1 Introduction to C/C++ and MPI

1.1 Compiling Programs using MPI

Programs with MPI routines require special libraries and runtime facilities to be compiled into the final executable. To include the MPI related libraries, you can use the UNIX shell script (mpicc or mpicxx) to compile MPI programs. Mpicc or mpicxx uses GCC (or other compilers) as the backend compiler but sets up all of the environmental parameters needed for successful compilation. The following command should be used to compile the program:

```
cd (to where you save your files)

mpicc -o Helloword Helloword.c
```

1.2 Components of a MPI Program

```
/* Helloword.c */
#include <stdio.h>
#include "mpi.h"
void main(int argc, char* argv[])
{
        int   my_rank;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
        printf("Hello from process %d\n", my_rank);
        MPI_Finalize();
}
```

1.3 Running a MPI program

A set of steps (or programs) are involved to ensure the user application is executed correctly.

1.3.1 Starting the MPI Daemon (1.2.x release series and before)

The daemon is responsible for managing the MPI applications as they execute, in particular the daemon processes many of the communications and message transmission that MPI applications require to execute. To start a daemon, execute the following command:

```
mpd &
```

Note: "mpd" refers to a "multiprocessing daemon" that runs on each workstation.

To stop MPD, use command:

```
mpdallexit
```

Starting from version 1.3.x, another process manager hydra is introduced.

1.3.2 Finding out which machines are in the MPI ring

An MPI ring is a collection of machines which MPI programs can use to execute (they are registered to a central daemon). The following command is used to find out which machines are connected to the MPI ring:

```
mpdtrace
```

1.3.3 Executing a MPI program

The following command needs to be issued to execute a MPI program. Note: You cannot load MPI executables from the UNIX command line since they need to connect to the locally running daemon.

```
mpirun -n 4 ~/Helloword
```

Note: mpirun (or mpiexec) is called a launcher and is the basic remote node access mechanism. It is the tool that communicates with the mpd daemon to start MPI applications. The mpd daemons are already in communication with one another before the job starts. The program mpirun runs in a separate (non-MPI) process that starts the MPI processes running the specified executable. It serves as a single-process representative of the parallel MPI processes in that signals sent to it, such as ^Z and ^C are conveyed by the MPD system to all the processes.

Notice the inclusion of the -n 4 option instructs mpirun to run the program on four processors - these may be local processors or nodes within a cluster. If four processors are not available the ring will wait (hang) until they are. You may prefer to the use the -np 4 option instead which instructs mpd to simulate four processors through using threads. This will execute immediately but may not provide any parallel performance improvement as the execution may be happening in four threads on the same processor. In general you should use the -np option to test that your code works fully and then use the -n option to obtain performance timings.

2 Portable Batch System

The Portable Batch System (PBS), is a batch job and computer system resource management package. It will accept batch jobs (shell scripts with control attributes), preserve and protect the job until it is run, run the job, and deliver output back to the submitter [2, 3].

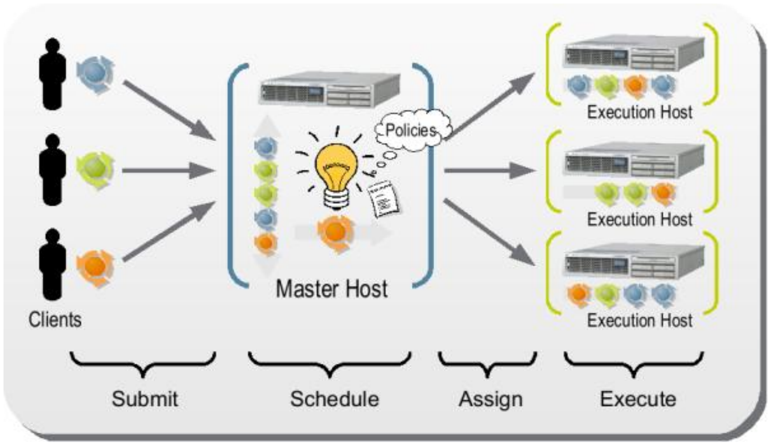


Figure 1. https://blogs.oracle.com/templedf/entry/sun_grid_engine_for_dummies

2.1 Components of PBS

PBS consists of four major components: **commands, the job Server, the job executor, and the job Scheduler.**

User commands: **qsub, qstat, qdel**

qsub [script] : to submit an executable script to a batch server, which creates a job.

qstat [-u user_list] [job_identifier... | destination...] : The qstat command is used to request the status of jobs, queues, or a batch server. The requested status is written to standard out.

qdel job_identifier ... : The qdel command deletes jobs in the order in which their job identifiers are presented to the command. A job that has been deleted is no longer subject to management by batch services.

