Clusters, Queues and Modules

Note: as we discussed in the videos, different systems can use different queuing systems and module systems. This means we can't give detailed instructions, so this set of suggestions focus on guiding you through finding out what you need to know. Consult the documentation for the system you're using, or use your favourite search engine, to fill in the details.

If you are already familiar with clusters, you may wish to skip some of the steps here - DO NOT skip the parts about reading the documentation and verifying what you may and may not use the system for! ***This is very important to know!***

01 Getting on a System

- Obviously, before you can experiment with clusters, you need to have access to one!
- At Warwick we have local clusters provided by the Scientific Computing RTP. Your institute may have something similar, or there are various local and national facilities.

Info:

ALWAYS ALWAYS ALWAYS read the documentation. Read what you have agreed to when you sign up! If you're asked to sign up for mailing lists or a bug tracker, do so!

When you log-on to a machine, there may be a "Message-of-the-day" - do read this! Pay attention if you're told about standing maintenance windows or other downtime.

Try:

Log on to whatever facility you have signed up for and make sure you have access. You might need to use SSH keys.

Find out how to access files on this system - is it linked to other places you store files?

Find out how to transfer files on and off, and check if there are limitations on data rates or volumes when doing so - many cloud facilities limit the amount of data per day on or off and you may incur unexpected charges if not careful!

- 02 Where to do things

- Most big systems have two (or more) sorts of *node* (individual computer) . Often there is one sort which you use to log in, edit files and do normal tasks, and another sort where you may run jobs. Find out, and make sure to follow the rules. ***Failing to follow the rules may get your account suspended, or worse!***

Info:

Again, it is important to make yourself familiar with the documentation and to pay attention to restrictions. These are designed to make things fair for everybody, and to limit problems that can arise from overloading machines etc

Try:

Check the docs for your facility - are there login nodes? Are there special nodes for anything else? Some places have dedicated options for data visualisation (a task which isn't terribly heavy averaged over time, but can have very "bursty" requirements) or the like.

Find out which queuing system your facility uses, and have a look at the basics of using it.

Find out how your system manages resources. For instance, should you request compute cores and memory independently? Some places combine the two in their "billing"

How does "billing" work? Is this a fair-share system? Or will you be charged in some way for time used?

What are you **allowed** to do on the nodes "outside the queue"? Can you compile code there? Can you visualise your data? Often the restrictions on this are that you may *as long as it does not impact other users.* For instance, plotting a simple image may be acceptable, but a visualisation process that needs significant compute resource may not be.

03 Running a simple job

- First we are going to run the simplest job we can, just to get the hang of the queue submission

Info:

Before submitting this job, make sure your facility is OK with single-processor (serial) jobs. This one will be very short, but some places are designed for parallel jobs ONLY. If this is the case, you will want to skip to the parallel job.

Also, make sure if you need to specify a project, grant, or anything else to have your job run properly.

Try:

Write a very simple code in your language of choice - for instance, a Hello-World code. Compile this if necessary (paying attention to what you found out in Step 2 about where to do this). Have the job print something to the screen - we'll see below what this does.

Put together a submission script. You may need to load some modules even for this simple code. You will want only one core, and some small amount of memory.

Submit the job, and then quickly look at the queue. Is your job there? Is it waiting? Take a note of the job ID. Watch the queue until the job runs. If it doesn't run, try step .. below.

Now find out where output ends up on your facility. Often, a file will be created containing what would be printed to screen. This usually has a name containing the job id (for instance, slurm-12345.out). Sometimes you can control this naming in the submission script. Verify that your code ran properly and you can find its output.

Find out how to cancel a job. You will probably need to know this sooner or later.

04 Running a parallel job

- Next, we want to run a basic parallel job. There are many ways to do this, depending on the sort of job.

- To get us started, we are going to run several identical jobs in parallel, by starting several copies of our code. This is *usually NOT what you want to do*, but we're just practicing!
- We might need to load some modules in our submission script, depending on what the code does. For instance, I have to load GCC before I can load Python, and I have to make sure to load a Python-3 distro.

Info:

We are going to assume there are no restrictions on job specification. Again, it is up to you to check for max or min core numbers and any other restrictions on what you may request, and to adapt the submission accordingly

Try:

Modify your Hello World program to do pick a random number. Use a fixed seed, so you get the same number every time.

Write a basic submission script asking for N cores. For the run part, ask the scheduler to start N copies of your code (note you may not have to do anything special - this may happen by default)

Submit the job and check the output - you should see N results. Note that they may be jumbled together.

Extra - if you can, find a way to get each job to give a reliably different answer. You could use the process id (Python os.getpid()). We tried a common trick of using the time - will this work? Try out the code given in the pack on github, called **pythonRandom.py** to see

- 05 Running a real parallel job

- Now we can run some "real" parallel jobs
- For simplicity, we have provided some code examples. If you would prefer to use something else, you may have to alter the instructions a little

Info:

We have provided a couple of code examples to get you started. We have a Python based MPI code, and a Python code using numba based threading.

You will need the Numba, Numpy and MPI4Py Python packages to run these

Try:

If you have a multi-core machine at hand which you can run mpi jobs on, try running the code **pythonParallel.py** locally, just to see how it works. It uses a simple trial-division method to find prime numbers between two bounds. What package(s) do you need for it? What inputs does it need? Find an input that runs in say 1-2 minutes on 4 cores. Note that we need to invoke 4 copies of this program and they will communicate with each other - so we do something like `*mpiexec -n 4 python3 pythonParallel.py*` We use a worker-controller model for the parallelism

Put together a submission script to run for say 5 minutes on 4 cores, and submit this job. Does it work? Do you get the same output?

