WITS UNIVERSITY

School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

A Tutorial on Using the Mathematical Sciences HPC Cluster

Hairong Wang

April 15, 2025

This document provides a quickstart tutorial on using the Mathematical Sciences (MS) high performance computing (HPC) cluster.

1 Basics

1. To Log into the cluster, run the following bash shell command in a Linux terminal:

```
ssh <username>@ZZ.ZZ.ZZ.ZZ
```

   where `ZZ.ZZ.ZZ.ZZ` is replaced by the IP address of the login (or head) node of the cluster. The IP address of MS cluster login node is currently **146.141.21.100**. Alternatively, run the following command

```
ssh -X <username>@ZZ.ZZ.ZZ.ZZ
```

   for opening a GUI on your local machine (may not work depending on how the cluster is setup). Enter your login password at the prompt following the `ssh` command. If it is the first time you are logging in the cluster, you will see a message like "The authenticity of the host cannot be ... Are you sure you want to continue connecting (yes/no) ?" Type 'yes' to continue. This will add the remote server to the list of known hosts on your local machine.
2. Once you are logged in, you can use Linux shell commands to perform operations such as creating (`mkdir <directory name>`) or removing (`rm <filename>`) a file.
3. To transfer code or data between the local and remote (i.e., the cluster) machines, we can use `scp` or `rsync` commands. The usage of `scp` is typically

```
scp </path/to/the/sourcefile> </path/to/the/destination>
```

   For example, from my local machine to copy a file to the cluster, I can use

```
scp localFile.txt <myusername>@146.141.21.100:~/myFolder/
```

   which copies a file 'localFile.txt' from my local machine to the remote machine. The symbol '~' means your home directory.
   Similarly, the following command

```
rsync -r </path/to/the_code_directory> <myusername>@ZZ.ZZ.ZZ.ZZ:~/
```

   transfers a directory from the local machine to the remote machine. After the transfer is completed, you will find the code on the cluster at `~/the_code_directory`.
   You may also transfer the data or code from the cluster using the above two commands, where you just need to specify the source and destination accordingly.
4. Always type `logout` in the terminal to exit the remote machine.

5. **Note** that when you login, you are logging into the head node, or login node, of the cluster, which manages all the accounts. **It is very important that you should never run your program directly on the head node. To run your program on the cluster, you should submit a job, usually a batch job that will run when the resources become available.**

2 Running Your Code on the Cluster

The MS cluster uses SLURM as the cluster management and job scheduling tool. When running our code, we must run them on the compute nodes, not on the login node. There are different ways to do this. One way is to submit a job using `sbatch` command. Once the job is submitted, it will return a job ID, which can be used to track your job status, such as pending, running, stop, etc. The submitted job will be queued, and the SLURM job scheduler will schedule the job once the requested compute resources become available.

The other way is to run a job in an interactive way. In this case, we can use `srun` and `salloc` commands. See below for more details.

2.1 Submitting a Job on the Cluster

`sbatch` is a scheduler command that submits a SLURM job. In order to use this command, you need to write a job script first.

1. Note the following when you write a job script (typically consists of SLURM and shell commands) for SLURM.
   - Include the type of shell you are using as the first line in your batch script — `#!/bin/bash`.
   - All `#SBATCH` lines must come before shell script commands.
   - Batch scheduling script (statements that start with `#SBATCH`) is interpreted upon job submission. Shell scripting (shell commands) is only interpreted at runtime. Therefore, it is possible to successfully submit a job that has shell script errors which won't cause problems until later when the job actually runs.
   - The `sbatch` command is used to submit your job. For example, the following

```
hwang@mscluster-login1:~/src_1$ sbatch tut_1.sh
Submitted batch job 105448
```

   Submits a job using a job script named `tut_1.sh`. Upon submission, a job ID is returned (it is 105447 in the example) and the job is spooled for execution.
   Now I can use `squeue` command to query the job queue.

```
hwang@mscluster-login1:~/src_1$ squeue --me
JOBID  PARTITION  NAME     USER   ST  TIME  NODES  NODELIST(REASON)
105448 bigbatch   tut_trai hwang  R   0:32      1  mscluster44
```

   The output from `squeue` shows the job ID, the partition it is scheduled, the name of the job, the user name, and status of the job ('R' means the job is currently running; 'PD' means the job is pending), the time it is taking so far, the number of nodes it is using, and finally, the name(s) of the compute node(s) the job is running on.
   Once the job is completed, you can check possible error and terminal output in the designated files you specified in the job script, or in the code.

