Using Job Arrays on Hydra (slides)

In the intro portion of the workshop you will learn:

- What are jobs arrays, when and why use them?
- How to write job arrays scripts.
- How to submit jobs arrays:
  - task range, increment and limit concurrent tasks.
- Job arrays tips and tricks.
- Parallel job arrays.
- How to consolidate small tasks in job arrays.
- How to manage job arrays: `qstat[+]`, `qdel`, `qacct[+]`.
- HPC wiki on job arrays:
- https://confluence.si.edu/display/HPC/Job+Arrays

Introduction

What are jobs arrays, when and why use them?

- Job arrays allow you to run the same job file multiple times in a single job submission.
- They are typically used for running a given analysis on different input files or parameters.
- They allow you to use the same job file and a single `qsub` to run a type of analysis instead of writing a myriad of very similar job files.
- Job arrays have each a unique job id with multiple task ids

How to write job arrays scripts

A trivial example

- Job array scripts, or job files, are like any other job file, except that they have a task identifier stored in the variable `SGE_TASK_ID`
- Example:

```
echo + `date` $JOB_NAME started on $HOSTNAME in $QUEUE \
 with jobID=$JOB_ID and taskID=$SGE_TASK_ID
model < model.$SGE_TASK_ID.inp
echo = `date` $JOB_NAME for taskID=$SGE_TASK_ID done.
```

this example runs `model` using the input file `model.N.inp`

How to submit jobs arrays

That trivial example can be queued on 100 tasks with

```
qsub -t 1-100 trivial_example.job
```

this queues one job with 100 tasks, or the equivalent of 100 job files with

```
model < model.1.inp
```

in `test1.job`

```
model < model.2.inp
```

in `test2.job`, etc..., up to

```
model < model.100.inp
```

in `test100.job` - hence one job file instead of 100.

this assumes that you have 100 input files called `model.1.inp`, `model.2.inp`, ... , `model.100.inp`

A more complete job array file

task range and limit concurrent tasks: csh syntax

```
# /bin/csh
#
#$ -N model-100 -cwd -j y -o model.$TASK_ID.log
#$ -t 1-1000 -tc 100
#
echo + `date` $JOB_NAME started on $HOSTNAME in $QUEUE \
 with jobID=$JOB_ID and taskID=$SGE_TASK_ID
#
set INPUT  = model.$SGE_TASK_ID.inp
set OUTPUT = model.$SGE_TASK_ID.out
./model < $INPUT > $OUTPUT
#
echo = `date` $JOB_NAME for taskID=$SGE_TASK_ID done.
```

task range and limit concurrent tasks: sh syntax

```
# /bin/sh
#
#$ -N model-100 -cwd -j y -o model.$TASK_ID.log
#$ -t 1-1000 -tc 100
#
echo + `date` $JOB_NAME started on $HOSTNAME in $QUEUE \
 with jobID=$JOB_ID and taskID=$SGE_TASK_ID
#
INPUT=model.$SGE_TASK_ID.inp
OUTPUT=model.$SGE_TASK_ID.out
./model < $INPUT > $OUTPUT
#
echo = `date` $JOB_NAME for taskID=$SGE_TASK_ID done.
```

Note

- Task range and max concurrent task embedded in the script

```
-t 1-1000 -tc 100
```

- Different log file and output file for each task

```
-o model.$TASK_ID.log
INPUT=model.$SGE_TASK_ID.inp
OUTPUT=model.$SGE_TASK_ID.out
./model < $INPUT > $OUTPUT
```

- Use `$TASK_ID` in embedded `-o` directive vs `$SGE_TASK_ID` in the script

Job arrays tips and tricks

Various ways of using the task id `$SGE_TASK_ID`

1. formatting
2. using `awk`
3. using `sed`
4. using `bc`
5. using `cd`
6. using `<<EOF`
7. using your own tool

Formatting: replacing 1,2,...,100 by 001,002,...,100

- csh syntax

```
@ i = $SGE_TASK_ID
set I = `echo $i | awk '{printf "%3.3d", $1}'`
```

- sh syntax

```
let i=$SGE_TASK_ID
I=$(echo $i | awk '{printf "%3.3d", $1}')
```

Using awk to extract parameters from a single file

- csh syntax

```
@ i = $SGE_TASK_ID
set P = (`awk "NR==$i" parameters-list.txt`)
```

- sh syntax

```
let i=$SGE_TASK_ID
P=$(awk "NR==$i" parameters-list.txt)
```

the variable P will hold the content of the i-th line of parameters-list.txt, and can be used as:

```
./compute $P
```

assuming compute takes parameters.

