Overview

High performance computing (HPC) is often referred to as **supercomputers** or **clusters**. A cluster is a pool of computing resources (e.g. CPUs, GPUs and disk drives) that can be allocated to execute our computational tasks.

There are a number of clusters avaiable to UQ reserachers and studnets. Cluster users can submit jobs (code) to a cluster for execution on specified hardware. The computing resources on a cluster are shared among all cluster users. The resources and workload are managed by a central job management system (**slurm**).

The typical workflow of running a computational task on a UQ hosted cluster involve: 1. Remotely access the login node of a UQ hosted cluster, 2. Copy the code over to the cluster, 3. Set up the necessary environements (e.g. pytorch) needed to execute the code, 4. Send a request to the cluster workload managment system (`slurm`) to have your code executed, 5. Your code will be executed when the requested resources are available and it is your turn.

This guide will take you through the above steps on by setting up `pytorch` UQ's **Rangpur** cluster. Other clusters should work in a similar fashion.

NOTE: First, make sure you are familiar with using Unix Shell commands - most clusters don't have graphical interfaces.

1. Remotely Access the Login Node of UQ's Rangpur Cluster

Use `ssh` to connect to and interact with the **login node** node of a Rangpur as follows

```
ssh [user_name]@rangpur.compute.eait.uq.edu.au
```

Enter your password and this should put you in the `$HOME` directory. You can now interact with the cluster's login node by typing in Unix commands.

Enter `pwd` to see the absolute path of the current directory:

```
02:11:13 [user_name]@login1 ~ → pwd
/home/Staff/[user_name]
```

Enter `ls` , your current directory should be empty (for new users):

```
02:11:13 [user_name]@login1 ~ → ls
```

Leave the terminal window open.

NOTE: If you are connecting to a UQ hosted cluster off compus, you will need to connect to UQ's VPN first.

NOTE: `ssh` is a native commnad on macOS and Linux. If you are on Windows, you can use WSL or Putty instead of `ssh` .

2. Copy the Code Over to Rangpur

On your local computer, create an example script `main.py` using the editor of your choice. Pretend this is the training script of a neural network.

```
# main.py

import torch

# this should print "True" if the environment is configured correct with GPU access
print(torch.cuda.is_available())
```

Save the file and copy it over to the cluster using `scp [local file] [destination]` :

```
scp ./main.py [user_name]@rangpur.compute.eait.uq.edu.au:/home/Staff/[user_name]
```

We have now copied the just copied `main.py` to our remote `$HOME` directory (returned by the `pwd` command in step 1). `scp` can be used to copy files of any type and even entire directories with the `-r` flag.

Enter `ls` again in the cluster terminal window, you should see the copy of `main.py` on Rangpur

```
02:14:04 [user_name]@login1 ~ → ls
main.py
```

Lets do a quick test by invoking the `python` command

```
02:28:04 [user_name]@login1 ~ → python main.py
-bash: python: command not found
```

As you can see, the cluster does not have any enviroment set up to even run `python`, not to mention `pytorch`. We need to configure the enviroment oursevles for our script to run!

3. Set Up The Necessary Environements on Rangpur

3.1 Installing Miniconda on Rangpur

To successfully execute `main.py`, we will need to at least have `python` and `pytorch` configured on the cluster. Dependencies for deep learning are best managed using conda, which can set up an entire GPU enabled `python` environment with a few commands.

