sge-tutorial

Introduction

This is a short overview of how to use the batch queuing system on the CCBB clusters. It's a work in progress so please let us know if something is found to be not correct, or is confusing. In the tutorial below, text in blocks colored, and set off from the main text like this

```
This is a sample line.
```

is the literal text of one or more commands typed at the UNIX prompt, and the resulting output; or it's the literal contents of text file of some sort. Text contained in brackets like this <text> is an indicator that you are supposed to replace the brackets and texts with something more meaningful to your job script.

Basic Overview

Processing data on the cluster is a 5 step process consisting of

1. Naming the processing run, and creating a project directory
2. Uploading data files to project directory
3. Devising proper processing scheme
    a. What programs are to be used
    b. What needs to be copied over to the node for processing
    c. What needs to be copied back to be saved.
4. Write, and submit job script; or submit request for interactive login to a node
5. Once processing is done review logs to make sure no problems occurred

The last step is critical. You cannot assume that because your job finishes it actually completed successfully. Any number of things can go wrong, such as a node crashing, a resource (disk, memory) being exhausted, programming errors, or usage errors. We cannot predict or control when these things happen so you should be prepared for that. As mentioned below the output and error messages generated during a job run on the cluster are captured into output, and error files for you to review.

Depending on what software is needed we have some pre-written scripts that you will find in /share/scripts (or via \\files.ccbb.utexas.edu\scripts). We can help write others for you as needed, or you can use these as a template. Each of our scripts has several variables that tell the script where to find source data, config files, and other parameters that alter the job. Changing just these few items is enough to have a workable script that you can submit.

Whether using our scripts, or writing your own, it's advisable to run them a few at a time while you determine how much shared resources such as RAM, and disk that you need.

Submitting Jobs

In some cases, you may just want to run a very quick job, or otherwise just need to run some commands without packaging them up in a job script. If the footprint of the jobs is light, then they can just be run directly on the headnode after logging in. If the jobs are going to use a lot of memory, a large percentage of available CPU time (ie, be CPU bound), or a large percentage of avaiable Input/Output bandwidth (ie, be disk bound), then it is better to run them within SGE. If you run commands on the script,and those have any amount of significant amount of runtime, please monitor their CPU usage. If we let the too many CPU hogs pile up on the headnode it will quickly become very painful for interactive users trying to move files around, write scripts, and submit them. If you still do not know enough about your data processing to script it, it's possible to use the '"qlogin"' command to request a slot from SGE. Here is an example.

```
[cdupree@phylocluster sge]$ qlogin
waiting for interactive job to be scheduled ...
Your interactive job 4799 has been successfully scheduled.
Establishing /opt/gridengine/bin/rocks-qlogin.sh session to host compute-0-2.local ...
Last login: Sun Nov 18 11:29:25 2007 from phylocluster3.local
Rocks Compute Node
Rocks 4.1 (Fuji)
Profile built 20:27 01-Nov-2007

Kickstarted 16:35 01-Nov-2007
[me@compute-0-2 ~]$
```

Here SGE has logged me into node c0-2, and now I'm sitting at another shell prompt. When done with my processing, I type "'exit"' and I'm returned back to the head node:

```
Connection to compute-0-2.local closed.
/opt/gridengine/bin/rocks-qlogin.sh exited with exit code 0
[cdupree@phylocluster sge]$
```

Please avoid tying up a slot, and exit when you are done with your processing. If jobs start queuing up, and it's determined that an interactive session is not actively being used, then that session will be killed to free up the slot.

By far the nicest way to run jobs, is to write a job script and submit it for a non-interactive run. This lets you exit out of the head node, and walk away waiting until the job is done running. Here is a simple script, in a file called tut1,

```
#!/bin/bash

echo Hello World
```

This script when run just outputs "Hello World" and then ends. To submit the job into the queue you would use the command

```
qsub -S /bin/bash tut1
```

Right after doing this, you can type

```
qstat -u your_eid
```

you should see something like this

```
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-
ID
-----------------------------------------------------------------------------------------------------------------
-
  74952 0.00000 tut1       eid      qw    12/02/2007 11:55:54                                    1


```

which shows that the job is queued up and waiting for submission. If you type qstat again, you'll probably see the job seems stuck in this state, and may stay that way for a few minutes. If there are slots available, then the job will get run, and this job should exit very quickly after that.

We can make one quick simplification to the script. The -S /bin/bash qsub option specifies the shell which should be used to interpret the job script. We can make the submission process a bit easier, by exploiting the fact that SGE will interpret lines beginning with "$#" as arguments to the qsub command. Thus, we can modify tut1 to become tut2

```
#!/bin/bash
#$ -S /bin/bash

echo Hello World
```

You might wonder why you need to still use the sh-bang line "#!/bin/bash" if SGE is just going to ignore it. In fact, it's only there so that your job script can also be used as a normal shell script.

Writing Job Scripts

More About Job Script Options

There are a number of commonly useful qsub options (or if you'd rather options to embed in your scripts). We've already seen "-S /bin/bash" which specifies the shell used to interpret the job script. You can also use "-q <queue>" to select a queue (read below to determine what queue might exist on your system, and replace "<queue>" with one of those. You can actually specify a particular node with "queue@node", but generally this is not needed.

