Portfolio Report 5: HPC

Jake Spiteri

2020

Intro to HPC

This document will cover the use of BlueCrystal Phase 3. We will discuss how to login to the HPC, send/receive files, and run jobs.

Logging in on Linux

To log in from a Linux machine, we use the built-in `ssh` client. We enter the following command into the terminal:

```
ssh -X <username>@bluecrystalp3.acrc.bris.ac.uk
```

A prompt will ask for the account password, and upon entering the password we will be given access to a login node in which we can compile and submit jobs to a standard, high-memory, or GPU node.

Modules and the environment

BlueCrystal 3 has a wide range of software packages installed, and when submitting a job it is good practice to specify the modules required to run the job. By specifying the local environment in the job submission we can ensure that the job always runs in the same environment.

To use a package we use the Environment Modules system. We can easily print all packages available by using

```
module avail
```

The (shortened) output can be seen below.

```
[js14606@newblue4 ~]$ module avail

---------------------------- /cm/local/modulefiles ----------------------------
cluster-tools/6.0          module-info
cmd                        null
dot                        shared
freeipmi/1.1.3             use.own
intel-opencl/1.2-3.0.67279 version
ipmitool/1.8.11

---------------------------- /cm/shared/modulefiles ---------------------------
acml/gcc/64/5.1.0
acml/gcc/fma4/5.1.0
acml/gcc/mp/64/5.1.0
acml/gcc/mp/fma4/5.1.0
```

```
acml/gcc-int64/64/5.1.0
acml/gcc-int64/fma4/5.1.0
acml/gcc-int64/mp/64/5.1.0
acml/gcc-int64/mp/fma4/5.1.0
acml/open64/64/5.1.0
acml/open64/fma4/5.1.0
acml/open64/mp/64/5.1.0
acml/open64/mp/fma4/5.1.0
acml/open64-int64/64/5.1.0
acml/open64-int64/fma4/5.1.0
acml/open64-int64/mp/64/5.1.0
acml/open64-int64/mp/fma4/5.1.0
languages/R-2.15.1
languages/R-2.15.1-ATLAS
languages/R-3.0.2
languages/R-3.0.2-ATLAS
languages/R-3.1.1-ATLAS
languages/R-3.2.0-ATLAS
languages/R-3.2.2-ATLAS
languages/R-3.2.4-ATLAS
languages/R-3.3.1-ATLAS
languages/R-3.3.2-ATLAS
languages/R-3.3.3-ATLAS
languages/R-3.4.1-ATLAS
languages/R-3.4.4-ATLAS-gcc-7.1.0
languages/R-3.5-ATLAS-gcc-7.1.0
languages/R-3.5.1-ATLAS-gcc-6.1
languages/R-3.6.2-gcc9.1.0
```

We see that multiple versions of R are available. In order to add R to our environment we can use `module add` as follows

```
module add languages/R-3.5.1-ATLAS-gcc-6.1
```

Similarly, we can remove a package using `module del`, and to show which modules are currently loaded we can use `module list`.

Jobs

In order to run a job we must submit a job script to the queuing system. Before we submit the job we should compile the necessary code, and move any data required to BlueCrystal using scp. The job script should contain all of the commands required to run the job, and is often written as a shell script but this is not necessary — an R script will also work. There are no limits or requirements on what the shell script does, but there are some suggestions, such as comments which tell the queuing system which resources are required for the job.

To demonstrate the queueing system and jobs we will use the `workshop` files provided on BlueCrystal. To move the .tar file to the local directory and unpack it, we use the following commands

```
cd
cp ../workshop.tar .
tar xvf workshop.tar
cd workshop
```

There are five shell scripts in the `workshop/` directory. Below we print the contents of one of them.

```
[js14606@newblue1 workshop]$ more job1.sh
```

```
#!/bin/env bash
#
#
# Define working directory
export WORK_DIR=${HOME}/workshop

# Change into working directory
cd ${WORK_DIR}

# Execute code
/bin/hostname

sleep 20
```

The script simply sleeps for 20 seconds. To submit this to the queue, we run the command `qsub job1.sh` which will return the job number. To check on the status of the job we can use `qstat <job_number>`. A useful tip is to instead use the Linux command `watch` to rerun the `qstat` command every 2 seconds by default (e.g. `watch qstat <job_number>`). Once the script has run, two output files will appear in our directory with the names `<job_script>.o<job_number>`, and `<job_script>.e<job_number>` which are the standard output and error respectively. For example, in the local directory (workshop) we have `job1.sh.o9190965` and `job1.sh.e9190965` as a result of running the `job1.sh` script.