However, `conda` is not available to users of Rangpur by default as of early 2022 (other cluster may differ. You can type `module avail` in the cluster terminal to check if `conda` is an option.).

```
 02:38:05 [user_name]@login1 ~ → conda
-bash: conda: command not found
```

Luckily, `conda` can be easily installed. Open the miniconda website, Copy the link of the package for `Miniconda3 Linux 64-bit`. Use `wget` on the cluster login node to download the installer to our `$HONE`.

```
 02:41:16 [user_name]@login1 ~ → wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
--2022-04-13 14:41:21--  https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
Resolving xxx.xxx.xxx.xx (xxx.xxx.xxx.xx)... xxx.xxx.xx.xxx
Connecting to xxx.xxx.xxx.xx (xxx.xxx.xxx.xx)|xxx.xxx.xx.xxx|:8080... connected.
Proxy request sent, awaiting response... 200 OK
Length: 75660608 (72M) [application/x-sh]
Saving to: ‘Miniconda3-latest-Linux-x86_64.sh’

Miniconda3-latest-L 100%[===================>]  72.16M  41.9MB/s    in 1.7s

2022-04-13 14:41:24 (41.9 MB/s) - ‘Miniconda3-latest-Linux-x86_64.sh’ saved [75660608/75660608]

02:41:24 [user_name]@login1 ~ → ls
main.py  Miniconda3-latest-Linux-x86_64.sh
```

Run the installer, Press "Enter" to start installation:

```
 02:42:13 [user_name]@login1 ~ → sh Miniconda3-latest-Linux-x86_64.sh
Welcome to Miniconda3 py39_4.11.0

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
```

Agree to terms and conditions by entering `yes` when prompted:

```
 Do you accept the license terms? [yes|no]
[no] >>> yes
```

Press "ENTER" to use `conda`'s preferred default location. All the `conda` binaries will be stored here:

```
 Miniconda3 will now be installed into this location:
/home/Staff/[user_name]/miniconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

```

`conda` will now fetch its essential dependencies. At the end, make sure you enter `yes` to initialise `conda`:

```
  ...
  zlib               pkgs/main/linux-64::zlib-1.2.11-h7f8727e_4



Preparing transaction: done
Executing transaction: done
installation finished.
Do you wish the installer to initialize Miniconda3
by running conda init? [yes|no]
[no] >>> yes
```

NOTE: this step is important! Your shell will not be able to locate the installed `conda` command if it is not initialised.

You should see the following once the installation is complete:

```
==> For changes to take effect, close and re-open your current shell. <==

If you'd prefer that conda's base environment not be activated on startup,
   set the auto_activate_base parameter to false:

conda config --set auto_activate_base false

Thank you for installing Miniconda3!
```

As suggested, we should re-login to the cluster to see `conda` active. Enter `exit`, you will be disconnected from the cluster's login node. Use the same `ssh` command to reconnect:

```
02:52:34 [user_name]@login1 ~ → exit
logout
Connection to rangpur.compute.eait.uq.edu.au closed.

❯ ssh [user_name]@rangpur.compute.eait.uq.edu.au
[user_name]@rangpur.compute.eait.uq.edu.au's password:
```

Type `conda` again, you should see the following (no longer `command not found`):

```
 02:53:52 [user_name]@login1 ~ ➔ conda
usage: conda [-h] [-V] command ...

conda is a tool for managing and deploying applications, environments and packages.

Options:

positional arguments:
  command
    clean        Remove unused packages and caches.
    compare      Compare packages between conda environments.
    config       Modify configuration values in .condarc. This is modeled after the git config command. Writes to the user .condarc
                 default.
    create       Create a new conda environment from a list of specified packages.
    help         Displays a list of available conda commands and their help strings.
    info         Display information about current conda install.
    init         Initialize conda for shell interaction. [Experimental]
    install      Installs a list of packages into a specified conda environment.
    list         List linked packages in a conda environment.
    package      Low-level conda package utility. (EXPERIMENTAL)
    remove       Remove a list of packages from a specified conda environment.
    uninstall    Alias for conda remove.
    run          Run an executable in a conda environment. [Experimental]
    search       Search for packages and display associated information. The input is a MatchSpec, a query language for conda packag
    update       Updates conda packages to the latest compatible version.
    upgrade      Alias for conda update.

optional arguments:
  -h, --help     Show this help message and exit.
  -V, --version  Show the conda version number and exit.

conda commands available from other packages:
  content-trust
  env
```

[Optinal] We may now remove the installer:

```
 02:57:04 [user_name]@login1 ~ ➔ rm Miniconda3-latest-Linux-x86_64.sh
```

3.2 Installing a `pytorch` Envornment Using Miniconda

We will now use `conda` to install a **self-contained** python virtual enviroment containing `pytorch`. In other words, we will ask `conda` to fetch everything we need to run `pytorch` code.

