HPC at Warwick and Beyond
Section 2 - Running Jobs

"The Angry Penguin", used under creative commons licence from Swantje Hess and Jannis Pohlmann.

Warwick RSE

Updated for 2020

Queues and Jobs

Login Nodes

- When you login to Godzilla or a Cluster machine you're on a login node
- These can be used for (non-exhaustive list):
  - writing and compiling code
  - submitting and monitoring jobs
  - ***with caution -*** quick single core stuff
    - viewing a data file
    - checking code will start (e.g. all modules are loaded)
  - zipping or tarring files, copying files on/off the cluster machines

When you login to Orac/Tinis, there's a reminder that you must not run jobs on the login node, as well as reminders of links to the mailing list and wiki. Generally, you should always read the sign-on messages. They can be important!

IMPORTANT!

- Login nodes, including Godzilla, ***MUST NOT be used to run code!!!***

Managing Jobs

- The Cluster machines often have dozens of people each asking for hundreds of cores
  - far more than machine has available
  - have to schedule jobs to avoid running too much
- Have to track per user and per group usages
- Have to try and schedule “fairly”
  - This is actually a really hard problem. Please don’t file bugs unless your jobs just aren’t getting run

Orac and Tinis are kept 85-95% full of running jobs and nearly always have plenty in the queue waiting.
Point 3, “running too much” means overloading a processor so that it has to work on one job, then swap to another, to the point where running the jobs at the same time takes longer than running them one after another. If you’re interested, look up Context Switching.
Bit more about fairshare later - for instance it is proportional to departmental subscription.

Queues

- We use Slurm scheduling system
- You submit a job to the system, asking for a given amount of:
  - time
  - processors and/or nodes (if using Hybrid)
  - memory per core
- Your job enters the queue
- Assuming you have hours to spend, your job runs when resources are available
- Sometimes you have to specify a queue (e.g. devel)

Cluster is now pure Slurm.
Devel queue is special - 2 nodes max, 1hr runtime, max 2 simultaneous jobs per user. Gets high priority, intended for quick running while developing. MUST NOT be used for running normal jobs. See https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/ClusterUserGuide#Testing_and_development_jobs

Modules

- Unlike starting a job directly, a queued job may not know about the environment when it was submitted
- This means any modules your job needs have to be set in the job script
- For more on modules see https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/ClusterUserGuide#Software_environment_and_modules

Note that jobs do know the directory they were submitted from, and might know more. If you have a lot of default modules, you might want to add a `module purge` before the `module load` commands in your job scripts, to avoid unexpected module collisions etc

ASIDE - code snippets

- Slides will show some snippets of code
- Usually put them in back-ticks, e.g. `code`
- Going to use Linux man syntax
- `command [-n ***option***]`
  - [] denote an optional piece of command
  - text in ***italics*** should be replaced with required value(s) (using bold italics for ease of viewing)

E.g.
ls [-l] [--sort=***word***]
List files, in long format (-l), sorting by ***word,*** e.g. --sort=time

Running Jobs with Slurm

6 Step Guide - MPI

1. Compile/Unpack Code and setup any required input files
2. Create Batch script
    1. Start with example script
    2. Modify nodes and processors-per-node required
    3. Set Memory needed per core
    4. Modify walltime
3. Add line to script to load any required modules
4. Add the line to actually run the job, with any inputs needed (pipes etc)
5. Submit the job
6. Wait…

Example scripts at https://warwick.ac.uk/research/rtp/sc/rse/training/hpcbeyond (probably where you got these notes)

Demo

1. Note that the only significant difference between Orac and Tinis here is the available cores per node (28 vs 16), and the max memory. See https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/ClusterUserGuide#Resource_limits for details

2. Walltime format is dd:hh:mm:ss. You can omit larger units if they're zero, so if you put something like 1:30 this is interpreted as 1 minute, 30 seconds

3. Output and Error (stdout and stderr) are written to a file slurm-<job-id>.out in the directory you submit from

7 Step Guide - OpenMP

1. Compile/Unpack Code and setup any required input files
2. Create Batch script
    1. Grab example script
    2. Set nodes=1 and processors-per-node to required threads
    3. Set Memory needed per core
    4. Modify walltime
3. Add line to load any required modules
4. Set the number of OMP threads
5. Add the line to actually run the job, with any inputs needed (pipes etc)
6. Submit the job
7. Wait…

