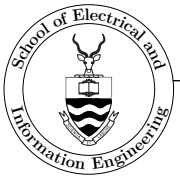

User guide for the ZA-Wits-Core Cluster grid.core.wits.ac.za

Scott Hazelhurst

4 September 2022

1 Introduction

The Wits Core Cluster (ZA-WITS-CORE) is a shared compute cluster between different academic units at Wits. The lead partners have been Electrical & Information Engineering, Physics and Wits Bioinformatics, with support from Computer Science and the Mathematical Sciences.

The cluster is kindly hosted by the Wits ICT, but and managed by the academic groups who own the cluster. The cluster management software allows us to share equipment so that we can minimise wasted CPU cycles, but also ensure the groups can get dedicated use of the resources that they put in at peak times.

2 Getting an account

An account needs to be authorised by a PI who has invested in the cluster. Get this person to send me an email to that effect saying who you are. Once that's been sent you should send me your public ssh key.

ssh keys

Log-in will generally only be allowed passwordless, using ssh keys. Here's a good introduction and you can find lots on the internet (please note that these instructions are for using default RSA keys while we would prefer you use ed25519 keys as explained below).

- `https://www.digitalocean.com/community/tutorials/ssh-essentials-working-with-ssh-servers-clients-and-keys`
- `https://help.github.com/articles/generating-ssh-keys/`

To get an account on the Core cluster you will need to provide an SSH *public* key. Please use version 2.

We recommend that you use `ssh-keygen -t ed25519` on a Linux or Apple machine. Once you've done that you will be prompted for the file name for the key *default* name and location will be shown. Just press enter (nothing else) and the default location will be selected. Unless you're an expert and really know what you're doing, and probably even if you are an expert, you should select the default name and location.

You will then be asked to enter a passphrase. This can be any text including spaces. Choose something meaningful you can remember – perhaps a number and words "2022 brilliant bottles", "books latitude margaret", "smooth boron muffins" and "twelve potato bricks"

are examples. Keep it simple – if you have three distinct words you can keep it secure without having to worry about having lower/upper case letters, punctuation etc. Make sure you choose something you can remember. Do not share your passphrase with anyone and do not store your passphrase on your laptop or local computer. Rather choose a simple passphrase that is memorable to you (something amusing perhaps) and remember it.

This will result in two files being created

- The public key in `~/.ssh/id_ed25519.pub`
- The private key in `~/.ssh/id_ed25519`

Please note that we are actively phasing out other types of keys: DSA will not longer be supported from August 2021 and RSA from February 2023

The private key really is private and should not be shared with anyone.

Your public key file **should** contain one (probably very long) line like this (if it says ssh-dss or ssh-rsa you've done the wrong thing – please make sure you are asking for ed25519 keys as shown above):

```
ssh-ed25519 AAAAC1182jahdjahd83398hajhd89y7ha828hajhddda89ya9hajhdakhddaajdkajd2782yadjahdM s
```

Execute `cat ~/.ssh/id_ed25519.pub` which will display your public key. Copy and paste it into an email or Slack and send it to me. There is no need to put in an a document and attach the document (in fact that just creates work for me).

If you are generating this on Windows using Putty please generate an OpenSSH compliant key. If you are generating this on Windows using Putty please generate an OpenSSH compliant key (see `http://the.earth.li/~sgtatham/putty/0.58/htmldoc/Chapter8.html`).

- Please do not send me your private key – it's bad security practice and I can't do much with it.
- If you send me an SSH2 format key rather than OpenSSH key, I will not be able to help you.

  This is what the first few lines of an SSH2 public key file look like and is **WRONG** for us:

  ```
  ---- BEGIN SSH2 PUBLIC KEY ----
  Comment: "rsa-key-20150629"
  ...
  ...
  ```

What I do with the key

The key point is that the file `.ssh/authorized_keys` on your account on the cluster contains the public keys of any machines *from* which you wish to log-in. When you get an account you give the administrator a public key, who also generates a public key for your ZA-WITS-CORE account. Both of these public keys are added to the `authorized_keys` file on the cluster. This means that you can log-in remotely from some machine and you can mutually log-in between all machines on the cluster (since all machines see the same home directory).