Now look back in your home directory. You should see two files named something like this:

```
tut1.e6216
tut1.o6216
```

As mentioned in the [UNIX tutorial] every UNIX process has the concept of standard input from which it reads input, standard output to which it writes normal output, and standard error to which the program can write critical error messages (note that which of these output channels a program uses is it's own choice; most user written programs tend to use only standard output). It's obvious from the above examples that SGE sets your job script up so that when it runs, all programs started in the script will use "<scriptname>.e<job id>" for error messages, and "<scriptname>.o<job id>" for output messages. These can be overridden by using

```
-e <filename>
-o <otherfilename>
```

which hard codes the names of the output and input files. A related option is to override SGE's use of the script name; this is done with the option

```
-N <jobname>
```

Two final options which are generally useful are used to instruct SGE to mail you when important events occur. These are

```
-m beasn
-M <email>
```

with 'beasn' having the following meaning

- b - beginning of job
- e - end of job
- a - job is aborted or rescheduled
- s - job is suspended
- n - never

Choose whatever set of options you wish.

Note that SGE depends on the seeing "$#" at the beginning of a line in order to find its options. If you temporarily need to override an option you can do so by separating the "$" and "#", by putting a space at the beginning of the
line, or by putting in an extra "#" to make "##$".

Shell Environment Variables

SGE adds the following shell variables to each job it runs

- $HOME home directory of the user on execution machine
- $USER user ID of job owner
- $JOB_ID current job ID
- $JOB_NAME current job name (see -N option)
- $HOSTNAME name of the execution host
- $TASK_ID array job task index number

Bare Bones Script

Generally a processing run will consist of the following steps:

1. Create local working directory
2. Transfer input files (data and science config files) to working directory
3. Perform science processing
4. Copy results back to home directory

Here is a bare bones script that accomplishes this.

```
#!/bin/bash

# SGE Options
#$ -S/bin/bash
#$ -N MyJob

# Create Working Directory
WDIR=/state/partition1/$USER/$JOB_NAME-$JOB_ID
mkdir -p $WDIR
if [ ! -d $WDIR ]
then
  echo $WDIR not created
  exit
fi
cd $WDIR

# Copy Data and Config Files
cp $HOME/Data/FrogProject/FrogFile .

# Put your Science related commands here
/share/apps/runsforever FrogFile

# Copy Results Back to Home Directory
RDIR=$HOME/FrogProject/Results/$JOB_NAME-$JOB_ID
mkdir -p $RDIR
cp NobelPrizeWinningResults $RDIR

# Cleanup
rm -rf $WDIR
```

Monitoring Queue Status

You can use the "qstat" command. By default this shows a list of jobs running in the queue and their state. The output will look like this

```
job-ID  prior   name       user   state   submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
-
74950 0.55500 tut1       eid     qw     12/01/2007 20:42:45 queue@compute-0-6.local           1
```

and lists out the job id, priority, name, user, state, submission time, queue, and CPU slots used by the job. You can view individuals jobs using the

```
qstat -j <job_id>
```

option. The output in this case is much more verbose, and includes information about the state of the job, and queuing considerations. You can also use the

```
qstat -u <user_id>
```

to see only your jobs. One final option is to use the

```
-f
```

option to see the status of the queues on the systems. Note that if you are on one of the clusters with more than one queue (see below), then with -f you must also select a queue using

```
-q <queue_name>
```

Deleting Job Scripts

It might happen that you realize there is a mistake with a job, or want to move it to another queue. In this case, you can use the "qdel" command to remove the job. This is true whether the job is still in the wait state, or is running. The syntax is

```
qdel <jobid>
```

You can also use

```
-u <user_id>
```

```

```

to remove all of the jobs you have on a particular system.

Cluster Queues

Some of the CCBB clusters might have special queues set up. This is typically done so that the research group owning a cluster, can preempt the use of a node by a non-group member. Sometimes it might be done so that a set number of slots can be reserved for a class to use to complete homework assignments or labs. These queues tend to be very ephemeral in nature, though. Below is the list of queues

Note that on systems with multiple queues you must use -q with qsub to select the queue which you wish to use. Otherwise SGE will use all available queues, and this is probably not what you want to happen.

- queues on phylocluster
    - all.q - all users
    - phylo - old 4 processor, 2 GB RAM systems (being phased out)
    - bigmem - 16 processor, 64 GB system (all can use)
    - kirkp - 16 processor, 64 GB systems (all can use, but overridden by kirkp-lab queue use)
    - kirkp-lab - same nodes as kirkp. When a job is placed on a node in this queue, the same node is suspended in kirkp. This stops currently running jobs from running in that queue on the node, and it stops further submissions on to that node. This queue should only be used if you you need to ensure that you are reserving all of the resources to a node, or that you need to override the use of a node which is full. This queue is restricted to use by Kirkpatrick lab members only.
    - wilke - queue for Wilke lab users
    - test - uses 1 of the older phylo nodes. This queue is intended for people that need to test a job script, or other short running program. Jobs in the test queue are terminated after 15 minutes of wall clock runtime. (Note: activation of this queue is pending).

Linux-related Problems & Solutions

Some user may find their script doesn't work as they expected. The reason could be just Linux basics related.

1. Some program require you include Full Path of the input file or other files.

Full Path is also called absolute path.

Use ***ls $PWD/filename.extension*** to get the full path of the files. It looks like /root-level/second-level/third-level/filename.extension.

./ or ../ or $HOME or ~/ is Relative Path.

2. How can lock one node so that only me can use the cores on that node and run multi-thread program?

Use **qusb -pe serial 16 qsub_Job_script_name.qsub**

where 16 is the number of cores you need.