Using sed and a template

- csh

```
@ i = $SGE_TASK_ID
sed "s/NNN/$i/" input-template.inp > model.$i.inp
model < model.$i.inp > model.$i.out
```

- sh

```
let i=$SGE_TASK_ID
sed "s/NNN/$i/" input-template.inp > model.$i.inp
model < model.$i.inp > model.$i.out
```

replace NNN in the template by the task id

Using bc to run models on temperatures

- run on tp starting at 23.72 and increasing by 2.43 increments
- csh

```
@ i = $SGE_TASK_ID
set tp = `echo "23.72 + $i*2.43" | bc`
sed "s/TP/$tp/" input-template.inp > model.$i.inp
model < model.$i.inp > model.$i.out
```

- sh

```
let i=$SGE_TASK_ID
tp=$(echo "23.72 + $i*2.43" | bc)
sed "s/TP/$tp/" input-template.inp > model.$i.inp
model < model.$i.inp > model.$i.out
```

replace TP in the template by the computed temperature stored in $tp

Using cd and different directotories for each task

- csh

```
@ i = $SGE_TASK_ID
cd task.$i
model < model.inp > model.out
```

- sh

```
let i=$SGE_TASK_ID
cd task.$i
model < model.inp > model.out
```

assumes there is a `model.inp` file in each `task.N` directory

Using the <<EOF construct

- csh

```
@ i = $SGE_TASK_ID
set tp = `echo "23.72 + $i*2.43" | bc`
model <<EOF > model.$.out
$tp
EOF
```

- sh

```
let i=$SGE_TASK_ID
tp=$(echo "23.72 + $i*2.43" | bc)
model <<EOF > model.$.out
$tp
EOF
```

Using your own tool, `mytool`, to convert a task id to parameters

- csh

```
@ i = $SGE_TASK_ID
set P = (`./mytool $i`)
./compute $P
```

- sh

```
i=$SGE_TASK_ID
P=$(./mytool $i)
./compute $P
```

How to consolidate small tasks in job arrays.

Why

- Each task is started like a job, hence has the same overhead as starting one job
- Users should avoid running lots of very short tasks ($<$ 10-30m)
- It is relatively easy to consolidate short tasks into longer ones, using the task increment:
  - `qsub -t 200-500:20` will run tasks with id=200,220,240,...,500

How

- use the variables:

`$SGE_TASK_FIRST`
`$SGE_TASK_LAST`
`$SGE_TASK_STEPSIZE`
`$SGE_TASK_ID`

Example to consolidate short tasks: csh syntax

```
@ iFr = $SGE_TASK_ID
@ iTo = $iFr + $SGE_TASK_STEPSIZE - 1
if ($iTo > $SGE_TASK_LAST) @ iTo = $SGE_TASK_LAST
#
echo running model.csh for taskIDs $iFr to $iTo
@ i = $iFr
while ($i <= $iTo)
  ./model.csh $i >& model-$i.log
  @ i++
end
```

Example to consolidate short tasks: `sh` syntax

```
let iFr=$SGE_TASK_ID
let iTo=$iFr+$SGE_TASK_STEPSIZE-1
if [ $iTo -gt $SGE_TASK_LAST ]
then
  let iTo=$SGE_TASK_LAST
fi
#
echo running model.csh for taskIDs $iFr to $iTo
let i=$iFr
while [ $i -ge ]<= $iTo ]; do
  ./model.sh $i >& model.$i.log
  let i++
done
```

Where

- the script `model.csh` or `model.sh` do the work and takes one argument: the id.

Parallel job arrays

- Job arrays can run parallel tasks
- Each task request a parallel environment, as per the `-pe` specification:
  - `-pe mthread N` for multi-threaded
  - `-pe mpich N` or `-pe orte N` for MPI

- Check the HPC wiki for more info at https://confluence.si.edu/display/HPC/Job+Arrays

How to manage job arrays: `qstat[+]`, `qdel`, `qacct[+]`.

- job status with `qstat` or `qstat+`
- job deletion with `qdel`
- job accounting with `qacct` or `qacct+`

(details missing)

Also remember

- separate name spaces: some of the tasks will run at the same time and should not write in the same file
- test on a small set of tasks first
- avoid sending emails with `-m abe`, it applies to each task (lots of emails).
- manage the results files, esp. if a lot of them are created

HPC wiki on job arrays

- https://confluence.si.edu/display/HPC/Job+Arrays

Hands on portion