You can add more public keys to this `authorized_keys` file. You are welcome to change your key for your ZA-WITS-CORE account, but you must rememeber to add the new *public* key to `authorized_keys` and remove the old one.

If you have trouble with this step, please speak to someone in your group. I can't support users on this.

Caching your passphrase

Every time you run ssh or scp you will be prompted for your passphrase. This can be tedious. You can ask your system to cache your passphase by executing the following (note the single *back ticks* – not forward quotes)

```
eval ‘ssh-agent -t 1d ‘
ssh-add -t 1d ~/.ssh/id_ed25519
```

You will be asked to enter your passphrase and an agent (program(will remember your passphrase for one day (or till you reboot the computer or end your session on the local computer). This is a reasonable trade-off between usability and security. The security of ssh primarily relies on the secrecy of your private key. The point of the passphrase is to protect you if your laptop/desktop were stolen or hacked. For this reason please never share your passphrase with anyone and do not store the passphrase on your local computer.

3 Logging on

You should either log on to one of our two log-in machines:

- `cream-ce.core.wits.ac.za`
- `ui.core.wits.ac.za` (only if you have a grid certificate and you are using the grid software).

Users will generally only be allowed to log in to these computers (and all the others described below) using ssh keys.

In general, users should *only* log into one of the log-in machines, and **not** the worker machines.

Jobs should be run from `cream-ce` or from `ui`, and *not* from one of the worker nodes.

3.1 Networking and firewall issues

Firewall: We run a firewall. If you access the cluster remotely and can't make a connection (e.g., connection timed out) then you are probably being prevented access. Please send me your real IP address and ISP detail (if you don't know your IP address use a lookup service like `http://ip-lookup.net/` to find out).

3.2 cream-ce as a gateway

Note that the cream-ce machine deso not have full development environments and all libraries installed. The head node's environment is deliberately not the same as the worker nodes. The usual mode of working would be that you edit files, run simple scripts, etc on the head nodes, but do the actual work on the worker nodes. Don't log directly into the worker node (because you don't know what the loads are on the machine), rather you either do:

- `sbatch` : to launch a job
- `srun --pty bash` to get an interactive session on a worker ndoes.

Please read Section 6 carefully.

The worker nodes are named `n01.core.wits.ac.za`, `n02.core.wits.ac.za`, ... They do not have external DNS entries: the corresponding IP addresses are 146.141.240.1**01**,...

3.3 A note on working interactively

Many analyses are are useful to run interactively, but there is a risk that your connection to the cluster drops (e.g., a power failure at home, a network point dropping, and often if you have an idle connection – you don't type anything etc for a long period the connection 'times out'). When this happens usually your jobs will be killed shortly afterwards.

There are two solutions:

- Use *sbatch* with a batch file rather than *srun*. Create a batch file with the instructions you want to run. *sbatch* submits the job to the SLURM cluster manager where it is queued and when resources become available the job will execute. You can use `squeue` to monitor the state of the queue. See Section 6 for more details.
- Use the *screen* command as described in the next session.

Note that the use of *sbatch* reduces your chance of timeouts slightly and also the discipline of writing a SLURM batch job allows you more easily to reproduce work, keep track of what you did and write a variant of it.

3.4 screen

The *screen* command allows you to disassociate your terminal session from the physical computer you are running on. For example, you are doing something on your computer at work (you ssh into the head node, running an interactive session) and you need to go home. Even is the session on your work computer stays up that doesn't help you when you get home and ssh from home. Even if you're working on your laptop it's a problem: if you close your laptop the session will die – and even if you keep your laptop open there's a risk in network handover from WiFi to 4G your session will break and anyway it's not really a practical option to do this. *screen* tackles and the more general problem of sessions timing out. *screen* has many, many options and I'm only showing the basics, which is pretty much all you need IMO. There are three *screen* commmands you need

- `screen -S` create a session

- `screen -ls` list current screen sessions
- `screen -r` reconnect to a screen session, and also a variant `screen -dr` reconnect to a screen session which you're currently connected to either somewhere else or on the same same computer – `dr` means disconnect and reconnect.

