Notes on Job Scheduling on Hoffman2 Cluster (draft)

Shao-Ching Huang
`schuang@idre.ucla.edu`

2020-02-26

This is an early draft. As such it may still contains typos and errors. It is provided to you in the hope that it is useful in understanding Hoffman2 job scheduling details. Please submit your feedback or suggestions or corrections to the email above. Thank you. See also: `https://github.com/schuang/hoffman2-job-scheduling-tutorial/tree/master/pdf`. (A new version of this document will be uploaded here.)

1 Introduction

Hoffman2 cluster is a major high performance computing infrastructure at UCLA, currently consisting more than one thousand computing nodes equipped with infiniband interconnect, high-performance storage system and advanced capabilities such as GPU computing and a large number of scientific packages. Hoffman2 cluster uses a shared-cluster model, as defined in [1]: most of the computing nodes are purchased by research groups using grant funding, while other compute nodes (a much smaller set) are funded by the university for general use. Users within a contributing research group have high priority access to the purchased nodes, and non-high priority access to the pool of nodes purchased by all other research groups. Currently, high priority jobs that runs on the group's purchased node(s) are guaranteed to start within 24 hours after submission, and can run as long as 14 days. General users not in any contributing research groups have (non-high priority) access to the university-funded nodes. Each job can run up to 24 hours. Pending available resources and the circumstances, it may be possible to make arrangement to reserve compute nodes for a finite period of time (e.g. for a class, a workshop or some special occasion); this normally requires lead time and is reviewed on a case by case basis.

This main body of this document focuses on explaining the scheduling policy, implications and consequences; the syntaxes of specifying the actual job parameters are available in the appendix. Reference [2] is Hoffman2 cluster's user guide. Reference [3] is Hoffman2 cluster's user support portal.

2 Job Scheduling Considerations

Hoffman2 cluster's user base spans across multiple disciplines, from social sciences to engineering and medicine and more. Users from different domains tend to have significantly different computing patterns. Some users run a large number of independent sequential jobs, or IO intensive tasks, while others run massively parallel MPI-style jobs. Hoffman2 cluster's job scheduler provides a number of parameters for users to specify different needed computing resources. In addition, due to the fact that Hoffman2 cluster has different generations of compute nodes (CPU features and speeds, purchased over the years) and different memory sizes, among other things, a user has to specify sufficient information to the job scheduler in order to allocate the proper computing resources (e.g. CPU(s) and memory size). This may add some complexity in job submission. The burden is on the users to deal with this flexibility-induced complexity. This document is an attempt to summarize and clarify the fundamental ideas behind how the current scheduler is set up and works, in the hope that it can help users to determine the optimal job parameters.

2.1 High-priority jobs

A high-priority job always runs on, and only on, a research group's purchased nodes. The job normally starts within 24 hours, assuming the group's jobs are not already occupying their purchased nodes. When a high-priority job is submitted to the scheduler queue and is detected by the scheduler (the frequency of scanning high-priority jobs is every 15 minutes), the scheduler starts to "drain" the purchased nodes, which typically are already running other non-high priority jobs by other users (e.g. from other research groups). As soon as enough (purchased) compute nodes, or CPU cores, are drained and able dispatch the incoming high priority job, the job starts (on the purchased nodes), the draining stops, and the purchased nodes are open to accept other non-high priority jobs again, until the next high-priority-job scanning cycle. As long as there are still pending high-priority jobs, the draining will continue (to prevent other non-high priority jobs from starting on the purchased nodes). The implication of this is that, as long as there are pending high priority jobs in the queue, only the first high priority job needs to wait for the draining; subsequent high priority jobs will run back to back, if not simultaneously, without waiting. Another implication of submitting a a high priority job is that, while it is guaranteed to start within 24 hours, it can only start on the purchased nodes even if many other compute nodes (purchased by other research groups) are available.

