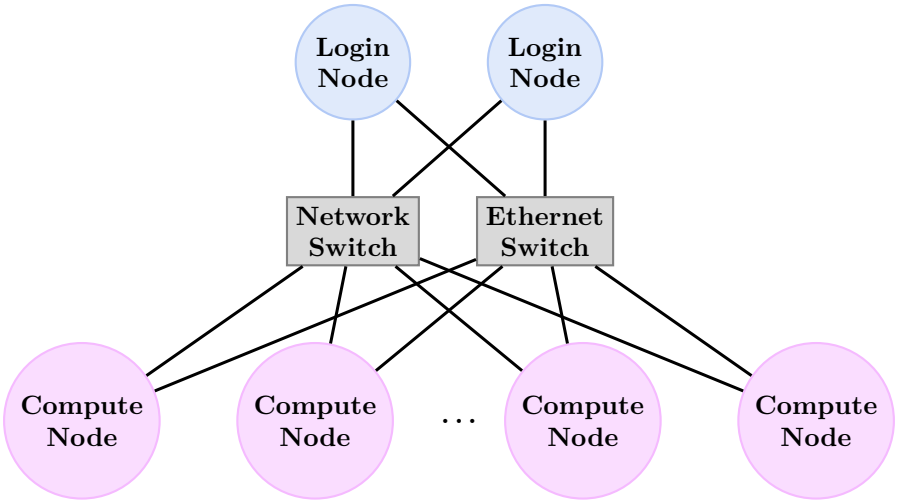
HPC Portfolio

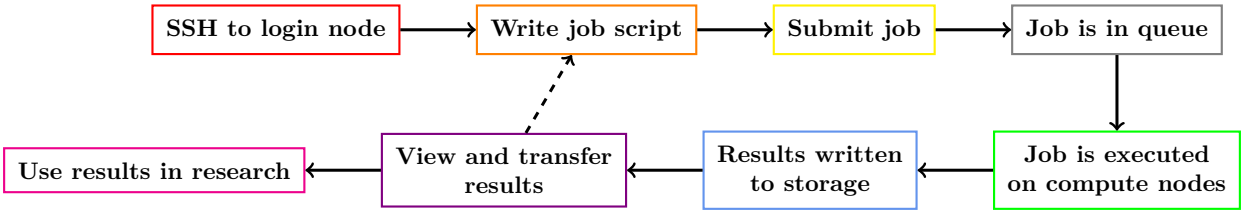
Rachel Wood

2023-04-25

This portfolio on high performance computing will look at BluePebble. The general structure of these systems can be thought of in the following way:



The process for running a job on the HPC can be broadly described as:



We can now go through these steps one by one.

Logging On

This step is relatively simple - with a Linux machine we can just use the `shh` command in the command line with `<username>@<hostname>` then enter the password:

```
ssh -X ac18826@bp1-login.acrc.bris.ac.uk
The authenticity of host 'bp1-login.acrc.bris.ac.uk (172.25.9.103)' cant be established.
ECDSA key fingerprint is SHA256:IhtH3HdjJiZN9urxgdSBewWGocOTEXxW/ZTq9jpJ2FI.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

Writing a job script

The following subsections detail the possible building blocks of a typical script to be run on an HPC.

Loading Modules

One thing that needs to be added to a script is the module the script makes use of. We can check what is available using `module avail`, however there are so many that it is hard to go through all the modules. We can get the count of lines in the output of `module avail` as 347:

```
module avail 2>&1 | wc -l
347
```

To make this easier, we can search for specific software. For example if we want `R`, we can use `module avail lang/r`

```
module avail lang/r

----------------------- /sw/tools/easybuild/modules/all ------------------------
   lang/r/3.6.0-gcc    lang/r/4.0.2-gcc
   lang/r/3.6.1        lang/r/4.0.3-bioconductor-gcc
   lang/r/3.6.2-gcc    lang/r/4.1.2-bioconductor-gcc
   lang/r/4.0.1        lang/r/4.1.2-gcc              (D)

  Where:
   D:  Default Module
```

Once we have indentified the appropriate modules we can add them in our script. As an example, say we want to use the `lang/r/3.6.0-gcc` module in our script, we can include:

```
module add lang/r/3.6.0-gcc
```

`#SBATCH` directives

We might want to include some information for `Slurm` to better allocate compute nodes. Some options we can set are:

- `--nodes`: requests the number of nodes
- `--ntaskts-per-node`: requests the number of tasks per node
- `--cpus-per-task`: requests number of cores per task
- `--time`: requests length of runtime
- `--mem`: requests RAM allowance per node
- `--error`: gives the name of output file to write errors to
- `--output`: gives name of output file for results

Note: Once the length of the `time` value, the job will be terminated, so it's important not to underestimate it.

Executable files

We will illustrate how to include a compiled program to be run in the job script using the simple "Hello World!" program in C. Say we have the following in a file called `hello.c`

```
#include <stdio.h>
int main()
{
   printf("Hello, World! \n");
   return 0;
}
```

We can load the appropriate module and compile the program in our workspace to an executable file called `hello`:

```
module load lang/intel-parallel-studio-xe/2020
icc hello.c -o hello
```

Then all that's needed is to include a line to run the compiled program in our bash script:

```
./hello
```

Including job information in output

Another useful thing for us to do might be to include some of the job information in the output, which we can do using `echo`. For example we might include

```
echo JOB ID: "${SLURM_JOBID}"
echo Working Directory: $(pwd)

echo Start Time: $(date)
echo End Time: $(date)
```

Submitting a job

Once we have a script, we can submit it using the `sbatch` command and we will be given a job id. For example:

```
sbatch job.sh
Submitted batch job 4891240
```

Queued Jobs

We can also check the job in the queue or cancel it using the following commands:

```
squeue -j 4891240
JOBID     PARTITION   NAME     USER  ST       TIME  NODES  NODELIST(REASON)
4891240         cpu job.sh   ab1234   R       0:01      1     compute49

scancel 4891240
```

Job execution

It is possible to use the `top` command to get a live update on the status of all jobs being executed, and so can see the progress of your job. We can also log onto the compute node our job is on using

```
ssh compute49
```

Results

All jobs produce an output file, whose name defaults to `slurm-<job_ID>.out`, we can examine the contents of these using `cat`. This will be stored in the `work` directory of the user space.