Note that when (and only when) you are in an active *screen* session C-a (pressing control and *A* at the same time) has special meaning:

- C-a d (Press C-a, let go and then press d): *Disconnect* from the current screen session
- C-a a (Presss C-a, let go and then press a): Get the normal behaviour of C-a which is to move the cursor to the beginning of the current line
- C-a ESCAPE. Allow you within the scroll session to move up and down.

Here's an example of how you would use it.

1. `screen -S experiment1`

   Creates a screen session called *experiment1*. I now start the work that will get my first Nobel prize, and start running things.

   Now I want to keep the session going (i.e., the computer will continue working) so I disconnect – C-a d.

2. I reassure myself that I have a screen session running: `screen -ls` Note is shows the list of sessions (also with a number in case you name two sessions with the same name by accident or even forget to name a session – you can distinguish between them.
3. I am really paranoid so I say `screen -r experiment1` – I go back into the session. Now I've reassured myself I can disconnect `C-a d`
4. Now I go home. When I get home, I log into the cluster again and I can use `screen -r` to enter the session again. The session is still busy so I do `C-a d` to disconnect.
5. I have an idea for my second Nobel prize, so I create a new session: `screen -S prize`

   This gives me a new session – I do what I need and when I want to disconnect I say `C-a d`
6. Now if I say `screen -ls` I'll see two sessions open.
7. I can then pick which session I want to connect to or do something else . . .

Terminating a screen session: From within the screen session type `C-d` on a line by itself. The session terminates and cannot be recreated. Remember the difference: `C-a d` says *disconnect* from the current session but computer keeps on doing what it needs to; `C-d` terminates the current session.

4 Equipment overview

Besides the two log-in nodes, there are 9 storage servers and about 40 worker nodes, `n01.core.wits.ac.za` to `n45.core.wits.ac.za` (not all operational at any point in time), and some specialised servers hosting virtual machines. The number of worker nodes fluctuates depending on the overall load of the system.

All users have a common home directory on all machines. Take some time to set up your ssh keys so that you can log in from one machine to all others.

5 Storage

We currently have about 1PB of globally addressable disk space, as well as local disks on each server.

The key places are:

- `/home`

  Each user has a directory under /home, which is accessible from all machines. Please keep your storage used in this directory to *well* under 200GB. You can always say `du -s -B G ~`, which tells you in GB how much you are using in this directory. Please expect rude email if you go above this limit. If you need more than this or if you have long running projects and a need to share data please speak to me and we can find alternative space for you

- There is a `/spaces/` hierarchy for project work. You can have larger data sets. This is good quality data storage but not backed up. If you don't have a spaces directory you can ask for one.

- Data sets can be found in (again things change)

  - /dataA, /dataB, . . . , /dataH, /dataJ

- Note that much public data can be found `/dataB/popdata` . In `/dataB/aux` there are some useful data files like reference sequences for Build 37 and 38 of the human genome

- `/external/diskC`, `/spaces`

  These is also globally viewable and contain extra user and data projects A key point about these spaces is that it is sharing the resources physically across the cluster. However, the data is *private*. You should only access directories you have explicitly been given access to.

If you wish any data to be added to the cluster or need directories created please let me know.

These are moderate performance disks – for example, under a simulated load of 40 processes (spread across the cluster) each reading or writing a different 10MB file, the performance was good. However, if you tried the same with much larger files (e.g., large plink file), the cluster would probably collapse under the load. You might consider first copying to the `/tmp` directories of each machine (which are local) and then running your job. The same goes for output.

Please note backup/redundancy on the cluster:

- `/home`: this is stored on a RAID6 disk, which is highly reliable. In addition, the disk is backed up each morning to another RAID6 disk.
- `/spaces`: is RAID6, not backed up
- `/external/diskC`: is stored on a gluster (2-replica) volume and so is stored redundantly to the failure of one disk can be tolerated. No backup.