Job Server

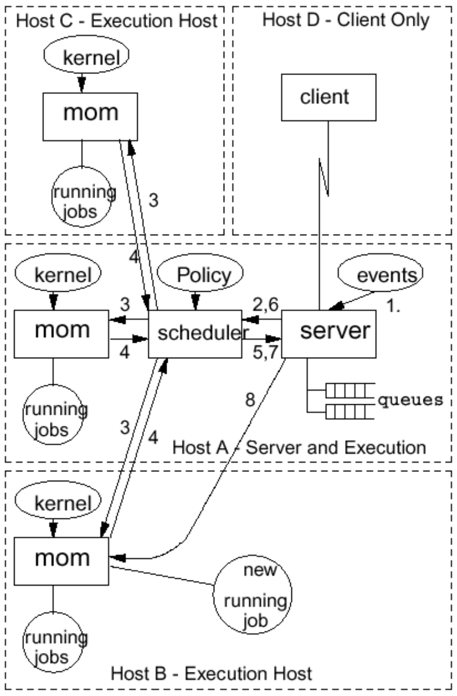
The Job Server is the central focus for PBS. It is generally referred to by the execution name as pbs_server. All commands and the other daemons communicate with the pbs_server via an IP network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, protecting the job against system crashes, and running the job (placing it into execution). One server manages one or more **queues**; a batch queue consists of a collection of zero or more batch jobs and a set of queue attributes. Jobs are said to reside in the queue or be members of the queue. Access to a queue is limited to the server which owns the queue. All clients gain information about a queue or jobs within a queue through batch requests to the server. Two main types of queues are defined: **routing queues** and **execution queues**. When a job resides in an execution queue, it is a candidate for execution. A job in execution is still a member of the execution queue from which it was selected for execution. When a job resides in a routing queue, it is a candidate for routing to a new destination. Each routing queue has a list of destinations to which jobs may be routed. The new destination may be a different queue within the same server or a queue under a different server. The Job Server must know the list of nodes that can execute jobs: they are declared in a file in the server private directory PBS_HOME/server_priv.

Job Executor

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

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.



1. Event tells Server to start a scheduling cycle
2. Server sends scheduling command to Scheduler
3. Scheduler requests resource info from MOM
4. MOM returns requested info
5. Scheduler requests job info from Server
6. Server sends job status info to scheduler Scheduler makes policy decision to run job
7. Scheduler sends run request to Server
8. Server sends job to MOM to run

3 Running MPI Parallel Programs within PBS or other batch system

Job scripts [4]

Jobs are submitted to the compute nodes via the patch system by a script (note: ND uses SUN Grid Engine batch submission system) . Basic SGE batch scripts should conform to the following template:

```
#!/bin/csh

#$ -M netid@nd.edu          # Email address for job notification
#$ -m abe                   # Send mail when job begins, ends and aborts
#$ -pe mpi-12 12  # Specify parallel environment and legal core size
#$ -q long                  # Specify queue
#$ -N job_name             # Specify job name

module load xyz            # Required modules

mpiexec -n $NSLOTS ./app # Application to execute
```

Note: **mpiexec** is a replacement program for the script **mpirun**, which is part of the **mpich** package. It is used to initialize a parallel job from within a PBS batch or interactive environment. **mpiexec** uses the task manager library of PBS to spawn copies of the executable on the nodes in a PBS allocation [5].

Reasons to use **mpiexec** rather than a script (mpirun) or an external daemon (mpd):

- Resources used by the spawned processes are accounted correctly with mpiexec, and reported in the PBS logs, because all the processes of a parallel job remain under the control of PBS, unlike when using mpirun-like scripts.
- Tasks that exceed their assigned limits of CPU time, wallclock time, memory usage, or disk space are killed cleanly by PBS. It is quite hard for processes to escape control of the resource manager when using mpiexec.
- Starting tasks with the task manager (TM) interface is much faster than invoking a separate rsh * once for each process.
- You can use mpiexec to enforce a security policy. If all jobs are forced to spawn using mpiexec and the PBS execution environment, it is not necessary to enable rsh or ssh access to the compute nodes in the cluster.

Ref:

[1] http://www.mcs.anl.gov/research/projects/mpi/mpich1-old/docs/

[2] http://www.OpenPbs.org/

[3] http://hpc.sissa.it/pbs/pbs-1.html#ss1.1

[4] http://wiki.crc.nd.edu/wiki/index.php/CRC_Quick_Start_Guide

[5] http://www.clusterresources.com/torquedocs21/7.1mpi.shtml

[6] http://arc.liv.ac.uk/SGE/howto/mpich2-integration/mpich2-integration.html