2.2 Non-high priority jobs

A non-high priority job can run essentially anywhere on the cluster, except for a number of special-purposed reserved nodes. Currently, a non-high priroity job has a time limit of 24 hours. While there is no guaranteed start time, it allows the scheduler to use its algorithm to efficiently dispatch and achieve high cluster utilization. High utilization means that, overall, more jobs will be processed within the same time frame, implying, shorter wait time, at least statistically. Submitting non-high priority job is the recommended way of submitting jobs. For jobs requiring longer than 24 hours to complete, the user can break the computations into multiple less-than-24-hour segments, and run piece by piece sequentially (also called "checkpoint" or "restart"). Hoffman2 cluster's 24-hour time limit is quite generous, considering the much shorter time limits per job in other NSF or DOE supercomputer centers.

2.3 Limit on the number of jobs

A user can have few hundreds of pending jobs into the queues; subsequent submission will be denied until enough have started (so the number of pending jobs drops under the threshold). The limit is dynamic; for example, when the cluster is less loaded, the limit is relaxed. While this may seem an inconvenience, it is a safeguard so that the scheduler is not overwhelmed by the sheer number of pending jobs (recall that the Hoffman2 cluster has hundreds of active users). In practice, for users having to submit a large number of jobs, two approaches can be considered to circumvent the limit. One is to consider using “job array” (which requires a special syntax to submit jobs). The other is to pack multiple tasks (or jobs) into one submission (e.g. by looping through multiple jobs sequentially in one submission).

2.4 Memory limits

Hoffman2 cluster imposes memory limits on jobs since each compute node has a finite amount of memory; a job using excessive amount of memory can affect the performance or even crash other jobs running on the same node. It is the user’s responsibility to specify the appropriate amount of memory per job. If the user specifies too much memory, it may be difficult for the scheduler to find available compute node to dispatch the job to. For example, when a job requires 64GB of memory to run, the scheduler cannot consider many of the computing nodes having 24GB or 48GB of memory. If the specified memory is too low, and the peak usage exceeds it, the job may be killed by the scheduler. One tricky part of specifying memory is that Hoffman2 cluster’s scheduler checks the virtual memory use, not the resident memory use (which may be closer to the actual consumed memory size). When the users cannot determine the needed memory size for a job, it is suggested to run the job in the exclusive mode. Once the job is done, use the appropriate commands to inspect the job log from the scheduler to see the peak virtual memory size reached. Use that virtual memory as a reference in subsequent submissions of the same or similar jobs.

2.5 Interactive jobs

An interactive job is initiated by the `qrsh` command (other than the `qsub` command to submit a batch job). Unlike batch jobs, which can wait in the queue for hours, users expect an interactive job to return with a few minutes. While a number of compute nodes are reserved exclusively for interactive use, their number of CPU cores and memory sizes are not infinite. There are times that interactive jobs cannot return because, for example, the user requests an exceedingly large memory size or the number of CPU cores not immediately available at the moment of running the `qrsh` command.

It is possible to run high-priority interactive jobs (on the user group’s purchased nodes, if applicable). It should be understood that normally the purchased nodes, not particularly reserved for interactive use, are most likely already running other batch jobs; the interactive request might not return immediately.

2.6 Specifying special hardware

A diversity of different CPU architectures and GPUs are available upon request. When a job does not specify special hardware features, it can be dispatched to any compute nodes. When a job specifies additional features, it will be dispatched only to compute nodes having those features. Of course, when the conflicted parameters are specified, it is possible that the job become unable to start (i.e. no matching hardware). For example, Hoffman2 cluster has both Intel and AMD CPUs. To force a job to run on the Intel CPUs, the user can specify `-l arch=intel*`. Specifying certain hardware features give the user some control where to run the job. At the same time, when the choices become too strict, the wait time may significantly increase because the number of available compute nodes may be small, if not zero.