Memory

We have a heterogeneous system, partly by design, partly by circumstance that allows us to cost effectively support a wide range of computing needs. Our machines vary fro 24GB to 1TB of RAM.

6 SLURM

We use SLURM as our scheduling system. This is only a very quick introduction to SLURM. You can read more here

- A list of commands with links to detail: `https://slurm.schedmd.com/man_index.html`
- A summary of commands: `https://slurm.schedmd.com/pdfs/summary.pdf`

We have a small user community and we expect that everyone wil behave in courteous and thoughtful way to all other users.

The basic idea is that you put the job you run into a job script: this is essentially a shell script with some SLURM directives. These directives describe resources that users need. It's in your interest to be as accurate as possible. The fewer resources your job demands the sooner it will run; however, if your job exceeds its resource demands, your job will be killed off.

The basic things you can ask for are:

- how many computers (and processes) you want;
- how much memory you want;
- how long you want your job to run.

To repeat:

- Be honest for two reasons:
  - It's the right thing to do and respectful of other users.
  - If you lie or make a mistake (e.g., underestimate the amount of memory you need), your program will suffer. For example, suppose you say you need 8GB when in fact you need 20GB. The worker node will run your program, limiting your space. The likely effect is that your program will thrash and you are going to get CPU utilisation of less than 1%.

Please only ask for the resources you actually need. The less you ask for the better. Simple jobs should only need the following header (where you replace *MyJobName* with something sensible – this is descriptive and does not have to be related to any executable code name.).

Here some basic SLURM commands – there are other commands you can read up about

6.1 Informational

- `sinfo` State of the system – what nodes are available and their current state.
- `squeue` What does the job queue look like.
- `sdiag` For the nosy.

6.2 Running jobs – non-batch

srun

This can be used to run relatively short jobs.

```
srun  hostname
```

An example of running an interactive session is as follows:

```
  srun --cpus-per-task=8 --pty bash
```

The `--pty` command gives you the interactivity, the *bash* asks for bash terminal. In this example, we're also asking for one node to be allocated and asking for 8 cores on that node.

6.3 Running batch jobs

To run a batch job, create a batch file and submit it using the *sbatch* command. Here is an example

```
#!/bin/bash -l
#SBATCH -J hello_SLURM
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 01:20:00
#SBATCH -c 1
#SBATCH -p batch
#SBATCH --mem=4096
#


hostname
echo "hello world"
sleep 20
```

See the SLURM documentation referred to above for more features

- `-J` is the name of the job – choose something meaninful
- `-o` and `-e` is the name of the output file want for anything that your script produces on standard output and standard error. Note if you want, you can use the %j for the job number to be included in the name

- `-t` how much time do youw ant?
- `-c` or `--cpus-per-task` How many cores do you want to run for this job? For example if you are using PLINK and will be using 8 threads, then use `-c 8`
- `-p` which partition do you want to run? The partition is the same as the *queue*. Most cluster users should use *batch*, which is the default and so not needed.
- `--mem` how much memory do you want – measured in **mega**bytes.

There are other requests you can make. Note that SLURM also has `-N` and `-n` options for specifying multiple nodes and tasks. This is ONLY needed if you are using MPI or similar systems where you have parallelism across nodes. Most people will NOT want to use this – use the `-c` option which specifies the number of CPUs a job should have on the same computer.

6.4 Killing jobs

The `scancel` command will kill a batch job.

6.5 Running GUI programs on the cluster

Although we generally use the command line on the cluster for input and get output textually, there are times when we need or want to run interactive programs with graphical user interfaces (GUIs). This is sometimes very klunky because latency and bandwidth issues, particularly when running from off-campus, can make the user experience less than stellar, but still it can be done. We do this using a library called X11. If your local machine is Linux then you probably have the libraries installed already. If running MacOS, install a package called XQuartz and then open up a terminal using XQuartz rather than the the standard Terminal program. On a Microsoft Windows system, the Windows Subsystem for Linux (WSL) can support X11.

