DEC 2017

ICER
TICKET HIGHLIGHTS

Using Interactive Jobs to increase productivity

Patrick Bills, Research Consultant, ICER

MSU HPC users often have a work process similar to the following :

1. they test their software or code on a development node with small amounts of data, or short time periods;
2. write a submission (qsub) script to submit to the queue;
3. let the system run their programs and wait for output; then
4. check the output for errors or valid results, and usually;
5. make changes and try again.

Sometimes the testing phase can take a really long time, especially when one has to wait for a job to complete to see if it fails or not. Many times, the development nodes are busy and they can get slow. Also a single development node is not the best environment for those users who want to test multi-node (parallel worker or MPI) software.

What if you could run a job on the cluster but have full interactivity just like on the development node? You can - with an "*interactive job*"

Jobs that need to be run interactively are not just for testing, but often require a lot of user input and run a graphical user interface (GUI). Some examples of typical interactive jobs include:

- Multi-node jobs you need to test
- Matlab code that requires a user interface and needs to run for more than 2 hours
- Large-memory interactive jobs
- Work that requires both a GUI and also high amount of memory and processors
- Debugging large OpenMP or MPI jobs (e.g using TotalView GUI or the

Our development nodes are designed for much of this kind of work, but by using the scheduler, you can run longer on a compute node and excede the CPU time limits imposed on the development nodes.

Scheduling an Interactive Job

To schedule a command-line or GUI interactive job, you must provide the -I option to the qsub command. For example:

```
qsub -I -l nodes=2:ppn=4,walltime=02:00:00,mem=32 gb -N MyInteractivejob
```

Note on syntax: we recommend that qsub options appear in a single qsub script, but you can also provide those directly on the command line as you see above. The above command will seek out 2 nodes that have 4 free cores and 16 gb each available, and reserve it for 2 hours for you.

After you hit enter to the command above, you will have to wait until the scheduler can provide the resources that you requested. Once that happens, then you will be logged in to a compute node directly that has been reserved for this job. If you reserve all of the cores (28 cores in the case of Laconia/Intel16 nodes) or memory then you will have the node to yourself (but it may take a long time to schedule. If you request less than all the cores, you will share this compute node with other jobs but is still less than on a busy dev-node. >

For the example above, you will be given 2 hours to work at which point the job will be cancelled and you will be logged out. If you exit from the compute node before the time is up, the job is cancelled. You can simply re-submit the qsub, and start again.

Example commands

1) Connect to the HPCC normally from your computer (terminal, moba xterm, putty, etc). Use the "-X" command if you are using MobaXterm or XQuartz and intend to use a GUI program (e.g Stata, Matlab, Ansys), but do not include the -X if your using Mac terminal

```
ssh -X yournetid@hpcc.msu.edu   # -X is only neede for GUI jobs.
```
you will be connected to a gateway node.

2) Connect to any of the development nodes. The dev-node you select does not affect which type of compute node the interactive jobs runs on. Note that Interactive jobs do not always work from gateways (you may get the error "job apparently deleted"). Again use -X only if you are using XQuartz

```
ssh -X dev-intel14-k20
```

3) Issue the qsub command from any dev-node and issue from there. Again, if you plan on using GUI software, you need to include the "-X" option. For example, here is a qsub command asking for one node, 10 cores, and 2gb/cores.

Note if you get an error "[qsub: DISPLAY not set] " then you are not using an X-enabled terminal (e.g. Mac Terminal) and should not use this option. (Use XQuartz/Mac or MobaXterm/Windows or our Remote Desktop Connect instead if you want to use GUI software)

```
qsub -I -X -l nodes=1:ppn=10,mem=20gb,walltime=04:00:00 -N testing
```

You should, after a while, see something like this:

```
[billspat@dev-intel14-k20 ~]$ qsub -I -l nodes=1:ppn=10,mem=20gb,walltime=01:00:00
-N testing
qsub: waiting for job 50482872.mgr-04.i to start
qsub: job 50482872.mgr-04.i ready
[billspat@csp-018 ~]$
```