Note that you rarely want to set more processors per node than there are available

Demo

1. Note that this can use only one node!
2. If you're using a node, nobody else can, so ideally you'll use all the processors
3. The main exception to that is when you need lots of memory - you might run as few as one task on a node, but use all of the memory. Alternately, this is what the Tinis FAT nodes are for

7 Step Guide - Hybrid

1. Compile/Unpack Code and setup any required input files
2. Create Batch script
    1. Grab example script
    2. Modify nodes and processors-per-node required
    3. Set Memory needed per core
    4. Modify walltime
3. Add line to load any required modules
4. Set the number of OMP threads
5. Add the line to actually run the job, with any inputs needed (pipes etc)
6. Submit the job
7. Wait…

Demo

1. It’s quite common to use one MPI task per node, and one OMP thread per core. This is usually most efficient, but isn’t necessary

2. There's more than one way to specify the number of cores, nodes etc - you may want to refer to Slurm docs and cluster docs for details

6 Step Guide - GPUs

1. Compile/Unpack Code and setup any required input files
    1. Load CUDA module
    2. Compile with nvcc
2. Create Batch script
    1. Grab example script
    2. Set tasks per node and cpus per task
    3. Select one gpu
    4. Modify walltime
3. Load any required modules
4. Add the line to actually run the job, with any inputs needed (pipes etc)
5. Submit the job
6. Wait…

On Orac, you should see: https://wiki.csc.warwick.ac.uk/twiki/bin/view/HPC/OpenPowerTestBed
[If you have existing access to Tinis - NB decomissioning in progress] If you want to use more than 1 gpu, see https://wiki.csc.warwick.ac.uk/twiki/bin/view/HPC/ClusterUserGuide#GPU_nodes for an example script. Any problems - go to Bugzilla for help.

Demo

1. NEVER try and use the GPU nodes to jump the CPU queue!

2. Tinis has 4 Tesla K80 nodes, each with 4 GPU units (2 cards each). The OpenPower TestBe has 4x NVIDIA P100

Intel MKL

BLAS and LAPACK

- Commonly used linear algebra libraries
- The Intel compiler suite on the clusters provides the Intel Math Kernel Library (IMKL)
- Heavily optimised implementation of BLAS, LAPACK and more
- You have to explicitly link this
- See https://wiki.csc.warwick.ac.uk/twiki/bin/view/HPC/ClusterUserGuide#Numerical_libraries
- Need to be cautious - default build is parallel
  - If your code is also parallel, you'll oversubscribe the nodes. See link above

For more about the linking step of building code in general, see https://en.wikipedia.org/wiki/Linker_%28computing%29#Overview or https://www.airs.com/blog/archives/38

A brief explanation of the thing to watch for: a code which uses IMKL and is run in parallel starts some number of threads on each node, usually one per processor. These can be separate MPI processors, OpenMP threads or some combination. Each core has one thread to work on. IMKL by default will start one thread per available processor, each time it's loaded by one of your threads. This can quickly run out of control and grind to a halt.

Checking on your Jobs

This section shows the Slurm commands first. For completeness we show the Moab options in these notes, in particular checkjob, which has not direct Slurm equivalent. Watch the mailing lists for the continuing changeover.

Squeue

- Can see full list of all running, waiting and blocked jobs with `squeue`
- Can see just your jobs with `squeue -u ***usercode***`
- By default shows the partition (“queue”), State (Running, PenDing, Blocked), and in final column a reason (if Pending) or the nodes in use (if Running)

Reasons include: Resources (cluster cannot run your job right now), Priority (you can't run more jobs right now - e.g. you're using all the CPUs you're allowed to at once), Dependency (you said this job must follow another).

Stopping Jobs

Scancel

1. Cancel a job (remove from queue, or stop if running) with `scancel ***jobid***`
2. You can't cancel somebody else's job as you don't have permission, so don't worry
    1. If you do have permission, congratulations on your new sysadmin job
        1. Get back to work!

