day    22
146       May 26        Memorial Day    22
185       Jul 4         Independence Day    22
244       Sep 1         Labor Day    22
286       Oct 13        Columbus Day    22
315       Nov 11        Veteran's Day    22
331       Nov 27        Thanksgiving    22
359       Dec 25        Christmas Day    22
```

Example of the resource group file for fair share 2

```
#    2
# the groups "root" and "unknown" are added by the scheduler    2
# All the parents must be added for the children.  This is why all the groups    2
# are added first.  The cresgrp numbers the users have are their UIDs    2
#    2

# name        resgrp        child resgrp     shares    2
```

```
grp1      50    root    10        22
grp2      51    root    20        22
grp3      52    root    10        22
grp4      53    grp1    20        22
grp5      54    grp1    10        22
grp6      55    grp2    20        22
usr1      60    root    5         22
usr2      61    grp1    10        22
usr3      62    grp2    10        22
usr4      63    grp6    10        22
usr5      64    grp6    10        22
usr6      65    grp6    20        22
usr7      66    grp3    10        22
usr8      67    grp4    10        22
usr9      68    grp4    10        22
usr10     69    grp5    10        22
```

Example of strict priority resource group file 2

 # this is a strict priority resource group file. These are people who should 2
 # get priority over everyone else. The amount of shares is the priority of 2
 # the user. 2

```
sally       1000     root     4          22
larry       1001     root     6          22
manager     1010     root     100        22
vp          1016     root     500        22
ceo         2000     root     10000      22
```

Example of dedicated file 2

 # Format: 2
 #   FROM   TO 2
 # MM/DD/YYYY HH:MM   MM/DD/YYYY HH:MM 2

```
04/10/1998    15:30    04/11/1998    23:50      22
05/15/1998    05:15    05/15/1998    08:30      22
06/10/1998    23:25    06/10/1998    23:50      22
```

3.6. Scheduling and File Staging 2

A decision must be made about when to begin to stage in files for a job. The files must be 9
available before the job executes. The amount of time that will be required to copy the files is 9
unknown to PBS, that being a function of file size and network speed. If file in-staging is not 9
started until the job has been selected to run when the other required resources are avail- 9
able, either those resources are ‘‘wasted’’ while the stage in occurs, or another job is started 9
which takes the resources away from the first job, and might prevent it from running. If the 9
files are staged in well before the job is otherwise ready to run, the files may take up valuable 9
disk space need by running jobs. 9

PBS provides two ways that file in-staging can be initiated for a job. If a run request is re- 9
ceived for a job with a requirement for staging-in files, the staging in operation is begun and 9
when completed, the job is run. Or, a specific stage-in request may be received for a job, see 9

pbs_stagein(3B), in which case the files are staged in but the job is not run. When the job is 9
run, it begins execution immediately because the files are already there. 9

In either case, if the files could not be staged-in for any reason, the job is placed into a wait 9
state with a ‘‘execute at’’ time 30 minutes, in the future. A mail message is sent to the job 9
owner requesting that s/he look into the problem. The reason the job is changed into wait 9
state is to prevent the scheduler from constantly retrying the same job which likely would 9
keep on failing. 9

Figure 5.0 in appendix B of the ERS shows the (sub)state changes for a job involving file in 9
staging. The scheduler may note the substate of the job and chose to perform pre-staging via 9
the pbs_stagein() call. The substate will also indicate completeness or failure of the opera- 9
tion. The scheduler developer should carefully chose a stage in approach based on factors 9
such as the likely source of the files, network speed, and disk capacity.

4. GUI System Administrator Notes

Currently, PBS provides two GUIs: xpbs and xpbsmon.

4.1. xpbs

xpbs provides a user-friendly point-and-click interface to the PBS commands.
The system administrator can specify a global resources file, *${PBS_LIB}/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.

4.2. 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 chapter 2.5 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*.

5. Operational Issues

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

5.1. 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.

5.1.1. 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 file 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 give 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 attach on a system. To prevent this type of attach, each daemon resets its own environment when it starts. The source of the environment is a file named by PBS_ENVIRON set by the configure option --set-environ. If it does not already exists, this file is created during the install process. It should be edited to include the minimum set of variables required on your system. Please note that one variable, **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 the PBS_ENVIRON file 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.

5.1.2. Host Authentication

PBS uses a combination of a 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.

5.1.3. 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.

5.1.4. 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.

5.1.5. 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.

5.1.6. 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 0
groups within a system. Each and every group in which a user might execute a job MUST 0
have a group name and an entry in /etc/group. If no group lists are ever used, PBS will use 0
the login group and will accept it even if the group is not listed in /etc/group. Note in this 0
case, the **egroup** attribute value is a numeric string representing the user's gid rather than 0
the group ''name''.

5.1.7. 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 this or some other system, is listed in the server attribute **acl_roots**.

5.2. 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.

If a 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.

5.2.1. 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. †

† The PBS development group reserves the right in a future release to remove certain of the arguments passed to the Epilogue which could be obtained with in the script by calling qstat(1B) or pbs_statjob(3B).

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. For all systems execute the SP, this is /dev/null. For the SP, the file is the host file containing the list of nodes assigned to the job.

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.

5.2.2. 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 within src/resmom/prolog.c.

5.2.3. 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.

5.3. 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.

5.3.1. The Daemon Logs

Each PBS daemon maintains an event log file. 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 0
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) 0

Everything turned on is of course 511. 127 is a good value to use. The event logging mask is 0
controlled differently for the server and MOM. The server's mask is set via qmgr(1B) setting 0
the **log_events** attribute. This can be done at any time. MOM's mask may be set via her 0
configuration file with a `$logevent` entry, see the -c option on pbs_mom. To change her log- 0
ging mask, edit the configuration file and send MOM a SIGHUP signal. 0

The scheduler, being site written may have a different method of changing its event logging 0
mask, or it may not have the ability at all.

5.3.2. The Accounting Log

The PBS Server daemon maintains an accounting log. The format of the log is described in chapter 3 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.

If the default file is used, named for the date, the file will be closed and a new one opened every day at midnight. If the server receives a SIGHUP signal, it will close the accounting log and reopen it. This allows you to rename the old log and start recording anew on an empty file. For example, if the server was started on February 1, it will be writing in the file `19960201`. If the current date is February 9, then the following actions will cause the current accounting file to be renamed `feb1`, the server to close 032 and starting writing `19960209`.

