DQS Users Guide

Louis S. Revor
Computing and Telecommunications Division
Argonne National Laboratory

September 15, 1992

1 Introduction

DQS[1] - Distributed Queuing System - is an effective method for distributing the batch workload among multiple unix-based machines. In doing so, it increases the productivity of all of the machines and also increases the number of jobs that can be completed in a given time period. Also, by increasing the productivity of the workstations, the need for outside computational resources is reduced.

A job is submitted to the DQS system via a job shell script. The job script is a standard unix shell script (Bourn shell, C shell, or Korn shell) that will executed on one of the DQS machines when the appropriate resources become available. The script can also be used to tell DQS to do the following :

- Initiate the script under a specific unix shell.
- Write all output sent to **stdout** and **stderr** to a specified file.
- Mail a specific user when any changes are made to the status of the job.
- Prepare the system for running a parallel job under PVM[2] - Parallel Virtual Machine.
- Specify the type of machine you are requesting that the job should be run on.

The script file can be used to do multiple tasks. By combining shell commands with DQS commands, the user will be able to use the system more efficiently. A proper understanding of the script file is the key to learning how to run existing code under the DQS system.

Once the job script has been submitted to DQS, it is scheduled and then sent to one of the machines on the system when one becomes available. The process DQS uses to distribute the workload and decide which machine each job will be sent to will be discussed later.

Included in the system are routines that were developed for the user to have control over his jobs. These routines provide the ability to check the

[1]Developed at the Supercomputer Computations Research Institute at Florida State University by Tom Green and Jeff Snyder

[2]PVMwas developed at the University of Tennessee, Oak Ridge National Laboratory, and Emory University

status of a job(s), delete a job from the queuing system, and check on the availability of resources.

For increased ease of use, the DQS system provides an X windows interface. All users of the system that are running under X windows can use these routines. With these routines, the user can accomplish any task that he could have done from the command prompt with a little more ease. Also, the X interface allows users to get an interactive xterm window through DQS if he feels the job must be run interactively.

The DQS system was also developed with console users in mind. Routines are provided that monitor the X window server for activity to determine if the interactive user is at the terminal. If activity is detected, then DQS can suspend the queues that are on the machine. If the queues are suspended and the terminal has been inactive for a given period, DQS will unsuspend (enable) the queues on the machine.

Overall, DQS provides efficiency, greater productivity, and increased cost efficiency while still maintaining functionality.

2 Overview

The DQS system has one routine that controls the system. This routine and the machine that it runs on are called the **qmaster**. The *qmaster* is responsible for processing new requests (jobs submissions, queries, etc.), keeping track of already existing jobs, monitoring the status of all of the machines known to DQS, and updating all of the files used for accounting and management of the system.

All requests that are submitted to the DQS system are processed by the *qmaster* . This does not mean that the user has to be logged on to the *qmaster* in order to submitt a request. A request can be submitted from any machine the DQS system considers to be a trusted host. When the request is submitted, it is passed across the TCP/IP network to the *qmaster* for processing.

When a user submits a job, he must specify the resources that he wants the job to use. This is to tell DQS how many and what kind of machines he needs for the job. The two different types of resources in the DQS system are the **queue** and the **group**.

The *queue* is the standard computer science first-in-first-out (fifo) queue. Each *queue* resides on a particular machine. A job is sent to a machine by submitting it to the *queue* that resides on the machine. If the *queue* is busy with a job when a new job is submitted, the new job simply waits its turn in line in the *queue* . More than one *queue* may reside on a particular machine, but this is not recommended except for the purpose of having *queues* of differing priorities or if the machine has sufficient resources for handling more than one *queue* .

A *group* is a collection of *queues* , where each *queue* inside the *group* resides on a machine of a particular architecture. This is how DQS solves the problem of having multiple types of architectures. When the user specifies which *group* he wants, he is in effect specifying the class of machines he would like. When a job is submitted to a *group* , DQS sends the job to a *queue* in the *group* that is available. If all of the *queues* are busy when a job is submitted to the *group* , the job will wait its turn in line at the *group* level. This feature of waiting at the *group* level is what allows for load leveling across a network of machines.

It is better for all users to submit jobs to a *group* rather than a *queue* due to the load leveling capabilities of DQS. If all jobs are submitted to the *group* , the amount of waiting time will be minimized, because all of the machines inside the *group* will be efficiently utilized. The only reason a user should submit a job to a particular *queue* is if the code was designed to run on a particular machine instead of a class of machines.

3 Submitting a DQS Job

Since the major task for most users will be submitting jobs, this topic will be covered first. As mentioned earlier, jobs are submitted to the DQS system by submitting a unix shell script. Therefore, most of this section will deal with describing the shell script and how to customize it for DQS.