Interactive Jobs

Interactive Running

- For heavy data analysis, or small test runs, can use interactive mode
- You request a number of processors for a fixed amount of time and wait until they're available
- Once they're ready, you can run things
- Time starts when resource is ready, not when you start using it

As always, see the wiki for details https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/ClusterUserGuide#Interactive_jobs

Demo

1. Two ways to do this, either for one processor for heavy serial work, or multiple for parallel work

2. It's a good idea to check the queue before submitting a request, so you know whether resources are available

3. Use Slurm directly with salloc. The resources are put aside for us and we invoke srun on login node - this then uses what we've been given

4. Finish up from salloc with `exit`

Follow instructions at https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/ClusterUserGuide#Interactive_jobs for details
Also remember that serial running ties up an entire node even if you only use one processor

Fairshare and Quotas

Compute Hours

- Several departments contributed to purchase of cluster machines
- These generally get priority in running
- Other access can be bought - talk to your PI and see https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/ClusterUserGuide#Job_prioritisation_and_shares
- Once hours run out, your jobs drop down the priority list

Note that low priority jobs might run fine when there's not much on the machines. When they're heavily loaded however, they might not run at all. As far as I (HR) know, the scheduling system doesn't care how long something has been queued, although two jobs of equal priority often schedule first-come-first-serve

Fairshare

- Making sure everybody has “fair” access to compute is very hard
- Clusters use (relatively) simple system
  - User has fractional share corresponding to what their group/department has paid for
  - When jobs run, fairshare quotient drops according to job size
  - Fairshare quotient ticks back up over time
- Machines try to stay as full as possible - large jobs usually wait longer than smaller ones

The details of the share calculation are not simple - lots of effort has gone and still goes into working out the ‘best’ systems. It’s also not easy to work out what the results of a set of scheduling rules should be.
E.g. when the machines aren’t full anything not blocked will schedule quickly; as they fill up groups with larger shares will have higher priority and occupy more of the machines

Disk Quotas - Desktop

- Storage available on Home (fully backed up) and Storage
- See https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/StorageAndBackup
- Check usage at https://freecentre.csc.warwick.ac.uk/quota/
- If you run out of Home space you'll be unable to log in!

Disk Quotas - Clusters

- There is a home for the clusters, but this is different to the Desktop home, and different on Orac and Tinis
- See https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/ClusterUserGuide#Files_and_storage for details
- Check your quota with `mmlsquota --block-size auto`
- Note there is no backup! This is meant to be working space for code runs.
- See the link above for how to access the Desktop home from the cluster login nodes

Walltime

- There is a 48 hour limit for runtime on Warwick machines
- Bugzilla requests for a longer walltime are unlikely to be granted without very, very good reason
- If using a proprietary code, read its docs for ‘checkpointing’ or ‘restart’ capability
- If writing your own code, try and build in a mechanism for restarting code from a file
  - RSE forum might be able to help

Almost all clusters have such a limit, commonly anywhere from 24 to 72 hours. Scheduling is almost impossible without it. Some have special long queues, or a way to apply for a longer runtime - see their documentation for details.

Other Essentials

“Serial” jobs

- Clusters aren’t designed to run single-core jobs
  - Always get a whole node
- If you have multiple single-core jobs can run them all at once
- Recommend GNU parallel to manage this at the moment
- Details at https://wiki.csc.warwick.ac.uk/twiki/bin/view/HPC/ClusterUserGuide#Serial_jobs
- Or see https://warwick.ac.uk/rse/training/parallelismprimer

If you’ve heard of “Job Arrays” where you start a number of related serial jobs, those aren’t currently recommended on the Warwick systems.

Wrap Up

Summary

- NEVER run code on login nodes
  - Use interactive jobs even for heavy compilation loads
- Cluster jobs use queue system. Scripts are slightly different for MPI, OMP and hybrid jobs but always specify:
  - number of processors
  - memory
  - walltime
- Fairshare really is the fairest way to run things, honestly!

Useful Links

- For recap of basic Linux commands: https://warwick.ac.uk/research/rtp/sc/rse/training/linuxdesktop/
- For everything you need to know about the clusters: https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/ClusterUserGuide
- If you have problems you can't work out: https://bugzilla.csc.warwick.ac.uk/bugzilla/