Lets create a virtual enviroment to store the dependecies we need:

```
 conda create --prefix ./my-env pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
```

Breakdown: - `conda create` creates a new virtual enviroment. - `--prefix` allows us to specific a specific directory to story the downloaded dependencies. Refer to `conda`'s documentation for other options. - `./my-env` is the directory I specified to store the packages used by this enviroment, it can be any director you have access to. - `pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch` are the packages we need to download to get `pytorch` running on GPU. Others have already done the hard work building these packages so installing everything is a one-liner for us!

`y` to confirm when prompted:

```
   ...
  xz                 pkgs/main/linux-64::xz-5.2.5-h7b6447c_0
  zlib               pkgs/main/linux-64::zlib-1.2.11-h7f8727e_4
  zstd               pkgs/main/linux-64::zstd-1.4.9-haebb681_0



Proceed ([y]/n)? y
```

The installation of this virual envorment should take around 10 mins and you will see the following once done:

```
 done
#
# To activate this environment, use
#
#     $ conda activate /home/Staff/[user_name]/my-env
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Enter `ls`, you should see the directory (`my-env`) we instructed `conda` to store enviroment dependencies in:

```
 03:12:14 [user_name]@login1 ~ → ls
main.py  miniconda3  Miniconda3-latest-Linux-x86_64.sh  my-env
```

Up until this point, the enviroment is still not active so the packages in it are still not accessible to us. We will need to activate the environment next.

3.3 Activating the `pytorch` Envornment

`conda` created enviroments will persist on the cluster, we just need to activate them before we use them, no need to reinstall every time we log in. Any `conda` environment install using the `--prefix` flag can be activated using `conda activate [env-dir]`. In this case, type:

```
 conda activate /home/Staff/[user_name]/my-env
```

Finally, we do a quick test using our `main.py` to see if it works

```
 03:16:11 [user_name]@login1 ~ → python main.py
True
```

The script prints `True` indicating pytorch is indeed installed and it is running on GPU. We can theoretically run nerual networks on GPU now!

3.4 Installing Additional Packages in the `pytorch` Envornment

Once our enviroment is **activated**, you can add more packages to it by going through a similar installation process to pytorch:

```
 conda install opencv-python nibabel
```

`conda install` will install pacakges to the active enviroment you are already in.

*** However, we are not done yet! We are currently running everything on the **login** node (as indicated by `@login1`). **Running heavy computing jobs on the login node is strictly forbidden and can result in bans!**. The **login** node is shared and can slowdown under heavy load - it only intended for editing, copying code and setting up enviroments (as we have been doing).

4. Use `slurm` to Run Our Code

The real computing power of the cluster is hidden behind `slurm`, a popular cluster resource manager. We will need to submit a "request" to ask `slurm` to run our code on our behalf, using the hardware we specify. `slurm` will queue and execute jobs on *computing nodes* (not login). Wait times may increase if you already have too many running jobs.

4.1 Example `slurm` Script ("Request") for Rangpur

A `slurm` "request" comes in the form of `slurm` scripts. Lets create one. In your cluster terminal, type `nano slurm.sh` to create a file `slurm.sh`. Paste in the following content (remeber to replace the placeholder user name with your own):

```
#!/bin/bash
#SBATCH --time=1:00:00              # walltime limit (HH:MM:SS), job will be killed when this time is reached.
#SBATCH --nodes=1                   # number of nodes, usually 1 is enough
#SBATCH --ntasks-per-node=4         # 4 processor core(s) per node
#SBATCH --gres=gpu:1                # request 1 gpu (don't care what type)
#SBATCH --partition=vgpu20          # gpu node name
#SBATCH --job-name="test"