Below are some of the queuing system commands:

- `qsub <job_script>` — Submits the specified job.
- `qstat <job_number>` — Checks the status of the specified job.
- `qdel <job_number>` — Deletes the specified job.
- `showstart <job_number>` — Provides an estimate of the start and end time of the script.
- `showq` — Shows all of the jobs in the queue.

When we submit the `job1.sh` script, the queuing system attempts to estimate the expected running time, and the number of nodes required. However it is best to specify these manually by leaving comments in the shell script for the queueing system. For example, in the workshop file `job2.sh` we specify `#PBS -l nodes=1:ppn=1,walltime=00:10:00`. `-l nodes=1:ppn=1` tells the queueing system that the script requires 1 node, with 1 processor per node, and `-l walltime=00:10:00` tells the scheduler the expected runtime is 10 minutes.

Below we submit `job1.sh` and `job2.sh` as jobs and use `qstat -au js14606` to check the status of the jobs I have submitted.

```
[js14606@newblue4 workshop]$ qsub job1.sh
9564646.master.cm.cluster
[js14606@newblue4 workshop]$ qsub job2.sh
9564648.master.cm.cluster
[js14606@newblue4 workshop]$ qstat -au js14606

master.cm.cluster:
                                                                               Req'd    Req'd       Elap
Job ID               Username    Queue    Jobname          SessID NDS   TSK    Memory   Time    S   Tim
-------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ------
9564646.master.c     js14606     veryshor job1.sh              --    --     --     --  01:00:00 Q
9564648.master.c     js14606     veryshor job2.sh              --     1      1     --  00:10:00 Q
```

We see that the scheduler recognizes that `job2.sh` requires 1 core (NDS) and 1 node per core (TSK), but it does not know the resource requirements for `job1.sh`. We also see that the required time of `job1.sh` is estimated by the scheduler to be 1 hour. Given that `job2.sh` does not require many resources, it is likely that this job may be scheduled to run first and so it is worthwhile to specify the required resources in your job script.

The queueing system also allows us to specify other job properties, such as the job name, required memory, a job array etc. When running complex jobs which require a lot of resources, a useful command to give the scheduler is `#PBS -M <email_address>` which tells the scheduler to send an email to the specified address at certain points in the job's lifecycle. By default, the scheduler will only send an email if the job is aborted, but we can specify that we want to receive an email when the job begins and ends via `-m b` and `-m e` respectively.

We implement this by creating a new file `job6.sh` which is shown below.

```
#!/bin/env bash
#
#
#PBS -l nodes=1:ppn=1,walltime=00:10:00
#PBS -m abe
#PBS -M jake.spiteri@bristol.ac.uk

# Define working directory
export WORK_DIR=${HOME}/workshop

# Change into working directory
cd ${WORK_DIR}

# Execute code
/bin/hostname


sleep 20
```

The line `#PBS -m abe` specified that we want to receive an email if the job is aborted, and when it begins and ends. Indeed, I received the following emails upon submitting the job:

```
PBS Job Id: 9564931.master.cm.cluster
Job Name:   job6.sh
Exec host:  node33-009/6
Begun execution
```

```
PBS Job Id: 9564931.master.cm.cluster
Job Name:   job6.sh
Exec host:  node33-009/6
Execution terminated
Exit_status=0
Error_Path: newblue4.cm.cluster:/newhome/js14606/workshop/job6.sh.e9564931
Output_Path: newblue4.cm.cluster:/newhome/js14606/workshop/job6.sh.o9564931
```

For more information, you can find additional PBS commands in the documentation.