sge-task-array

The use of task arrays is one way to have a single job script which processes several data sets. Of course, you could just submit the job script several times, but this puts a burden on the head node. Also, you might have to edit the script to make it data set specific. You could also write your job script to execute in a loop, although again if the loop value changes, then you would again have to edit the job each time you submit it. Task arrays make this much nicer as 1 job is submitted with a certain number of tasks specified. SGE then takes care of partitioning the threads out to individual nodes. If you have to remove the tasks, there is just the one job idea which has to be removed. Additionally, SGE puts a new variable name in to the environment of each job, so that the script can be made more generic.

Here is a very simple job script which just does some variable substitutions, and then exits.

```
#!/bin/bash

#$ -S /bin/bash

#$ -N job

#$ -m bea

echo this is a line of output # except for this

#echo this is another line of output > output 2> error

#echo this is yet another command > command-2 2>& error-2

# Create Working Directory
WDIR=/state/partition1/$USER/$JOB_NAME-$JOB_ID

echo $WDIR
```

To make this task aware we use the qsub option -t which takes an argument form '"n-m:s"'. Here '"n"' is the starting index number which should be at least 1, '"m"' is the ending index, and '"s"' is a step value. For example,

```
-t 3
-t 1-3
-t 1-10:5
```

are all legitmate uses of -t. Like all SGE options this can be permanently embedded into the script using the '"#$"' syntax. In response to the -t option SGE adds the variables '"$SGE_TASK_FIRST"', '"$SGE_TASK_LAST"', '"$SGE_TASK_STEP"', and '"$ SGE_TASK_ID"'.

Here is a job script that uses these variables

```
#!/bin/bash

#$ -S /bin/bash

#$ -N job
#$ -t 3
#$ -e err-$JOB_ID-$TASK_ID
#$ -o out-$JOB_ID-$TASK_ID

#$ -m bea

echo this is a line of output # except for this

#echo this is another line of output > output 2> error

#echo this is yet another command > command-2 2>& error-2

# Create Working Directory
WDIR=/state/partition1/$USER/$JOB_NAME-$JOB_ID-$SGE_TASK_ID

echo $WDIR

echo \$SGE_TASK_FIRST $SGE_TASK_FIRST
echo \$SGE_TASK_LAST $SGE_TASK_LAST
echo \$SGE_TASK_STEP $SGE_TASK_STEP
```

Notice that some of these variables will be set to the value '"undefined"' if they are not set. '"$SGE_TASK_STEP"' is empty unless it is used. None of these have to be used, but as the example script shows you should use at least '"$SGE_TASK_ID"' to make your working directory into a task and job specific name. Other uses of these variables might be to select what a particular task does.

There is one other issue which comes into play. If you let SGE pick the default error and output file names, the TASK_ID for a particular task will be embedded in the name. If you override the SGE default, then you should put '"$TASK_ID"' into the name as shown above. Otherwise, each task will end up using the same output and error files, and SGE makes no guarantees about what should happen in this case. Note the two uses. '"$TASK_ID"' is used in the $# -e, and #$ -o directives, and it is read and replaced by SGE. '"$SGE_TASK_ID"' is used within your script.

Here is a example showing the strength of array jobs, with a fragment of a script that 8 calculations on the same dataset:

```
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/Project/phylip.dat .
cp $HOME/Data/Project/garli.tre .
cp $HOME/Data/Project/garli_constraints .

#Put your Science related commands here
i=0
while [ $i -lt 8 ]
do
echo starting run number $i
/share/apps/Garli0.96b8 garli_constraints
tar cf run_$i.tar *
i=$((i+1))
done


# Copy Results Back to Home Directory
RDIR=$HOME/Data/Project/Results/$JOB_NAME-$JOB_ID
mkdir -p $RDIR
cp * $RDIR
```

This script loops 8 times running a garli job each time. This means this one job will take longer, and it's also much more complex. If you wanted to update the number of loops you have to update the script, and no provision is made for the fact that you might have runs 1-8 already done, but you need a second set of 8. The second version of this script handles these issues and is much cleaner:

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

# Copy Data and Config Files
cp $HOME/Data/Project/phylip.dat .
cp $HOME/Data/Project/constraint.tre .
cp $HOME/Data/Project/garli_constrain.com .

# Put your Science related commands here
/share/apps/Garli0.96b8 garli_constrain.com

# Copy Results Back to Home Directory
RDIR=$HOME/Data/Project/Results/$JOB_NAME-$JOB_ID/Run-$SGE_TASK_ID
mkdir -p $RDIR
cp * $RDIR
```

The first thing to note is that this script did not require very much to modify from the single run version to the task array version. Really there are just a few additions to make sure a task specific working directory is created. Also, task specific results directories are created. Besides ensuring that different tasks don't overwrite each others files, this provides a mechanism for adding the number of runs. For example, if you had 10 runs, then it's easy to add 10 more, using -t 11:20 as the command line. The output will then nicely be stored into 10 new output files.

Of course, you could use the variable '"$SGE_TASK_ID"' to select multiple data sets rather than running over the same data set. One way to do this is to just have the multiple files each of which has a index number such as phylip-1.dat, phylip-2.dat, ... Then it is easy enough to use '"$SGE_TASK_ID"' to select what file the task should process. However, if your files are not labeled this way, it might be a burden to go renaming them. An alternative is to use a file like so

```
cd ~/Data/Project
ls * > INPUTS
```

Replace the '"*"' on with some other file glob that matches the input names you wish to use. The job script should then copy the INPUTS file to the local working directory. Each task will then process the file on line '"$SGE_TASK_ID"'. This can be found using the following bit of shell magic

```
file=`awk 'NR==n' n=$SGE_TASK_ID INPUTS`
```

'"$file"' can now be used in your script in any location where you would have put a data set name. Note the use of the back tick ` here. The back ticks are a shell quoting mechanism that indicates you wish the shell to replace the back ticked quantity with the output of the command contained between the tick marks.

== Summary ==

Using task arrays gives you much power in your ability to control jobs. With an array, one job number is responsible for starting multiple tasks. This means that you have only one job to monitor, and only one job to remove if a problem arises. The same guidelines apply though. You should avoid tying up all of the slots with one job, and instead run several smaller jobs with different ranges of values. In fact, this is more pressing because if your usage of the cluster is causing problems for others we might have remove jobs. In this case, removing one of your several jobs and its tasks will be less of a pain than having to remove one job that has all of your tasks.