# The following is exactly the same as we did before, load up enviroment and run the code.

# Link the path to miniconda's binaries (created by conda installer)
export PATH="/home/Staff/[user_name]/miniconda3/bin:$PATH"
conda activate /home/Staff/[user_name]/my-env

# if for whatever reason, slurm is unable to locate conda, you can invoke the command with its full path:
python main.py
```

A slurm script has two main parts: - **specifications**: a list of `#SBATCH --...` specifications to describe your request to slurm, including job name, run time and hardware requirements. A full list of options can be found here. - **main program**: following the list of specifications is the commands we would like to execute (such as training a neural network, preprocessing and inferece). The commands there will be executed **after** on a non-login node with the specifications we defined (gpu etc).

For deep learning, the most important specifications are - `#SBATCH --gres...` : what resource (typically GPU) and how many to use. `gpu:1` means `1` of any `gpu`. We can also request a specific type of GPU if available (e.g. on Wiener we put `#SBATCH --gres=gpu:tesla-smx2:1`). - `#SBATCH --partition...` : what partition to run the job in. Most GPUs reside in `gpu` partiions. On Rangpur the paritions are `vgpu20` and `vgpu40` hosting 20 and 40GB cards, respectively. Hencing the partion choice will influce what GPU we end up getting.

In the example `slurm` script, we are requesting for `1` `gpu` from the `20GB` `gpu` partition.

You could also generate your own `slurm` script using a generator and copy it over once done.

5 Submit the `slurm` Script to Execute the Code

Save the `slurm` script (`ctrl-x` , `y` then `Enter` for nano) and submit it for execution using `sbatch` .

```
04:28:55 [user_name]@login1 ~ → sbatch slurm.sh
Submitted batch job 12614
```

You are given a `job_id` which can later be used to cancel the job if needed. Once the job starts, `slurm` will create two files under the current directory `[job_id].out` and `[job_id].err` to store `stdout` and `stderr` outputs, respectively. After execution, `12614.out` should contain the same line as executing `main.py` before. Except this time, the code was executed using `slurm` allocated resources rather than on the login node.

```
True
```

Jobs read and write to the same local file system we see, hence they will output as if they were executed from an interactive terminal. You can submit multiple jobs (even with the exact same code), just make sure their their outputs don't overwrite each other! (e.g. submitting 5 training scripts that save weights to `[ckpt]/weights.pth` will see them overwrting the checkpoint of each other). You can check the status of your submitted jobs with `squeue -u [user_name]` :

```
04:29:05 [user_name]@login1 ~ → squeue -u [user_name]
             JOBID PARTITION     NAME        USER ST       TIME  NODES NODELIST(REASON)
             12614    vgpu20     test [user_name] PD       0:00      1 (Priority)
```

`ST=PD` means pending. Once your jobs starts running (when its turn comes), you will see `R` in place of `PD` .

To interrupt / kill a running or pending job, use `scancel [job_id]` .

Some Final Recommendations

- Write the code and `slurm` scripts on a local computer using your preferred code editor.
- Alaways synce your code with `git` or Github, you can access the codebase using `git clone` on the cluster.
- Some clusters may ignore the `--time` specification. Make sure your job is not running indefinately.
- Test your code thoroughly before submitting to slurm, don't wait 5 hours for your job to start and hit an error immediately.
- If you are submitting multiple jobs with the same code, to avoid overwites, make sure they are not outputting to the same.
- You can mount a cluster directory to your local computer using `sshfs` , and access the cluster file system as if it was an connected hard-drive.
- The `$HOME` directories are limited a 5GB on Rangpur, be sure to only store code and maybe `conda` enviroments there. For data storage you should use the `/scratch` drive or your research group drive.