Portfolio 5 - HPC

Sam Bowyer

2023-05-29

High Performance Computing

Introduction

Many of the experiments that we want to perform require more computational resources than our own laptops/PCs can provide. This is where High Performance Computing (HPC) comes in, allowing us to run our code on a remote server with much more computational power than our own machines. In particular, we will be using the University of Bristol's BluePebble server, the documentation for which can be found at https://www.acrc.bris.ac.uk/protected/hpc-docs/index.html (along documentation for the other HPC systems run by the University of Bristol's Advanced Computing Research Centre).

Connecting to BluePebble

First we connect to the server using ssh:

```
ssh [username]@bp1-login01b.acrc.bris.ac.uk
```

We can also do this using the `-J` flag, which allows us to connect to the server through another server, in this case, we connect first to the University's VPN so that we can access the server from outside the University's network:

```
ssh -J [username]@seis.bris.ac.uk [username]@bp1-login01b.acrc.bris.ac.uk
```

(To avoid having to type in your password every time you connect to the server, you can set up ssh keys, see https://wiki.archlinux.org/title/SSH_keys for more details.)

This will connect us to a login node and present us with output like the following:

```
Last login: Mon May 22 15:08:24 2023 from seis-shell.bris.ac.uk




*********************************************************
*** University of Bristol ACRC BluePebble             ***
*** This system is available to authorised users only ***
*********************************************************

*********************************************************
27.01.2022:

Users can now check their storage quota using the
following command:

user-quota
```

```
*********************************************************
27.10.2022:

DO NOT RUN JOBS ON THE LOGIN NODES.

JOBS RUN ON THE LOGIN NODES WILL KILLED WITHOUT NOTICE


*********************************************************

06.06.2023:

Please remember that your HPC Project Code is now
required in your job submission script.
e.g. ABCD123456

#SBATCH --account=abcd123456

Please see the email sent to all users on 03.02.2023


JOBS SUBMITTED WITHOUT A PROJECT CODE WILL BE DELETED
WITHOUT NOTICE


*********************************************************
```

On the login node we can explore and modify our filesystem, but cannot run any jobs. To run jobs we need to submit them to the queue, which we will do using the `sbatch` command.

Before we discuss how to submit jobs to the queue, it is worth noting that in order to move files between the server and your local machine, you need to use the `scp` command, which has the same syntax as the `cp` command, but with the server name and path to the file you want to copy to/from prepended to the file name. For example, to copy a file called `test.txt` from the server to your local machine, you would run:

```
scp [username]@bp1-login01b.acrc.bris.ac.uk:/path/to/test.txt ~/local/path/to/test.txt
```

and to copy a file called `test.txt` from your local machine to the server, you would run:

```
scp ~/local/path/to/test.txt [username]@bp1-login01b.acrc.bris.ac.uk:/path/to/test.txt
```

Similarly, to copy a directory, you would use the `-r` flag:

```
scp -r [username]@bp1-login01b.acrc.bris.ac.uk:/path/to/directory ~/local/path/to/directory
```

And as with `ssh`, you can use the `-J` flag to copy files through another server, for example, running the previous command through the University's VPN would look like:

```
scp -r -J [username]@seis.bris.ac.uk [username]@bp1-login01b.acrc.bris.ac.uk:/path/to/directory
     ~/local/path/to/directory
```

Jobs and Queues

Suppose we have the following python script, `script.py`, that we would like to run as a job on BluePebble:

```
import sys, time
value = int(sys.argv[1])
```

```
print(f"The square of {value} is {value**2}")
time.sleep(5)
```

(Of course in practice we would be running a much more computationally intensive script, but this will suffice for our purposes.)

In order to run this script as a job, we need to create a job submission script, which is a bash script that tells the server what resources we need to run our job and what commands to run. For example, the following job submission script, `job.sh`, will run our python script on a single core with 10MB of memory for 10 seconds:

```
#!/bin/bash
#
#SBATCH --partition=test
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=0:0:10
#SBATCH --mem=10M
#SBATCH --account=[PROJECT_CODE]

module load lang/python/anaconda

# Change into working directory
cd "${SLURM_SUBMIT_DIR}"

# Execute code
python script.py 26
```

Note in particular here that we load Python through the `lang/python/anaconda` module, and that we specify the project code that we want to run the job under using the `--account` flag in place of `[PROJECT_CODE]`.