A shell script is a standard unix text file. It can be created by using any text editor (vi, ed, emacs, xedit, etc.). A DQS job shell script has two distinct purposes. The first is define the environment needed by the job. The second is to give the commands that comprise the job itself. Listed below is a sample script that could be submitted to the DQS system to compile a sample FORTRAN code and then execute the compiled program.

```
# This is a sample script file for compiling and
#  running a sample FORTRAN program under DQS.

# We want DQS to send mail when the job begins
#  and when it ends.
```

```
#$ -mu EmailAddress
#$ -mb
#$ -me

# We want to name the file for the standard output
#  and standard error.

#$ -eo test.out

# Change to the directory where the files are located.

cd TEST

# Now we need to compile the program 'test.f' and
#  name the executable 'test'.

f77 test.f -o test

# Once it is compiled, we can run the program.

test

# End of script file.
```

This script file is a short example, but it shows that script files are easy to use.

In this script file, there are three types of lines. These are comment lines, DQS command lines, and shell command lines. Each of these types will be described below.

Any line in the script file that has the '#' character in column one and is not followed by a '$' or a '!' is a comment line and is not considered by DQS or the unix shell. They are included in the file to provide information on what the file is supposed to be doing. Included in this script file are several comment lines to help explain what it is doing to those who are new to script files.

Lines in the script file that begin with #$ in columns 1 and 2 are commands to DQS. Our example contains four such commands. The **-mu** command tells the *qmaster* who to send all mail to. The **-mb** command signifies that mail should be sent when the job begins, while **-me** states that mail should be sent when the job completes. The **-eo** command specifies the file that all output to **stdout** and **stderr** (standard output and standard error) should be redirected to. All DQS commands are read when the job is submitted regardless of their position in the script file. These same DQS commands could have been specified on the **qsub** command line when the job was submitted. Any valid command line parameter for the *qsub* command can be placed in the scriptfile as a DQS command. Users should be aware of the parameters that are present in the scriptfile so they do not unintentionally replicate the parameter. If more than one resource request is present, DQS will assume that the job will need the stated resources and reserve them for the job. Parameters other than resource requests should not be duplicated. These parameters will be discussed later along with the *qsub* command and are given in appendix A.

The final type of command in a DQS script file is the unix shell command. Any line in the script file that does not begin with the '#' character (with the exception of the '#!' combination) is considered to be a unix shell command. When the job is run, each shell command will be executed in order starting from the top of the file. These commands will behave in the same manner as if they were issued from the unix prompt. For testing, a DQS script file can be run as a standard script file from the unix prompt if it does not contain any references to environment variables that are set by DQS. When this is done, all DQS commands will be treated as comments, because they all begin with the '#' character. To execute a DQS script file , the user must only give it execute privileges with the unix **chmod** command.

Once the script file is written, it can be submitted to DQS. The routine that submits jobs to DQS is called **qsub**. Two examples of *qsub* calls to submit the above script file are :

```
qsub -G groupname 1 sub
qsub -q queuename sub
```

where groupname is the DQS *group* that is being requested, queuename is the DQS *queue* that is being requested, and sub is the name of the scriptfile described above.

In the above examples, the -q and -G parameters are both resource requests. It should be noted that both of the *qsub* commands have a resource request. A job will not be accepted without specifying a resource on the command line or in the job script. If a resource is not specified, DQS would not know where to send the job. The '1' after groupname in the first example tells DQS that you will only need one *queue* in the *group* that is specified. This number should only be greater than one for a parallel job under PVM. If the job is not a parallel job, more than one resource should not be requested and the number after the queuename should be left at '1'. In any case, a value must be specified for the job to be accepted.

In the first *qsub* example, the job 'sub' will be sent to one of the *queues* in the *group* 'groupname' and then executed. In the second *qsub* example, the job will be sent to the *queue* 'queuename'.

The syntax for the *qsub* command is :

```
qsub [queue] [parameters] scriptfile
```

There are several command line parameters that can be given with the *qsub* routine. A list of these parameters is given in appendix A. These options may be used either on the command line for *qsub* , or placed in the job script file. They will have the same effect in either place.

4 Monitoring Jobs

Once a job has been submitted to the DQS system, it becomes almost invisible to the user. Because of this, DQS has routines included with it that allow a user to keep up with his job.

The first method is to have the DQS system send mail messages about the job. DQS can be told to mail a user when the job is started, finished, suspended, or enabled. All of this is done using the *qsub* command which is described above. First, the user that will receive the mail must be specified. This is done with the -mu option of *qsub* . After this is set, any of the -mb, -me, or -ms option can be specified which tell DQS to mail at the beginning of a job, end of a job, or when a job is suspended/enabled respectively.