```
mv 19960201 feb1
kill -HUP 1234     (the server's pid)
```

5.4. Problem Solving

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

5.4.1. 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 15000
```

Where `server_host` is the name of the host on which the server is running and `15000` is the port to which the server is listening (if built with a different port number, use that number instead of 15000). 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.

5.4.2. 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.

5.4.3. Job Cannot be Executed

If a user receives a mail message containing a job id and the line ‘‘Job cannot be executed’’, the job was aborted by MOM when she tried to place it into execution. The complete reason can be found in one of two places, MOM’s log file or the standard error file of the user’s job.

If the second line of the message is ‘‘See Administrator for help’’, then MOM aborted the job before the job’s files were set up. The reason will be noted in MOM’s log. Typical reasons are a bad user/group account, checkpoint/restart file (Cray), or 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.

5.4.4. 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 0
has no active processes. This should never happen as the death of the job’s shell should trig- 0
ger Mom to notify the server that the job exited and end of job processing should begin. The 0
fact that it happens even rarely means there is a bug in PBS (*gasp! Oh the horror of it all.*). 0

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

5.5. 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
```

6. Advice for Users

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

6.1. 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
```

6.2. 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.

Script as Standard Input

The standard method for PBS (built with SHELL_INVOKE = 0), is to open the script file as standard input for the shell. This is equivalent to typing `shell < script`. This 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.

Name of Script on Standard Input

The alternative method (PBS built with SHELL_INVOKE = 1), 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 also 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 a non-standard shell is used via the -S option, it will not receive the script, but its name, on its standard input.

The choice if 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.

6.3. 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
```

6.4. Delivery of Output Files

PBS uses a version of the **rcp**(1) command, **pbs_rcp**(1B) to return output files to the user at a remote host after the user's job terminates. This version of rcp is based on the bsd 4.4 lite distribution rcp. It is provided because it, unlike some rcp implementation, always exits with a non-zero exits status for any error. Thus pbs_mom knows if the file was not delivered. Note that the secure copy program, scp, is also based on this version of rcp and it is good about exit statuses.

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 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* .

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.

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 files, 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.

6.5. 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 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 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 is directory on a stage out directive, the local_file 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 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.

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.