We can then submit this job to the queue using the `sbatch` command:

```
sbatch job.sh
```

This will submit the job to the queue and return the job ID:

```
Submitted batch job 5045206
```

which we can use to check the status of the job using the `squeue` command:

```
squeue -j 5045206
```

which produces the output:

```
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           5045206      test   job.sh  dg22309  R       0:03      1 bp1-compute002
```

Here we can see that the job is running (`R`), the time that it has been running for (`0:03`), the number of nodes that it is running on (`1`), and the name of the node that it is running on (`bp1-compute002`).

We can also use the `squeue` command without the `-j` flag to see all jobs in the queue:

```
squeue
```

The first few lines of which might look like this:

```
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           5041673       cnu    sleep  vf20011  R    1:50:35      1 bp1-gpu004
           5044995       cnu HD_doubQ  px19783  R       4:39      1 bp1-gpu032
           5043184       cnu CT_doubQ  px19783  R       8:33      1 bp1-gpu031
```

```
5043166       cnu  Dyna_CT  px19783  R      14:48      1 bp1-gpu032
5042151       cnu CT_uncTD  px19783  R      19:40      1 bp1-gpu031
5019288       cnu  wgan_v2  al18709  R 1-03:22:27      1 bp1-gpu029
5024735   compute compile_  uv20102 PD       0:00      1 (Job's account not permitted
                                                         to use this partition
                                                         (compute denies default
                                                         including default))
4925154   compute    busco  tz22836 PD       0:00      1 (Resources)
4925519   compute    busco  tz22836 PD       0:00      1 (Priority)
5025375   compute     test  tk22111 PD       0:00      1 (Priority)
5042125   compute    DATA3  bh17536 PD       0:00      1 (Priority)
5003901   compute C1_noNAC  lt17359 PD       0:00      8 (Priority)
5024427   compute getAApro  uw20204 PD       0:00      1 (Priority)
```

Once the job has finished running, we can see the output of the job in the file `slurm-[JOB_ID].out`:

```
cat slurm-5045206.out
```

(The output is named `slurm-[JOB_ID].out` because BluePebble uses the Slurm Workload Manager (https://slurm.schedmd.com/overview.html).) This shows us the contents of the output file is:

```
cpu-bind=MASK - bp1-compute002, task  0  0 [198464]: mask 0x1000001 set
The square of 26 is 676
```

The first line is information from the server about the resources that were used to run the job, and the second line is the output of the python script.

It is worth noting that the submission script can be written in any language, so long as the server has the appropriate modules installed, and furthermore the submission script can be used to run any command, not just a script. For example, the following submission script, `job_2.sh`, prints some information about the job (including the working directory via the command `pwd`) and runs the `sleep` command for 10 seconds, as well as running the same python script as before.

```
#!/bin/bash
#
#
#SBATCH --partition=test
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=0:02:0
#SBATCH --mem=100M
#SBATCH --account=[PROJECT_CODE]

module load lang/python/anaconda

# Change into working directory
cd "${SLURM_SUBMIT_DIR}"

# Do some stuff

echo JOB ID: "${SLURM_JOBID}"

echo Working Directory: $(pwd)

echo Start Time: $(date)
```

```
# Execute code
python script.py 26

# Pause to give us time to see the job
sleep 10

echo End Time: $(date)
```

This can then be submitted to the queue and the output viewed as before.

```
sbatch job_2.sh
```

```
Job 5045736 submitted to partition test
```

Again, we can look inside the output file as follows:

```
cat slurm-5045736.out
```

which gives us:

```
cpu-bind=MASK - bp1-compute002, task  0  0 [199511]: mask 0x1000001 set
JOB ID: 5045736
Working Directory: /user/home/dg22309/intro2hpc
Start Time: Tue 23 May 17:29:33 BST 2023
The square of 26 is 676
End Time: Tue 23 May 17:29:48 BST 2023
```