3 Job Submission in Practice

The required parameters of submitting a job, batch or interactive, are the time limit (specified by `h_rt`), memory size (specified by `h_data`). For multithreaded (shared memory) or MPI (distributed memory) parallel jobs, the number of CPU cores also are required (specified by `-pe`). The job parameters can be passed to the job scheduler either from the command line or written in the job script. To request, for example, 8 hours of computing time and 4GB of memory for a sequential job, using the command line approach, the user would issue:

```
qsub -N $job_name -l h_rt=8:00:00,h_data=4G $job_script
```

where `$job_script` is replaced with the name of the shell script that drives the computations (e.g. `foo.sh`). The `-N` option, which specifies the job name, is optional but is helpful to label the jobs when the user has multiple of them (e.g. `foo1`, `foo2`, etc.). It should be noted that the run time environment of the job is typically different from the interactive shell the user has (e.g. in the terminal of running the `qsub` command). In addition, when a job starts, it starts from the top level directory of the user (i.e. the path defined in the environment variable `$HOME`), which may be different from the directory (e.g. containing data files and scripts) where the job is supposed to run. A user option to use is `-cwd`, which means "running from the directory where the job is submitted." Using the command line approach, the command can be:

```
qsub -N job_name -cwd -l h_rt=8:00:00,h_data=4G $job_script
```

To write all of the job parameters into the job script, see the examples Appendix. The general syntax is that the job parameters are prefixed by `#$`, which is viewed as a comment in a shell, but is recognized by the job scheduler as a job parameter. For example, the equivalent job script corresponding to the example above is:

```
#!/bin/bash
#$ -N job_name
#$ -l h_rt=8:00:00,h_data=4G
#$ -cwd
...
```

The order of the parameters does not matter, as long as they are all specified. When duplicate items appear in the same script, only the last one is effective. It is ok to split the parameters of `-l` into multiple lines in a job script, i.e.

```
#!/bin/bash
#$ -N job_name
#$ -l h_rt=8:00:00
#$ -l h_data=4G
#$ -cwd
...
```

For parallel jobs, additional parameters are required but can be specified in the same manner. See the Appendix for full examples.

4 References

- [1] https://idre.ucla.edu/resources/hpc/hoffman2-cluster
- [2] https://www.hoffman2.idre.ucla.edu/
- [3] https://support.idre.ucla.edu/helpdesk/

5 Appendix

5.1 Job script examples

Some examples are included below. More examples are available at https://gitlab.idre.ucla.edu/hoffman2/sge-job-scripts. In each example, save the script as a file, e.g. `foo.sh` then submit the job by the command `qsub foo.sh`.

5.1.1 Sequential job

```
#!/bin/bash
#$ -l h_rt=01:00:00
#$ -l h_data=2G
#$ -N job_name
#$ -cwd
#$ -o stdout.$JOB_ID
#$ -e stderr.$JOB_ID

# put your commands below, e.g.
date
hostname
```

5.1.2 Shared-memory (multithreaded) job

This type of jobs uses multiple CPU cores from the *same* compute node. The product of `h_data` and `-pe` has to be no more than a compute node's physical memory size. If the product is too large, the job may never start. Use the command `qhost` to inspect the available memory size of the compute nodes. An otherwise non-parallel program does *not* automatically become parallel by submitting it as a parallel job.

```
#!/bin/bash
#$ -l h_rt=01:00:00
#$ -l h_data=2G,exclusive
#$ -N job_name
#$ -cwd
#$ -o stdout.$JOB_ID
#$ -e stderr.$JOB_ID
#$ -pe shared 8

# put your commands below, e.g.
date
hostname
```

5.1.3 MPI distributed-memory parallel job

This type of jobs uses multiple CPU cores across multiple compute nodes. The inter-process communication typically uses MPI (Message Passing Interface) or something equivalent. The program (application) must be capable of performing such inter-process communication. An otherwise non-parallel program does *not* automatically become parallel by submitting it as a parallel job.

```
#!/bin/bash
#$ -l h_rt=01:00:00
#$ -l h_data=2G
#$ -N job_name
#$ -cwd
#$ -o stdout.$JOB_ID
#$ -e stderr.$JOB_ID
#$ -pe dc* 8

# put your commands below, e.g.
source /u/local/Modules/default/init/modules.sh
module load intel/18.0.3
date
which mpirun
mpirun -n $NSLOTS hostname
```