The second method is to query DQS about the status of the job. This is done with the **qstat** command. The *qstat* command reports information on each of the *queues* and all of the jobs that are pending. For every *queue* , it gives information on the *queue* and also on the the job (if any) that is running on the *queue* . Users can use this information to see the status of his jobs.

The *qstat* command gives a lot of useful information, but some of the information is often not very relevant. Because of this, the *qstat* command has several command line options that allow the user to tell DQS what information that he wants reported. These options are given in appendix A.

5 Killing a DQS Job

If a user decides that one of his jobs is no longer necessary, he can delete the job from the queuing system. The command for doing this is the **qdel** command. Only the owner of a job, or a manager of the DQS system can delete a job from DQS. The syntax for this operation is :

```
qdel jobnumber
```

Where jobnumber is the number of the job that is to be killed.

The jobnumber is returned by the *qsub* command when a job is submitted. If the user forgets the number for the job, it can be obtained by the *qstat* command. It is useful to know that every job is named jobnumber.scriptfile where scriptfile is the name of the job script file that was submitted.

6 Appendix A

The following is a quick reference for the syntax of the commands covered in this document.

6.1 qsub

qsub allows a user to submit a batch job to DQS.

SYNTAX

```
qsub [queue] [parameters] scriptfile
```

PARAMETERS

-e fname Tells DQS to redirect all output to STDERR to the file specified by fname. DQS defaults to jobnumber.jobname.ERR if this option is not specified.

-o fname Tells DQS to redirect all output to STDOUT to the file specified by fname. DQS defaults to jobnumber.jobname.OUT if this option is not specified.

-eo fname Tells DQS to redirect all output to STDERR and STDOUT to the file specified by fname.

-cwd The cwd flag tells DQS to execute the script from the same absolute directory that the job was submitted from. If this is not specified, DQS will default to the users home directory, and the files will not be found, unless they are in the home directory or the script file changes the directory via the cd command. If the absolute pathnames are not the same across the machines, then the user should use the cd command instead of setting the -cwd flag. If the user sets the cwd flag and the machine cannot find the directory, then it will default to the users home directory on that machine.

-mu EmailAddress Tells DQS who to send mail to if the -mb, -me, or -ms flags are set. This must be set if any of the mail flags are set.

-mb Tells DQS to send mail to notify when the job has begun.

-me Tells DQS to send mail to notify when the job has finished.

-ms Tells DQS to send mail to notify when the job has been suspended or enabled after a suspend.

-G groupname num This option tells DQS that the job will need 'num' *queues* in the *group* 'groupname'. More than one -G parameter may be specified, and they can be used in conjunction with the -q parameter. However, only one resource should be specified unless the job is a parallel job under PVM. If the job is not a parallel job, it will be keeping resources unavailable that could be used by other jobs.

-q queuename This option tells DQS that the job will need to use the *queue* 'queuename'. More than one -q parameter may be specified, and they can be used in conjunction with the -G option. However, users should not specify a specific *queue* unless the code was written for a specific machine.

-pvm This flag tells DQS that the job will be using PVM. DQS will then set up the environment appropriate for running a PVMjob by creating the PVMhost file and starting the PVMdaemons on each of the allocated resources. This process is invisible to the DQS user.

-M queuename The tells DQS that the PVMmaster should be on the *queue* 'queuename'.

-s shell This specifies the shell that the script needs to execute under.

-v This is the verify option. With this option set, the command line and script file are parsed and the requests are then written to STDOUT.

-H With this flag set, all request are considered to be hard. This means that the job cannot be executed until all resource requests are available. This is the default for all jobs.

-S This sets all of the resource requests to be considered soft. The job can be executed even when all of the resources are not available.

-r This enables the job to be restarted by DQS if a system error occurs while it is running. This is the default.

-nr This tells DQS not to restart the job if a system error occurs when the job is running.

6.2 qstat

qstat allows users to see the status of the *queues* and jobs in the DQS system.

SYNTAX

```
qstat [parameters]
```

PARAMETERS

-s Reports only the *queue* name, architecture, and the status of the *queue* . It also reports the job id and job owner if there is a job running on the *queue* .

-A arc Reports information only on *queues* of the given architecture type.

-G group Reports information only on the *queues* that are in the *group* specified.

-q queue Reports information only on the specified *queue* .

-r Sort the *queues* in increasing order of load average.

-u username Report only information on jobs owned by the user specified.

6.3 qdel

qdel deletes a job from the DQS system.

SYNTAX

```
qdel jobnumber
```