

# Using PBS

The Portable Batch System is a batch queuing system that provide controls for initiating and scheduling the execution of batch jobs. It operates on networked, multi-platform UNIX systems. A PBS batch job can run a serial program on one node, a script on multiple nodes, multiple programs on multiple nodes, and so on. In addition, it can schedule MPI jobs on a shared computing resource such as a cluster.

PBS uses a client/server model. The **pbs\_server** server process manages high-level batch objects such as queues and jobs. Each compute node also runs a server process named **pbs\_mom**. This process communicates with the **pbs\_server** daemon. The **pbs\_mom** process also launches the job to which the **pbs\_server** hands off control.

PBS assumes that a user submits a job in the form of a script to the **pbs\_server**, which schedules the job for later execution by a **pbs\_mom** process. When submitting a job, the user defines job parameters such as the number of nodes, maximum execution time, and the like either within the script file or by using command-line options.

The PBS interactive mode allows users to obtain a shell within the allocated resources. The interactive session ends when the job times out or the user exits the shell. The client program for submitting a PBS job is called **qsub**. The **xpbs** GUI program shows the job queues, scheduled jobs. It also allows the user to submit new jobs.

## What PBS is not

PBS does not fully define a computing environment. Its scope is more limited, as follows:

- It manages resources (nodes or processors).
- Allocates resources for jobs in its queues according to the scheduling policies defined by the administrator.

Because PBS does not have a security model, the PBS environment may or may not become work well with TotalView.

PBS is not coupled to an MPI implementation. Consequently, the system administrator must make sure that PBS and the MPI implementation work together.

Because PBS does not fully define the policies of a system, there cannot be just one method for using TotalView within a PBS-managed cluster environment. However, Etnus can provide suggestions.

## OpenPBS 2.3.16 and MPICH 1.2.5: An Example

### Installing and Integrating PBS and MPICH

The PBS web site recommends making a small script change to better integrate MPICH into the PBS environment. This change lets you submit an MPI job to PBS without having to first know which hosts are granted to the job. When a job is run by **pbs\_mom**, the job inherits a set of PBS environment variables. The **PBS\_NODEFILE** variable names a file on the file system where the job is running. This file lists the hosts that have been granted to the job by the **pbs\_server**. Consequently, you will need to change the MPICH script so that it automatically uses this file for the **-machinefile** command-line option used by **mpirun** parameter. Here is the patch:

```
*** ../../i386-linux.orig/bin/mpirun.args  \
      Thu Apr 22
--- mpirun.args Thu Apr 22 16:08:11 2004
*****
*** 26,31 ****
--- 26,35 ----
      #p4ssport=
      just_testing=0
      machineFile=
+ if [ "$PBS_NODEFILE" != "" ]
+ then
+     machineFile=$PBS_NODEFILE
+ fi
      debugger=""
      rshcmd="$RSHCOMMAND"
      mvhome=0
```

If you do not apply this patch, you need to name the variable as part of your **mpirun** command. For example:

```
mpirun -machinefile ${PBS_NODEFILE} ...
```

### Using PBS and MPICH (Interactive mode)

The PBS client program is called **qsub**. This program submits a batch or interactive job to **pbs\_servr**. For example:

```
qsub -I -l nodes=num_of_nodes
```

## Using PBS and MPICH (Batch mode)

This command returns a shell on one of the nodes in the job allocation. If you have installed the MPICH PBS patch, you can run an MPICH MPI job in an interactive shell as follows:

```
mpirun -np num_of_processes program_name
```

PBS is most typically used in batch mode. Although `qsub` is still the client interface, you must define your job in a batch script. For example:

```
#!/bin/sh

# Define qsub command options here. Each line
# processed by PBS begins with "#PBS"
#
#PBS -l nodes=<# of nodes>
#PBS -N MyJobName
#PBS -o /some/output/file/directory

cd /some/directory
mpirun -np <# of processes> <program name>
```