Job Arrays

We can also submit multiple jobs at once using job arrays. We do this by specifying the range of job IDs that we want to submit as a job array using the `--array` flag in the job submission script, we can then access the job ID of the current job in the array using the `SLURM_ARRAY_TASK_ID` environment variable and provide this to our script as an argument. In our case, we'll run five jobs and each job will square a different number between 1 and 5.

```
#!/bin/bash
#
#

#SBATCH --partition=test
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=0:02:00
#SBATCH --mem-per-cpu=100M
#SBATCH --array 1-5
#SBATCH --account=[PROJECT_CODE]

module load lang/python/anaconda

# Change into working directory
cd ${SLURM_SUBMIT_DIR}

# Do some stuff
echo JOB ID: ${SLURM_JOBID}
echo SLURM ARRAY ID: ${SLURM_ARRAY_TASK_ID}
echo Working Directory: $(pwd)
echo Start Time: $(date)
# Execute code
python script.py ${SLURM_ARRAY_TASK_ID}

echo End Time: $(date)

sleep 10
```

This can then be submitted to the queue and the output viewed as before.

```
sbatch job_array.sh
```

```
Submitted batch job 5045783
```

We can then check the status of the job array using the **squeue** command:

```
squeue -j 5045783
```

which gives us:

```
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         5045783_5      test job_arra  dg22309  R       0:14      1 bp1-compute002
         5045783_4      test job_arra  dg22309  R       0:14      1 bp1-compute002
         5045783_3      test job_arra  dg22309  R       0:14      1 bp1-compute002
         5045783_2      test job_arra  dg22309  R       0:14      1 bp1-compute002
         5045783_1      test job_arra  dg22309  R       0:14      1 bp1-compute002
```

(Note that we can also use the `squeue` command with the `-u` flag to see all jobs submitted by a particular user, e.g. `squeue -u [username]`.)

Finally, we can look inside the output files and see that each job has run the script with a different argument:

```
cat slurm-5045783_1.out
```

```
cpu-bind=MASK - bp1-compute002, task  0  0 [200620]: mask 0x1000001 set
JOB ID: 5045784
SLURM ARRAY ID: 1
Working Directory: /user/home/dg22309/intro2hpc
Start Time: Tue 23 May 17:38:38 BST 2023
The square of 1 is 1
End Time: Tue 23 May 17:38:43 BST 2023
```

```
cat slurm-5045783_2.out
```

```
cpu-bind=MASK - bp1-compute002, task  0  0 [200628]: mask 0x2000002 set
JOB ID: 5045785
SLURM ARRAY ID: 2
Working Directory: /user/home/dg22309/intro2hpc
Start Time: Tue 23 May 17:38:38 BST 2023
The square of 2 is 4
End Time: Tue 23 May 17:38:43 BST 2023
```

```
cat slurm-5045783_3.out
```

```
cpu-bind=MASK - bp1-compute002, task  0  0 [200626]: mask 0x4000004 set
JOB ID: 5045786
SLURM ARRAY ID: 3
Working Directory: /user/home/dg22309/intro2hpc
Start Time: Tue 23 May 17:38:38 BST 2023
```

```
The square of 3 is 9
End Time: Tue 23 May 17:38:43 BST 2023
```

```
cat slurm-5045783_4.out
```

```
cpu-bind=MASK - bp1-compute002, task  0  0 [200622]: mask 0x8000008 set
JOB ID: 5045787
SLURM ARRAY ID: 4
Working Directory: /user/home/dg22309/intro2hpc
Start Time: Tue 23 May 17:38:38 BST 2023
The square of 4 is 16
End Time: Tue 23 May 17:38:43 BST 2023
```

```
cat slurm-5045783_5.out
```

```
cpu-bind=MASK - bp1-compute002, task  0  0 [200653]: mask 0x10000010 set
JOB ID: 5045783
SLURM ARRAY ID: 5
Working Directory: /user/home/dg22309/intro2hpc
Start Time: Tue 23 May 17:38:39 BST 2023
The square of 5 is 25
End Time: Tue 23 May 17:38:44 BST 2023
```

Automating Job Submission

In order to avoid some of the tedium involved in job submissions, we can write scripts that automate some of the work for us. Or even better, use scripts that someone else has already written, such as https://github.com/LaurenceA/infrastructure.