2. For instance, the following script specifies some essential values such as partition, time limit, memory allocation, and number of cores. It also specifies additional parameters such as job name and output file. (For more details, go to https://slurm.schedmd.com/quickstart.html)

```
#!/bin/bash
# These lines are comments for slurm
# specify a partition by its name in which to run your job. In this case, the name of the partition is bigbatch
#SBATCH --partition=bigbatch
# specify number of nodes you are requesting to run your job
#SBATCH --nodes=2
#or #SBATCH -N 2
# specify number of tasks
#SBATCH --ntasks=8
# or #SBATCH -n 8
# specify the number of tasks per node
#SBATCH --ntasks-per-node=1
# specify the total number of tasks
#SBATCH --ntasks=1
# specify the number of CPUs per task
#SBATCH --cpus-per-task=1
# specify memory required per node. In this case 10MB is requested
#SBATCH --mem=10
# specify the wall clock time limit for the job. In this case, it is 5 mins
#SBATCH --time=00:05:00
# specify the job name
#SBATCH -J PC-com
# or
#SMATCH --job-name=PC-com
# specify the filename to be used for writing output. You need to put the correct path and the output file name
#SBATCH -o /home-mscluster/<username>/slurm.%N.%j.out
# specify the filename for stderr
#SBATCH -e /home-mscluster/<username>/slurm.%N.%j.err
# The following are shell commands -- which will be executed according to the settings specified above

mpiexec -n 8 ./mpi_mat_vect_mult 64 64
```

2.2 Running a Job Interactively

The advantage of running jobs in batch mode is obvious. The jobs will be queued and run when the required resources become available. The job(s) will be queuing or running even when you log out of the cluster. In an interactive mode, however, this is not the case. Interactive mode can be useful for debugging, testing, or performing tasks that require user interaction. So, if your job can run in a very short amount of time, e.g., a few minutes or a few dozens of minutes, you may run it in an interactive way. Commands such as `srun` and `salloc` are often used to run interactive jobs.

1. Using `srun`, for example,

```
srun --partition=stampede --nodes=1 --ntasks-per-node=1 --ntasks=1 --cpus-per-task=1 --time=00:05:00 --pty bash -i
```

the `srun` starts an interactive bash shell session on a single compute node on the `stampede` partition, and some other options where their names suggest what they are for. The pseudo terminal (`--pty`)

allows you to interact with the compute node (not your login node) directly as if you were logged in via a terminal.
Once the interactive session starts using the above example, you can run a job by just typing in the command, such as `python train.py`. The following is another example

```
srun -p bigbatch --nodes=4 --ntasks=8 --ntasks-per-node=2 mpiexec -n 8 ./
  mpi_mat_vect_mult 64 64
```

where it requests 8 processes using 4 nodes in `bigbatch` partition. If there are available interactive nodes, job will run immediately. Otherwise, it will wait until there are free nodes to run it.

2. `salloc` command is used to request an allocation from the SLURM scheduler and takes the same arguments as `srun`. For example, when the following command

```
salloc --nodes=1 --partition=bigbatch --ntasks-per-node=1 --time=00:30:00
```

runs, it returns with

```
salloc: Granted job allocation 105453
(base) hwang@mscluster-login1:~/src_1$
```

where it shows an interactive bash shell session starts. If I run `srun hostname` it prints out the compute node name, which is in this case `mscluster44.ms.wits.ac.za`, that is allocated for job 105453. Once I have this session, I can run code using `srun`, e.g., `srun python train.py`.

3. **Note** that it is very important to cancel the interactive sessions once you are done. You can do so by cancelling the job using `scancel`, for example `scancel $SLURM_JOB_ID` cancels the interactive job you are scheduled using the job ID (retrieved using the environment variable in SLURM). See below for more information on `scancel`.

