SGE Array Jobs

** NOTE ** In the examples below, data files are accessed via the shared cluster file system. This can result in slow performance, especially when the file server is overloaded. To improve your application's performance, and also to avoid adding additional load on the file server, please modify the scripts below so that your program is using Scratch Disk Space.

Introduction

An SGE Array Job is a script that is to be run multiple times. Note that this means EXACTLY the same script is going to be run multiple times, the only difference between each run is a single environment variable, `$SGE_TASK_ID`, so your script MUST be reasonably intelligent. However, compared to submitting 100's of independent SGE jobs, Array Jobs can be more readable - there may be only one script that you have to understand (an alternative approach might have one script to write a set of batch-scripts, one script to submit those batch-jobs to SGE, and another script which performs the actual program-logic that you want to run).

An array job is started like any other, by issuing a qsub command with the '-t' option:

```
qsub -t 1-1000 myscript.q
```

This will run the script (usually csh or bash) 1000 times, first with $SGE_TASK_ID=1, then with $SGE_TASK_ID=2, etc. Again, it is up to your script to do something different in each case ... perhaps just cd into a different directory, process inputfile.$SGE_TASK_ID, etc. If, for some reason, you want to process every Nth number in the sequence, you can use, e.g., "qsub -t 1-1000:4" to do every 4th number (1 .. 5 .. 9).

One of the easiest ways to use an array task is to pre-compute or set-up N different input files, or input directories if more than one input file is needed. Let's say files inputA and inputB are needed by the program. We could create 100 directories, dir.1, dir.2, through dir.100 and put different input data into each directory. Then the script might look like:

```
#!/bin/csh
# : call this file 'example1.q'
cd dir.$SGE_TASK_ID
run_my_program
```

then we do "qsub -t 1-100 example1.q" to start all 100 jobs. SGE will start as many of the individual tasks as it can, as soon as it can.

Note that you can also embed the "-t 1-100" into the .q file:

```
#!/bin/csh
#
#$ -cwd -m b,e
#$ -t 1-100
cd dir.$SGE_TASK_ID
run_my_program
```

With the '-t' embedded in the file, you can submit it as "qsub example.q".

Environment Variables

There are a few environment variables that SGE sets during array tasks: $SGE_TASK_FIRST, $SGE_TASK_LAST, $SGE_STEP_SIZE. So the following script may be useful if you need certain things to happen at the start or end of the job:

```
#!/bin/csh
#
#$ -cwd -m b,e
#$ -t 1-100
if( $SGE_TASK_ID == $SGE_TASK_FIRST ) then
   # do first-task stuff here
endif
# do normal processing here
if( $SGE_TASK_ID == $SGE_TASK_LAST ) then
  # do last-task stuff here
endif
```

NOTE: remember that SGE may start multiple jobs simultaneously! So you probably shouldn't expect the "do first-task stuff" code to do on-the-fly initializations - other jobs may already be running when that "first-task stuff" is starting! Do NOT expect the first task to build files or directories for the other tasks to use.

Similarly, the last task may NOT be the last one to complete - it is simply the last one to be started. Do NOT expect the last task to clean up after the other tasks - they may still be running! If the last task deletes a file that another task is using, it will probably crash and its output will be corrupted.

The first task could be used to write a temporary file so you know that it has started. The last task could be used to submit a new job to SGE ... but ONLY IF you make that second job dependent on the current one (i.e. force the second job to wait until SGE knows that all current tasks are complete) - see [SGE Job Dependencies] for more info.

See SGE Env Vars for more information on accessing environment variables within other programming environments, e.g. Perl and Matlab.

Array Jobs vs. Multiple (Identical) Job Submissions

- Submitting multiple (identical) jobs works fine
  - You can easily write a shell script to submit 100's of jobs with different input files, etc.
  - Using an Array Job (with 100's of sub-jobs) may be easier to read since all of the logic is contained within one file
- If you've submitted multiple (identical) jobs, then you can delete or cancel individual jobs without interrupting any of the others
  - E.g. you may realize, after hitting 'qsub', that job-12345 is going to produce bad results ... so you can kill job-12345 and job-12346 will still run to completion
  - If you submit one Array Job, you can delete or cancel "ALL" sub-jobs with 'qdel JOBID'
  - To kill only one sub-task out of an array job, you can use 'qdel JOBID.TASKID'
- There is no performance difference for Array Jobs vs. Multiple Jobs
  - Multiple Array Job tasks will be run simultaneously if enough machines are available ... i.e. your one Array Job "will" run in parallel
  - There is no preference given to Array Jobs vs. Multiple Jobs

Using tsub instead of qsub for low-priority Array jobs

A tsub example:

```
[head4 ~]$ /opt/apps/bin/tsub myLowPrioArrayjob.q
Checking for free machine groups ...
qsub -q *@machinegroup1-n*,*@machinegroup2-n*,*@machinegroup3-n*,*@machinegroup4-n*, ... myLowPrioArrayjob.q
Your job 3168 ("myLowPrioArrayjob.q") has been submitted
```

Although the SGE scheduler can usually avoid preempting a low-priority job on a given processor core (or "slot"), the possibility of at least one job being preempted (actually, partially suspended) becomes increasing likely when the running large array jobs. An unfortunate side effect is that some jobs make take 10x (or even 20x) longer to finish than the others. One way to avoid this scenario is to use the SGE "-q" directive to direct the job to specific machine groups (see the "Requesting certain machines" section of [Submitting Single-CPU Jobs]) on which no high-priority jobs are running at the moment and therefore, presumably, will not be launched until the low priority job completes. To facilitate this, we have written the "tsub" wrapper for qsub which does the following:

1. Parses the output of the command "qconf -shgrp @8cores" to generate a list of machine groups with the newest, fastest nodes in the DSCR (8-core minimum).

2. Queries qstat for each group to see if any of the nodes are running high-priority jobs. The names of machine groups not actively running high-priority jobs are appended (in the wildcard form above) to the "-q" list.

3. Prints the complete "-q" list to the screen for reference and invokes qsub with the "-q" directive and the batch script.

Note that tsub is only intended for low-priority jobs. If a batch script contains an "-l highprio" directive, then the job will be routed to the active group to which the user has high priority access and the "-q" directive will be ignored.