Unfortunately, direct built-in SLURM features for X11 clashed with some cluster options so the `--x11` option you may see in some SLURM manuals will not work. So you need to use a slightly different workflow – use the program `slurmx` rather than `srun`. This can be found in `/opt/exp_soft/bin` (and if you've loaded the `bioinf` package will be on your path – if not add `/opt/exp_soft/bin` to your path or just use `/opt/exp_soft/bin/slurmx`.

`slurmx` takes only some of the basic arguments to pass on to SLURM: `-c` for the number of cores (default 1); `-m` for the amount of memory in GB (default 4); -t number of minutes the scheduler should give you (default 480).

To use X11, when you log on to the cluster, use the `-Y` option

```
ssh -Y username@cream-ce.core.wits.ac.za
```

Then you use `slurmx` with the appropriate options. For example,

```
slurmx -c 2
```

SLURM will then allocate you a reservation on the cluster with two cores, 4GB of RAM and 480 minutes of wallclock time. An X11 window will then pop up with a session on one of the worker nodes. Your experience will depend on bandwidth and latency from your computer and the cluster.

6.6 SLURM and Nextflow

Nextflow supports SLURM well. Add this to your Nextflow config file *inside* the `profiles` stanza.

```
slurm {
      process.executor = 'slurm'
      process.queue = queue
  }
```

The run your job with a `-profile slurm` option, for example

```
nextflow run h3abionet/h3agwas/plink-qc.nf  -c beta.config -profile slurm
```

Running parallel programs

If you are using MPI, please see Section 6.11. If you are using a library that uses multiple cores (e.g., Pthreads or OpenMP [not OpenMPI]) then you must tell SLURM. A good example is the *admixture* program that we often use. Using the `-j` flag, we can tell admixture to use multiple cores, e.g., the command below allows admixture to run four threads at the same time on the same job thereby speeding up computation.

```
admixture -j4 data.bed 5
```

If you want to do this, you must also tell SLURM since it needs to know what your job requirements are. Of course, SLURM can't know what different flags of different programs mean so you must use the `-c 4` resource request (you asked `admixture` to use 4 cores; you need to ask SLURM 4 cores). The example below illustrates. Then, when you submit your job, SLURM will ensure that your program will run on a machine which as at least 3 cores free.

```
#!/bin/bash
#SBATCH -J admixture
#SBATCH -t 20:00:00
#SBATCH -c 5
#SBATCH -p batch

admixture -j5 ../${DATA}.bed 5
```

6.7 Running jobs on cream-ce

This is not a good idea and should not be done. For one, the system environment is not the same as the worker node. If you need to test something out, as long as your jobs run for less than a minute that's OK. I deliberately do NOT install all software on cream in order to prevent it being used for real computation.

Yes, you can run *nextflow* from the cream machine, provided you use the *pbs* executor.

(The name *cream-ce* comes from the grid world and means *Computing Resource Execution and Management Computing Element*.)

6.8 Logging on to worker nodes

As explained above, you should not run jobs from worker nodes. This confuses the scheduler and is considered anti-social behaviour. Many cluster systems simply forbid and prevent such logging. However, it's useful to allow since sometimes you need to check out the environment of a worker node. Thus currently logging in to worker nodes is allowed so that you can explore the environment. It is also permissible to run short test programs so that you can solve problems. As a rule of thumb you should only do this if

- The worker node concerned is not loaded;
- You only run 1 job at a time;
- Not for more than 5 minutes (and preferably less than 1 minute)

Please don't abuse this facility

6.9 Copying data to the cluster

For data that is relatively small say 30GB or so, I am very happy for people to do it themselves. If its data that lots of people might use then you should let me know so I can put it in a common place.

For very large data sets (say 100G+), please consult me before doing so we can find a suitable place to store the data and also make sure the data is not already on the cluster (so we don't download 1000 Genomes data five times).

You can use the *wget* command to download web-based data (http/https/ftp) directly to the cluster.

```
wget http://www.bioinf.wits.ac.za/software/genesis/Genesis.jar
```

For complex URLs, you right click on a link in your browser, there should be an option to copy the link into your clipboard which you can paste into the terminal window and just use wget.

