BCNI Computer Leaflet Series

Using BCNI Grid Engine

This leaflet introduces the BCNI Grid Engine (BCNIgrid)

What is a Grid Engine?

A Grid is a pooling of multiple computer resources that can speed up data processing. The Grid Engine on the BCNI is called BCNIgrid.

How does it work?

Normally, when you start a program you communicate directly with the processor. When a program starts in the Terminal Window, it waits until it has finished, and the 'prompt' returns.

With Grid computing, you submit your *job* (one or more programs) to the *scheduler* that calculates the best computer servers and the best time to run your job. It is designed to make sure that overall, everyone gets their jobs processed as quickly as possible. This means your job might not start immediately and you will not know in advance which servers will process your job. However, in almost all circumstances it will be a lot quicker than running your program in the Terminal Window

How do I get started?

BCNI users who applied for an account after **13 May 2014** can skip this step. If you applied for your BCNI account before this date you will need to setup your environment in the following way:

(1) Open a Terminal Window and type

```
gedit ~/.bashrc
```

(2) Add the following lines to the end of the file:

```
. /opt/sge/default/common/settings.sh
export FSLPARALLEL=1
```

(3) Save the file

You will need to restart the Terminal Window for the changes to come into effect.

How do I submit a job?

First using a text editor, create a *script* containing the commands you want to run. Give it a suitable name; for example `myscript.sh`. The commands in the script are just how you would type them if you were running them directly in the Terminal Window.

Then from a Terminal Window, type:

```
qsub myscript.sh
```

Note that the 'prompt' is returned in the Terminal Window immediately since you are simply instructing the scheduler to run the job for you.

There are different queues for jobs that are estimated to take different times. Choosing the most appropriate queue for your job will reduce the overall time for processing.

Queue	**Max time per job***	**Good for**
`veryshort.q`	3 hours	Jobs that take an hour or so to complete, e.g. FSL 1st level analysis
`short.q**`	1 day	Jobs that take longer than 2+ hours, e.g. FSL 2nd level analysis and FreeSurfer
`long.q`	3 days	For very long jobs such as big FSL 2nd level analyses
`verylong.q`	7 days	For jobs that take a very long time

* Max time per job refers to the maximum time a job can take before the scheduler assumes there's a problem and terminates. The clock only starts when the scheduler starts the job. A job can stay on the queue indefinitely.

** `short.q` is the default queue, and is the best queue to use if you are unsure how long a job will take.

To select a queue use qsub's "`-q <queue_name>`" option. For example, to get myscript.sh to use `long.q` type

```
qsub –q long.q myscript.sh
```

If your request is successful you will see the line:

```
job XXXXX ('script') is submitted successfully.
```

where `script` is the job name (the same as the script you submitted) and `XXXXX` is the *job-ID*. Take a careful note of the job-ID as this is the most reliable way of referring to your job.

You may start multiple jobs.

How do I know when the job is finished?

There will be no further notification from the scheduler about the status of your job. If you want to see how things are progressing, on the command line type:

```
qstat
```

and you will get (with some items missing for brevity):

```
job-ID     ... name     user    state  Submit/     queue
                                       start at
-------------------------------------------------------------
98765          script   userid  qw     05/14/2017  short.q
                                       11:10:09
```

Most fields are self-explanatory. Each character in the `state` column tells you about your data processing job, the commonest being: q (for queuing); w (for waiting); h (for holding); t (for transferring); and r (for running).

If there is no reply then either your job(s) has finished, or there was a problem with the job submission.

Where is the output from the job?

Often, programs write to screen. Processing on BCNIgrid is done without a screen, so BCNIgrid writes the screen output to a file in your home directory using the following naming convention:

`<job name>.o<job-ID>`

Will BCNIgrid work with FSL?

If you setup your `~/.bashrc` file as above, FSL will automatically use BCNIgrid . Note that FSL will also return the prompt to you immediately.

Use `qstat` to find out the status of your FSL processing, as above, or open `/path/to/outdirname.feat/report_log.html` using a web browser.

Troubleshooting

My job works fine in Terminal Window, but on BCNIgrid it complains that it cannot find files or directory

Possibly you have relative pathnames:

e.g. `myfile.txt` or `../myfile.txt`

instead of

`/path/to/myfile.txt`

Changing all relative pathnames to full path names will help.

When submitting jobs I get "Unable to read script file because of error: error opening <program>"

The job scheduler wants to copy the <program> to a safe location, but it cannot. This error report is normally seen when you are trying to run Linux's built-in command. Putting the command in the script should solve this problem.

Further Information

The full user documentation is at
http://www.subnetz.org/~til/sge/SGE_60_Users_Guide.pdf

Documentation for FSL can be found at http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/SGE%20submission%20FAQ and http://chrisfilo.tumblr.com/post/579493955/how-to-configure-sun-grid-engine-for-fsl-under-ubuntu

There are some example jobs in /opt/sge/examples/jobs which are worth looking at.

Contact Roger Tait (rt337@cam.ac.uk) and Cinly Ooi (co224@cam.ac.uk) if you have any question about BCNIgrid