What happens if you run this code (submit to the queue) on 8 processors instead? Is it faster? How can you find out the run time?

Now we'll look at another kind of parallellism, where we start one process, but allow it to use more than one processor internally. We use a simple Numba based parallel loop in the code **pythonNumbaParallel.py**

Try running this code on your local machine. How many processors does it use? You will probably find it uses all of them - you can control this by setting an environment variable (export NUMBA_NUM_THREADS=4) before running, or by modifying the program itself

Put together a submission script to run for 5 minutes, with access to 4 processors, starting one copy of the program. (In slurm, this is 1 task, with 4 processors per task). You might need to include that export command too - if so it is best to set it equal to the number of processors per task - see your scheduler's docs for how

Try giving the code 8 processors to use. Is it faster?

We'll look at Numba in more detail in one of the Skills modules, if you want to know more

Note: In both of these cases, speedup may only occur for sufficiently long-running or large jobs. For instance, if you have only 4 separable "tasks", you won't be able to make use of 8 processors at all

- Addendum - Troubleshooting!

    You're quite likely to run into problems at some point or another. Here are some causes that might be relevant. Remember - the administrators for clusters are supporting tens or hundreds of people - be polite and patient, and do as much of the troubleshooting yourself as you possibly can!

My jobs don't run

- If your job is in the queue, but doesn't seem to be starting, here are some possible causes
    - Your quota has run out. On some projects there may be a total amount you can run, and you may have used it up
    - You haven't waited long enough. If other jobs are running but yours aren't, you might just have to wait longer
    - You've asked for unavailable resources. For instance, if you ask for more memory than the system can give, it often rejects the submission, but if it doesn't, the job will never be able to run
    - You've asked for hotly demanded resources, and you haven't waited long enough
    - The system is being "drained" for maintenance or testing, so no jobs are being started - once again, you'll probably have to wait
- If your submissions are failing, here are some possible causes
    - Your script is malformed. For instance, there is a syntax error (typo)
    - Your script is valid, but incorrect. For instance, you're asking for more memory than the system has, or for a resource that isn't present
    - You're trying to submit to a queue/partition/machine that you aren't authorised to, or that has run out or ended
    - You've been suspended for some reason. Check your emails etc as you should have been told why
- If your jobs queue, then "disappear" again, here are some possible causes

- Usually, this means the job runs, but fails somehow. Check for error files that can tell you if
  - Something is wrong with your script. For instance, a malformed run command, or a package that cannot be loaded
  - Something is wrong with your code. For instance, your executable is missing, or an input file at the very start can't be found
- Sometimes this can happen because the job was cancelled. If there are no output files what-so-ever check these things
  - If a sys-admin did it, they are likely to have contacted you. Check your email etc to see
  - Some times other people have permission to cancel your jobs, such as a supervisor. Hopefully they would tell you
- If there are no files at all, it is possible the job still ran but for some reason gave no output - see job failure below

My jobs fail

- If your job runs, but "fails" here are some possible causes
  - Your script is incorrect. For instance, the run command is malformed, or you have asked for so little memory that your job can't start
  - Your executable is missing or buggy
  - You are missing some required modules. Check the load commands and verify if you have everything you need. For complicated cases, you might want to use an Interactive cluster job to try things out
  - You're missing input files. Check that your input files are in the correct place for the system you're using
  - You're using incompatible library versions - for instance, your code is Python3, but you've loaded Python2
  - Most of the things under "my jobs give the wrong answer" can sometimes lead to "failure" too

- If your job seems to run, but sticks/hangs or gives no output, check if
  - Code has been left in that is waiting for human input (typing at keyboard etc)
  - The code may not have finished because it ran out of walltime or otherwise was stopped and the output was never written. For instance sometimes if a file is not closed properly it can be malformed
  - The job may still be running, but taking longer than you expect. This is especially true if you have a lot of output, or you are comparing to a very powerful desktop
  - The code is broken - hanging is a common failure mode for parallel codes with certain bugs (MPI deadlocks)
  - Again, see the things under "my jobs give the wrong answer" - these can sometimes lead to infinite loops or other things where the code is running but is not making expected progress

My jobs give the wrong answer

- This is almost never the fault of the scheduler or the cluster itself, but there can be problems on one machine that are hidden on another. Some things to check include
  - You're using incompatible library versions. For instance, you use a new version of a package on your local machine, and the cluster has an older, buggy version
    - Look for a bug tracker entry or an issue report on the problem from the code maintainters. A polite enquiry (using the appropriate channel) to a sys-admin is always better received if you can point to the source of your problem
  - You're relying on "undefined behaviour" in your code. For instance, expecting variables to be set to 0 in C (not universally the case)
    - Use one of the many tools for code analysis to check things
    - Consult the code author for help
  - You're relying too heavily on specific behaviours. For instance, the exact pattern of numbers from and RNG. While technically defined, this is not something you can expect to persist between computers

- Your input files are incorrect or corrupted somehow.
    - For instance, different computer architectures store numbers differently, and if using binary input files, this can lead to problems
    - Transferring files using email etc, can add erroneous line ending characters or other encoding issues