If the URL is complex as is the case in your examples, you may need to do two things: (1) Put quotes around the URL so that characters like & dont get interpreted by the shell and (2) Explicitly name the output file with the -O option. For example

```
wget "https://x.ac.za/dwn/?acc=E32&file=dSE32%5Fnormalized%2Etxt%2E32.gz" \
-O GSE32504_non_normalized.txt.gz
```

6.10 Globus Online Endpoints

We run several Globus Online Endpoint for transfer of very large data sets. You need to get an account at `globus.org` and then speak to Scott Hazelhurst for details how to use the Wits endpoints.

6.11 Using MPI

MPI can be a tricky bugger, particularly as it can often fail silently. Patience is advised. **This has not been tested on the SLURM system**.

6.11.1 Compiling code

It is very highly desirable to use the same versions of MPI for compiling and runnning. If there's a system upgrade, you may need to recompile.

- You must compile on a worker node to ensure the enviroment is the same as the running environment (Hint: do a `srun --pty bash`).
- Decide which version of MPI you will use and set up the paths appropriately by doing *one* of the following

  1. `module load mpi/mpich-3.2-x86_64`
  2. `module load mpi/openmpi-x86_64`

- `mpicc` should now be on your path.

6.12 Advanced use of modules

We run the CernVM File system which "provides a scalable, reliable and low-maintenance software distribution service." `https://cernvm.cern.ch/portal/filesystem`.

6.12.1 Running an MPI job from SLURM

When you submit a job using SLURM, you give directives which results in a set of nodes being allocated for your use. We have MPICH and OPENMPI installed. Please note that as at September 2015, the version of MPICH installed in MPICH 3, and we just refer to it as MPICH. If you have any legacy code, making reference to MPICH2, please just make sure you are using MPICH. is given the name of a file with the computers that have been allocated.

Use the `--num-tasks` option to taks for each MPI process you want to run. Use the `-c` in addition to specify the number of cores per tasks if the individual tasks use multiple processes

When you run your MPI program you will specify how many processes you want to run using the `-np` option. This would be typically tied to the number of cores you've allocated.

More complex node requirements Skip this on a first reading (and perhaps even on a second or third reading). But if you are going to read it, you must read it carefully and understand it as this is an example of where a little knowledge is a dangerous thing.

Please note that there seems to be a bug with OpenMPI here – OpenMPI works well with the options previously given but not with the options givenhere.

When we launch an MPI job on a cluster we have to do two things: (1) use SLURM directives to make sure that we get the right number of resources in the right places; and (2) interact with the mpi-start and/or mpiexec programs to tell it how many processes we want to run and how to interact with the resources. It is our job to map between (1) and (2).

The main reason for using the options here is where you have a program which is parallelised using both MPI and threading (e.g., pthreads and OpenMP – NB. OpenMP is not the same as OpenMPI!!!). So in this scenario you want may want to have, e.g., 12 MPI processes each using 3 cores. This means that you need 36 cores in all but you also want to ensure that the cores are allocated per machine in batches of 3 (using OpenMP or threading requires that the cores are sharing the same physical memory).

To get this effect, you use `-N N -c M`, where you specify the number of nodes and the number of processes per node (e.g., `-B 12: -c 3`). The effect of this is to reserve 36 cores for you. How these cores are allocated is up to SLURM : this would have the effect of launching 12 tasks each with 3 cores. Depending on the scheduling algorithm and the state of the cluster it might run the tasks on 1 computer or 12 computers. All you can be guaranteed is that each task has 3 cores, all 3 on the same computer.

Why would you do this rather than just using 36 cores? This is not a rhetorical question – in many cases you should just use 36 cores. Unless you have really a good reason for doing otherwise, just use the "simple" MPI paralelleisation. The world is a difficult enough place tha you should make it even more complex than it needs to be.