You can add many additional command-line options to define and configure your job. After you create the script, use the `qsub` command to place a job into the job queue. For example:

```
qsub myjob.qsub
```

## Using TotalView with OpenPBS and MPICH

There are several cases that need to be considered when discussing using TotalView within the PBS/MPICH framework. The most important of which are as follows:

- Using TotalView in an interactive session
- Using TotalView from a batch session
  - Submitting a TotalView job
  - Submitting an `xterm` job
- Running TotalView from a master node when there are no X libraries on the compute nodes)
- Notes
  - Using SSH
  - `.cshrc`
  - `stdin`, `stdout`, `stderr`



*The examples and explanations in this document assume that you have applied the MPICH patch for `PBS_NODEFILE`.*

## Using TotalView Interactively

In some cases, TotalView may not correctly obtain the contents of the `DISPLAY` and `PATH` environment variables. In most cases, TotalView interoperates cleanly with a PBS MPICH job. For example:

```
[blake@beatles0 ~]$ qsub -I -l nodes=2
```

```
qsub: waiting for job 44.beatles0 to start
qsub: job 44.beatles0 ready
```

```
[blake@beatles2 ~]$ setenv DISPLAY foo:0.0
[blake@beatles2 ~]$ setenv TOTALVIEW /path/to/totalview
[blake@beatles2 ~]$ mpirun -tv -np 2 programname
```

Because you get a new shell for the interactive session, you should automatically export the current environment variables to the new interactive shell using the `-V` option or `-v DISPLAY,TOTALVIEW,PATH` options.



*Depending on the status of the system, number of scheduled jobs, and so on, the time between when you enter the `qsub` command and when the system displays the interactive shell can be long. There may also be special rules or limitations imposed by the PBS administrator upon interactive jobs.*

## Debugging a job using a batch session

There are two approaches to debugging a job in a batch session.

- Submit a `qsub` script which executes a `mpirun -tv ...` command.
- Submit `xterm` as a batch job for a quasi-interactive session.

### Using an mpirun Batch Script

Using an `mpirun` batch script is the easiest way to run TotalView in an PBS/MPICH environment. The major sticking point to make sure the environment is set up properly before you begin. For example, environment variables like `DISPLAY` need to be set prior to executing the `mpirun -tv ...` command. After submitting the command, you will wait until the job is submitted. Only after execution begins will the system display a TotalView window.

Here is an sample batch script that export the needed environment variables.

```
#!/bin/sh

# Define qsub command options here. Each line
# to be processed by PBS should begin as "#PBS"
#
#PBS -l nodes=2
#PBS -N TotalViewJob
#PBS -v DISPLAY,PATH

export TOTALVIEW=/usr/toolworks/totalview/bin/totalview
export TVDSVRLAUNCHCMD=rsh

cd ${HOME}/src/hello-mpi
mpirun -tv -np 2 hello_beatles
```

The disadvantages of this approach are:

- `stdout` and `stderr` messages from the program are not in a controlling console window. You will need to look at the log file for this information. This log file is contained within the Root Window.

- Figuring out why a job doesn't launch correctly from `mpirun` is time-consuming since you will need to resubmit the job (perhaps more than once) so that you can look at the log files generated by the job.
- X Windows must be installed on the compute nodes.

The advantages of the approach are:

- It's easy.
- You only have to figure it out once.
- TotalView is run within the node allocation.

### Using an xterm Batch Script

Etnus recommends using an `xterm` batch script for many batch queuing systems, including PBS. This approach lets you get an interactive debugging session from a job submitted to a batch queue. When the `xterm` appears, you are ready to run TotalView in a manner similar to the way you would interactively run TotalView.