2.3 Some Useful SLURM Commands

1. To list information on job queues
   - `squeue -u <username>` or `squeue --me` — lists all the current jobs for a user specified by the username.
   - `squeue -u <username> -t RUNNING` — lists all running jobs for a user.
   - `squeue -u <username> -t PENDING` — lists all pending jobs for a user.
   - `squeue -u <username> -p bigbatch` — lists all current jobs in the `bigbatch` partition for a user.
2. To cancel a job
   - `scancel <jobid>` — cancels a job with `jobid`.
   - `scancel -u <username>` — cancels all the jobs for a user.
   - `scancel -t PENDING -u <username>` — cancels all the pending jobs for a user.
3. You can get some information on the partitions using `sinfo`, where it shows the partition names, their availabilities, and status etc.

3 Running Deep Learning Models on the Cluster

3.1 Installing the Required Software

To run your deep learning (DL) models, the first step is to set up the environment, where you need to install the required software including Python, a deep learning framework (e.g., Pytorch), and any other dependencies (e.g., CUDA for GPU acceleration). Use tools such as virtual environments to manage dependencies and avoid conflicts.

In most cases, you probably need to install conda and pip to create environments and manage Python packages, respectively. Consider installing either anaconda (https://repo.anaconda.com/archive) or miniconda (https://repo.anaconda.com/miniconda/). For example, to install miniconda3, you can run the following from your terminal.

```
# Download the miniconda installer
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
# Install miniconda, and follow prompts
bash Miniconda3-latest-Linux-x86_64.sh
# conda should be in your bash shell now. If not, run
# source ~/.bashrc
```

Once you have `conda`, you can run `conda create` to create environment, `conda activate` to activate an environment, and `conda install` to install needed packages in an environment etc. See an example below.

```
# Create a new environment called 'myenv' with Python version being 3.10
conda create --name myenv python=3.10
conda activate myenv
```

To install the desired DL framework, you can use the official installation command from the relevant website. For example, to install Pytorch (https://pytorch.org/get-started/locally/#linux-anaconda) in the new environment using `conda`, the command to use for Linux, Python, and CUDA 12.1 is

```
conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c
  nvidia
```

The command varies depending on your OS, CUDA version, and package manager.

You can create different environments to manage different dependencies, and activate the corresponding environment before running your Python code.

For this tutorial, a folder named `examples` provides two example codes in two subfolders, namely `src_1` and `src_2`. In `src_1`, an example job script (`train_job.sh`) is given. In this job script, the example environment, `myenv`, we created earlier is activated, and a python script is executed in this environment. After setting up the environment, type

```
sbatch train_job.sh
```

to submit the job. The job script specifies the file names where the error message or terminal outputs are to be written, respectively. In the example, they are written to files in a folder named `output`.

Write a job script based on `train_job.sh` for the python script given in folder `src_2`.

3.2 Datasets

To load your datasets onto the cluster, please read the instructions on Page 12 of MS cluster community guidelines that can be found at https://www.dropbox.com/scl/fi/9m8g90bg6eudp853me6rz/mscluster-08-02-2024.pdf?rlkey=63tclved10ld94886af4zbo6u&st=0wdq1wmb&dl=1. The document indicates that **MS cluster provides storage for your datasets at three different locations**:

1. your home directory,
2. `/datasets/<username>/`,
3. and `/gluster/<username>/`.

Try to use the latter two, and avoid storing large datasets on your home directory.

3.3 Preparing Your Code

It is generally easier to prepare your code locally, and only transfer the mature code to the cluster for running jobs, potentially multiple of them simultaneously.

Finally, tools such as Git for managing your code and Weights & Biases for tracking your experiments could be very helpful. See an easy-to-follow tutorial on Weights & Biases at the following link: https://colab.research.google.com/github/wandb/examples/blob/master/colabs/pytorch/Simple_PyTorch_Integration.ipynb#scrollTo=qmRJWz4nvi8m.

3.4 Useful Resources

1. https://github.com/WitsHPC/HPC-InterestGroup/tree/main/talks/linux/clusters by Michael Beukman.
2. MS cluster community guidelines that can be found at https://www.dropbox.com/scl/fi/9m8g90bg6eudp853me6rz/mscluster-08-02-2024.pdf?rlkey=63tclved10ld94886af4zbo6u&st=0wdq1wmb&dl=1.