While inside the job, you can use all of the QSUB variables like $PBS_JOBID etc for example qstat -f $PBS_JOBID to see current statistics of the job you are inside. To see all of the PBS variables and settings you can use the command

```
env |grep PBS
```

(which shows all environment variables and filters only those that contain PBS). When you are finished using your commands and need to exit the job, use 'exit' to quit

```
[billspat@csp-018 ~]$ exit
logout
qsub: job 50482872.mgr-04.i completed
[billspat@dev-intel14-k20 ~]$
```

Note that the regular job policies are in place : keeping your job to < 4 hours will schedule more quickly as it can run on idle buy-in nodes (see https://wiki.hpcc.msu.edu/display/hpccdocs/Scheduling+Jobs)

Below is a full interactive session, where I use the R system for stats and examine aspects of parallel computing (see our R tutorial on the HPCC wiki by Nanye Long). Note I'm using R in text mode, not GUI mode (there is no -X parameter)

I hope these details help you find it easier to perform ad-hoc testing and analysis on our HPCC. As always, if you have any questions please contact us via https://contact.icer.msu.edu/contact

Example R session in an Interactive Job

```
$ qsub -I -l nodes=1:ppn=10,mem=20gb,walltime=01:00:00 -N testing
qsub: waiting for job 50482872.mgr-04.i to start
qsub: job 50482872.mgr-04.i ready
[billspat@csp-018 ~]$ # note - need to load modules in this new session
[billspat@csp-018 ~]$ module load GNU/4.9 OpenMPI R/3.3.2
```

~~~
[billspat@csp-018 ~]$ R
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

[Previously saved workspace restored]
> library(doParallel)
> Loading required package: foreach
> foreach: simple, scalable parallel programming from Revolution Analytics
> Use Revolution R for scalability, fault tolerance and more.
> http://www.revolutionanalytics.com
> Loading required package: iterators
> Loading required package: parallel
>
> # Request a single node (this uses the "multicore" functionality)
>
> registerDoParallel(cores=as.numeric(Sys.getenv("PBS_NUM_PPN")[1]))
>
> x <- iris[which(iris[,5] != "setosa"), c(1,5)]
> trials <- 10000
> ptime <- system.time({
- r <- foreach(icount(trials), .combine=cbind) %dopar% {
- ind <- sample(100, 100, replace=TRUE)
- result1 <- glm(x[ind,2]~x[ind,1], family=binomial(logit)) - coefficients(result1)
-}- })[3]

>  cat("Time elapsed:", ptime, "\n")
> Time elapsed: 4.635
> Currently registered backend: doParallelMC
> cat("Number of workers used:", getDoParWorkers(), "\n")
> Number of workers used: 10
>  q()
> Save workspace image? [y/n/c]: y

[billspat@csp-018 ~]$ qstat -f $PBS_JOBID
Job Id: 50482872.mgr-04.i
```Job_Name = testing
Job_Owner = billspat@dev-intel14-k20.i
resources_used.cput = 00:00:25
resources_used.energy_used = 0
resources_used.mem = 116460kb
resources_used.vmem = 524028kb
resources_used.walltime = 00:06:33
job_state = R
queue = main
server = mgr-04.i
Checkpoint = u
ctime = Thu Dec 14 11:20:57 2017
Error_Path = /dev/pts/0
exec_host = csp-018/0-9
~~~

~~~~
Hold_Types = n
interactive = True
Join_Path = n
Keep_Files = n
Mail_Points = a
mtime = Thu Dec 14 11:21:44 2017
Output_Path = /dev/pts/0
Priority = 0
qtime = Thu Dec 14 11:20:57 2017
Rerunable = False
Resource_List.nodes = 1:ppn=10
Resource_List.mem = 20gb
Resource_List.walltime = 01:00:00
Resource_List.nodect = 1
session_id = 59066
```

# etc... cut for brevity
# note - exit the job and return to dev node
[billspat@csp-018 ~]$ exit
logout
qsub: job 50482872.mgr-04.i completed
~~~~