Before submitting this kind of job, you must be careful in the way that you handle environment variables since the `xterm` will initialize a new shell that may not contain all of the environment variables you will need for running the job. You must make sure that `PBS_NODEFILE` gets exported to the `xterm`. For example:

```
#!/bin/sh

# Define qsub command options here. Each line
# to be processed by PBS should begin as "#PBS"
#
#PBS -l nodes=2
#PBS -N TotalViewJob
#PBS -v DISPLAY,PATH

export TOTALVIEW=/usr/toolworks/totalview/bin/totalview
export TVDSVRLAUNCHCMD=rsh

cd ${PBS_0_WORKDIR}
env PBS_0_HOME=${PBS_0_HOME} \
  PBS_0_LANG=${PBS_0_LANG} \
  PBS_0_LOGNAME=${PBS_0_LOGNAME} \
  PBS_0_PATH=${PBS_0_PATH} \
  PBS_0_MAIL=${PBS_0_MAIL} \
  PBS_0_SHELL=${PBS_0_SHELL} \
  PBS_0_HOST=${PBS_0_HOST} \
  PBS_0_WORKDIR=${PBS_0_WORKDIR} \
  PBS_0_QUEUE=${PBS_0_QUEUE} \
  PBS_JOBNAME=${PBS_JOBNAME} \
  PBS_JOBID=${PBS_JOBID} \
  PBS_QUEUE=${PBS_QUEUE} \
  PBS_JOBCOOKIE=${PBS_JOBCOOKIE} \
  PBS_NODENUM=${PBS_NODENUM} \
  PBS_TASKNUM=${PBS_TASKNUM} \
  PBS_MOMPORT=${PBS_MOMPORT} \
  PBS_NODEFILE=${PBS_NODEFILE} \
```

```
PBS_ENVIRONMENT=${PBS_ENVIRONMENT} \
DISPLAY=${DISPLAY} \
PATH=${PATH} \
TOTALVIEW=${TOTALVIEW} \
TVDSVRLAUNCHCMD=${ TVDSVRLAUNCHCMD} \
xterm -title "${PBS_JOBNAME}"
```

After the `xterm` that appears, you can type a standard `mpirun` command to invoke your program in TotalView. For example:

```
mpirun -tv -np 2 ./hello_beatles
```

The advantages of the `xterm` batch approach are:

- TotalView runs in a controlling console.
- You can see and control `stdin`, `stdout`, and `stderr`.
- TotalView runs on a node within the job allocation.

The disadvantages of this approach are:

- You have to be careful passing environment variables to the shell that is launched by `xterm`. For example, it is susceptible to the standard kinds of `.cshrc` issues.
- X windows and `xterm` must exist on the compute nodes.
- Some systems do not allow you to interactively access batch nodes.

## Special Cases

### Running TotalView From a Master Node

Because each system creates its own policies for managing a PBS cluster, you may need to run TotalView and possibly the `mpirun` command from the head node. (The head node is not considered part of the job.) You may need to do this if compute nodes do not have a X libraries installed.

You can run from the head node if you use the `mpirun's` `-nolocal` command-line option. This option tells MPICH that it should not include the node on which you are running the `mpirun` command in the MPI job.

The best solution is to use the TotalView `-remote` command-line option to start the job remotely. This does, however, count as one of the CPUs in your allocated job.

As an alternative, you can patch the `mpirun_dbg.totalview` option if the number of CPUs is an issue. If you use the `-nolocal` case, this option tells the script to `rsh` into the rank 0 node and run the TotalView command from there. That is somewhat fragile. Here is the patch:

**If the number of CPUs is an issue for MPICH are below.**

```
*** i386-linux.orig/bin/mpirun_dbg.totalview Thu Apr 2004
--- i386-linux/bin/mpirun_dbg.totalview Fri Apr 23 2004
*****
*** 28,52 ****
        ;;
    esac
done
#
if [ "$just_testing" = 1 ] ; then
    doitall="echo"
else
```

```

        # The eval must handle arguments containing blanks
        doitall="eval"
    fi
! # Note that this is run from within the mpirun.ch_p4
! # script only (!), so the argument list is p4
! # specific. FIX ME!
! #
! # mpirun.ch_p4 exports nlocal and machinehead if not
! # a local run.
! if [ "$nlocal" = 1 ] ; then
!     # Note that mpirun.ch_p4 exports P4_RSHCOMMAND
!     rshcmd=${P4_RSHCOMMAND-ssh}
!     if [ "$rshcmd" = "ssh" ] ; then
!         doitall="$doinall $rshcmd -X $machinehead"
!     else
!         doitall="$doinall $rshcmd env DISPLAY=$DISPLAY"
!     fi
! fi
- $doinall $tvcommand $progrnamemain -a $cmdLineArgs \
    -p4pg $p4pgfile -p4wd $p4workdir -mpichtv

--- 28,48 ----
        ;;
    esac
done
+
+ #
+ just_testing=0
+ if [ "$just_testing" = 1 ] ; then
+     doitall="echo"
+ else
+     # Eval must handle arguments containing blanks.
+     doitall="eval"
+ fi
!
! # mpirun.ch_p4 exports nlocal and machinehead if not
! # a local run.
! if [ "$nlocal" = 1 ] ; then
!     tvcommand="$tvcommand -remote $machinehead"
! fi

+ # Note that this is run from within the mpirun.ch_p4
+ # script only (!), so the argument list is p4
+ # specific. FIX ME!
+ $doinall $tvcommand $progrnamemain -a $cmdLineArgs \
    -p4pg $p4pgfile -p4wd $p4workdir -mpichtv

```

You must also make a few changes to the **qsub** script. Here are changes you need to make when running TotalView from an **xterm** from the batch queue.

### **xterm batch script approach (Master node only)**

The script is very much the same except for two differences. One is that the **xterm** is launched via an **rsh** command back to the server on which the **qsub** command was originally run. The second difference is that since the

mpirun command will be executed on a different host, we need to copy the file pointed to by \$PBS\_NODEFILE on to a common file system. (The node files usually exist within a spool directory that is not common between the machines.)

```
#!/bin/sh

# Define qsub command options here. Each line
# to be processed by PBS should begin as "#PBS"
#
#PBS -l nodes=2
#PBS -N TotalViewJob
#PBS -v DISPLAY,PATH

export TOTALVIEW=/usr/toolworks/totalview/bin/totalview
export TVDSVRLAUNCHCMD=rsh

#
# The $PBS_NODEFILE may be in a local spool so copy it t
# to a shared filesystem
#
cp ${PBS_NODEFILE} ${HOME}/.pbs_nodefile-${PBS_JOBID}
export PBS_NODEFILE=${HOME}/.pbs_nodefile-${PBS_JOBID}

rsh ${PBS_0_HOST} \
"\
    cd ${PBS_0_WORKDIR} && \
    env PBS_0_HOME=${PBS_0_HOME} \
        PBS_0_LANG=${PBS_0_LANG} \
        PBS_0_LOGNAME=${PBS_0_LOGNAME} \
        PBS_0_PATH=${PBS_0_PATH} \
        PBS_0_MAIL=${PBS_0_MAIL} \
        PBS_0_SHELL=${PBS_0_SHELL} \
        PBS_0_HOST=${PBS_0_HOST} \
        PBS_0_WORKDIR=${PBS_0_WORKDIR} \
        PBS_0_QUEUE=${PBS_0_QUEUE} \
        PBS_JOBNAME=${PBS_JOBNAME} \
        PBS_JOBID=${PBS_JOBID} \
        PBS_QUEUE=${PBS_QUEUE} \
        PBS_JOBCOOKIE=${PBS_JOBCOOKIE} \
        PBS_NODENUM=${PBS_NODENUM} \
        PBS_TASKNUM=${PBS_TASKNUM} \
        PBS_MOMPORT=${PBS_MOMPORT} \
        PBS_NODEFILE=${PBS_NODEFILE} \
        PBS_ENVIRONMENT=${PBS_ENVIRONMENT} \
        DISPLAY=${DISPLAY} \
        PATH=${PATH} \
        TOTALVIEW=${TOTALVIEW} \
        TVDSVRLAUNCHCMD=${TVDSVRLAUNCHCMD} \
        xterm -title \"${PBS_JOBNAME}\" \
"

# Remove the temporary PBS_NODEFILE
rm ${PBS_NODEFILE}
```