But one reason you might do this is memory concerns. Each MPI process is full Linux process and so has its own memory space. If you have 8 MPI processes running on one computer, each process will need its own copy of the data, which in *some* cases may be onerous. However, libraries like pthreads and OpenMP (not openmpi) allow *threads* to share the same memory space.

To recap: when we parallelise a *job* using MPI, the job gets broken into a number of *processes*. Each process has its own virtual memory space independent from the other processes of the same job. In turn, a *process* can be broken into *threads* using libraries such as pthreads or OpenMP. We need to map processes and threads to cores on our computers. While difference processes can run on different physical computers, all threads belonging to any process must run on the same computer.

In the SURM job we ask for resources: we ask for a number of nodes, or rather we ask for *virtual nodes*. Depending on configuration as specified by the sys admin and/or you[1] the virtual node could be a physical computer, or part of a physical computer. You also say how many *process**ors** per node* you need to run[2]. The total number of cores you ask for is the product of these two quanities.

Then in your call to launch your MPI program (using mpi-start or mpi-exec) you say how many *processes* you want to run and how many *processes per node* you want to run.

7 Cluster resources

We have 45 worker nodes with more than > 1300 cores. You can see the state by saying `sinfo -o "%C"`

This tells you number of active, idle, and other total cores.
The machine n16.core.wits.ac.za has a NVIDIA Corporation GK110GL [Tesla K20Xm] installed, and the n20 machine an Intel Xeon PHI 5100.

7.1 Storage

Storage has been discussed above. If you are storing more than 100GB please discuss with me.

Also note that for users running highly parallel systems that NFS can be a huge bottleneck. For example, running 80 processes all accessing files on the NFS system can bring down

[1] There are more complex requests you can make not covered here.

[2] Again, this is a virtual processor rather than a physical processor – it would have been more accurate to call this "core" rather than "processor".

the entire system. If you are wanting to write a highly parallel system or have lots of jobs that have significant I/O components you should ensure data is appropriately handled (for example, copying data to the local /tmp directories of worker nodes). The use of the Gluster directory described below coudl also help alleviate the load on the system.

7.2 Operating System

We run CentOS 7 on most worker nodes. The virtual machine hosts run Ubuntu 208.04.

7.3 Software

7.3.1 Software installed in the /usr hierarchy

Each computer has its own `/usr` directory. However, the following software is installed in the `/usr` hierarchy as part of our standard installation on all machines (through the yum package manager) and you can rely on these being present on all machines

- gcc (GNU compiler collection: not just the C compiler)
- gdb
- emacs, of course
- vi because I'm broad-minded
- MPICH/Open MPI
- python 2.7 and python 3.9 (but see below on modules)
- Perl 5.8.7
- Java 1.8

7.3.2 Shared software set up

User home directories and `/opt/exp_soft` are shared across all machines. Users may therefore install software in their own directory. The following software is installed in `/opt/exp_soft`. **A partial list of the installed bioinformatics software can be found at** `http://grid.core.wits.ac.za/bioinfsoftware.html`, but this is not always kept up to date so as a scientists you should use `ls` to explore.

Note that to use software that is not on the standard paths (e.g., /usr/bin/, /usr/local/bin), you need either to explicitly give the path, e.g., `/opt/exp_soft/bioinf/bin/plink`, or better add the path to the your PATH variable. To see the current value of the PATH variable, say, `printenv PATH` This is the list of directories that the system uses when searching for executables.

Some manuals can be found at `http://grid.core.wits.ac.za/manuals`

The easiest way of doing this is to add the following to your `~/.bashrc` file, logging out and then logging in

```
source /opt/exp_soft/bioinf/setup.sh
```

This loads a number of useful onto your PATH and makes many (but not all) programs available. Do note, however, that as the cluster is updated some verions of programs (like version of Python) may change If you need specific versions of code you should do set up as explained below.

If there are binaries in other places you could to change your PATH variable as follows by adding lines similar to this to your `.bashrc` file

```
export PATH=/opt/exp_soft/sagrid/abyss-1.2.7/bin/:${HOME}/lib/bin:${PATH}
```

7.3.3 Modules

To see other software packages available you can say :module avail:

7.3.4 Python versions

- Python 3.6 is the default version of Python for our operating system but bioinformaticists will find Python 3.9 more useful and many standard Python modules are available there. Put this in your `.bashrc` file: `module load python/3.9`
- Python 3.9 is available if you say `module load python/3.9`. This is our recommended version of Python.
- Python 3.7 is available as module `python/3.7`
- Python 3.8 is availble through: `scl enable rh-python38  bash`

There is one subtle point on using modules that may be confusing. When you ask for a module to be loaded, the module system checks to see if it is already loaded and if so does nothing.

So if your default Python is say Python/3.10.4 that you load at the start of the session, and you then need to run Python 3.7 (say because there is a package that is only compatible with an earlier version of Python), you would say `module load python/3.7` and happily use Python 3.7. When you want to go back to using Python 3.10, you must **unload**[3] Python 3.7: `module unload python/3.7` If you try to re-enable 3.10.4 by doing a *load* the module system sees that the 3.10.4 paths are already on your PATH variable and so does nothing.

7.3.5 Perl

- Similarly, the default version of Perl is 5.16 and we need to keep it there. But Perl 5.2.6 with many standard libraries is available
- Perl 5.26: `module load perl/5.26`
- Perl 5.30: available through `scl enable rh-perl530`

7.3.6 GPU

If you are using the GPU on n16, you can load the environment

```
module load nvidia
```

that puts all the NVidia development kit on the right path.

[3] When you loaded python/3.7 you did not remove python/3.10.4 from your PATH; it just added the new paths at the beginning of the PATH variable and gave it priority

7.3.7 Intel Xeon PHI

The Xeon PHI is on n20.core.wits.ac.za. The only compiler we currently hae capable of compiling for the PHI is gcc 5.2 `/opt/exp_soft/gcc52` but it only provides very limited capabilities and if you are developing code you will need to download the Intel development kit into your own directories.

Consult the Intel documentation at `https://software.intel.com/en-us/mic-developer`

We run the Intel Manycore Platform Stack (MPSS) 3.5.2. Log into n20 and then ssh into the Xeon PHI.

```
ssh mic0
```

7.3.8 List of shared software

- **A list of the installed bioinformatics software can be found at** `http://grid.core.wits.ac.za/bioinfsoftware.html`
- gcc: the default gcc compiler is 4.8.5. Other versions are available

  To access GCC versions 7, 8, 9, or 11, execute

  ```
  scl enable devtoolset-N bash
  ```

  where N is the version you want. If you prefer `zsh` or some other shell you can use that in place of *bash*. Other versions of GCC can be foud using the CVMFS versions below
- bioinf

  A range of other bioinformatics programs including: admixture, mafft, muscle, Clustal (clustalo – new version), structure, eigenstrat, clumpp, distruct, pseq, vcftools, plink, cgatools, exonerate, maker. . . and others
- Mathematica

  Mathematica is installed in `/opt/exp_soft/Mathematica`. Run

  ```
  source /opt/exp_soft/Mathematica/setup.sh
  ```

  and put this in your set up file.

  To run Mathematica interactively, log into the cream-ce and ask torque to allocate you a worker node.

  ```
  srun --pty bash
  ```

  You can now run Mathematica.

  The first time you run Mathematica, the licence must be activated. Choose “Other ways to activate” and then “Connect to a network licence server”. Enter `cream-ce.core.wits.ac.za` as the name of the licence server. Read and accept the conditions.

  Once you have activated the licence you don’t have to do it again and can run Mathematically without the GUI (which is what you normally would do when you run Mathematica in parallel production mode).

- We have CVMFS installed for ATLAS and software – look at `/cvmfs`. We also use this to make available software installed elsewhere in the scientific commmunity

```
cvmfs_config probe cvmfs-config.computecanada.ca  soft.computecanada.ca
source /cvmfs/soft.computecanada.ca/config/profile/bash.sh
module avail
```

  This may be slow the first time you run as files have to cache.

  We also support ATLAS CVMFS – please refer to the standard ATLAS documentation