When the `xterm` appears, you can use the following MPICH `mpirun` command to run your job.

```
mpirun -tv -nolocal -np 2 ./hello_beatles
```

### mpirun Batch Script Approach (Master node only)

The modifications you need to make when running just from the master node are as follows:

- Copy the PBS nodefile to a common location.
- Use the `rsh` command to log in to the node from which `qsub` was run to execute the `mpirun` command.
- Insure that important environment variables are available:
- Use the `-nolocal` command line option to the `mpirun` command.

Here is a script that you can modify:

```
#!/bin/sh

# Define qsub command options here. Each line
# to be processed by PBS should begin as #PBS
#
#PBS -l nodes=2
#PBS -N TotalViewJob
#PBS -v DISPLAY,PATH

export TOTALVIEW=/usr/toolworks/totalview/bin/totalview
export TVDSVRLAUNCHCMD=rsh

#
# The $PBS_NODEFILE may be in a local spool so copy it
# to a shared filesystem
#
cp ${PBS_NODEFILE} ${HOME}/.pbs_nodefile-${PBS_JOBID}
export PBS_NODEFILE=${HOME}/.pbs_nodefile-${PBS_JOBID}

rsh ${PBS_0_HOST} \
"\
    cd ${HOME}/src/hello-mpi && \
    env PBS_0_HOME=${PBS_0_HOME} \
        PBS_0_LANG=${PBS_0_LANG} \
        PBS_0_LOGNAME=${PBS_0_LOGNAME} \
        PBS_0_PATH=${PBS_0_PATH} \
        PBS_0_MAIL=${PBS_0_MAIL} \
        PBS_0_SHELL=${PBS_0_SHELL} \
        PBS_0_HOST=${PBS_0_HOST} \
        PBS_0_WORKDIR=${PBS_0_WORKDIR} \
        PBS_0_QUEUE=${PBS_0_QUEUE} \
        PBS_JOBNAME=${PBS_JOBNAME} \
        PBS_JOBID=${PBS_JOBID} \
        PBS_QUEUE=${PBS_QUEUE} \
        PBS_JOBCOOKIE=${PBS_JOBCOOKIE} \
        PBS_NODENUM=${PBS_NODENUM} \
        PBS_TASKNUM=${PBS_TASKNUM} \
        PBS_MOMPORT=${PBS_MOMPORT} \
```

```
PBS_NODEFILE=${PBS_NODEFILE} \  
PBS_ENVIRONMENT=${PBS_ENVIRONMENT} \  
DISPLAY=${DISPLAY} \  
PATH=${PATH} \  
TOTALVIEW=${TOTALVIEW} \  
TVDSVRLAUNCHCMD=${TVDSVRLAUNCHCMD} \  
mpirun -tv -nolocal -np 2 ./hello_beatles \  
" \  
  
# Remove the temporary PBS_NODEFILE  
rm ${PBS_NODEFILE}
```

### Notes

#### Using SSH

It is not uncommon for RSH services to be disabled on a cluster in preference of SSH. If this is the case, you will need to configure MPICH to use SSH instead of RSH.

#### About .cshrc, .bashrc, and other shell initialization scripts

Making job submission and debugging smooth for the PBS environment is dependent on passing environment variables between shells. Unfortunately, each time a shell is created, either as an effect of running the script or launching an **xterm**, the usual rules apply for shell initialization. This may have undesirable side-effects like resetting the **DISPLAY**, **PATH**, or **LD\_LIBRARY\_PATH** environment variables. While the examples in this section have explicitly set environment variables, shell initialization files are called after this occurs and can reset this information.

#### stdin, stdout, and stderr

The **stdout** and **stderr** streams for batch jobs are queued and the streams are written by PBS for batch jobs. If TotalView is launched from the batch queue, you will not be able to see the **stdout**, **stderr**, and control **stdin**.