ECE-NEBULA CLUSTER

Getting Started as a User

Contents

Quickstart Guide

Read this section to understand the steps you need to take get a job up and running. We will go into more detail about all of the steps in the sections below.

When your account is made, you will receive a username and password. This is your account information for the login node. For the remainder of this document, I will refer to the username as "user_name". To run your first job, perform the following steps:

1. Ensure you are either on a UW campus network or logged in via the VPN
2. Using your favorite ssh client, ssh into ece-nebula07.eng.uwaterloo.ca. For instance, I use the Linux command line, so I would log in with:

   ```
   >> ssh user_name@ece-nebula07.eng.uwaterloo.ca
   ```
3. This will log you in to your home directory (in our case, /home/user_name), but we can't run jobs from here. Immediately change to your NFS directory, which lives at /slurm_nfs/user_name. For instance, I would run the command:

   ```
   >> cd /slurm_nfs/user_name
   ```
4. If this is your first time logging in, you will see that there are two files already in your directory: slurm_job_t5.sh and test_t5.py. These two files are useful templates for you to use, so let's submit our first job. Execute the command:

   ```
   >> sbatch slurm_job_t5.sh
   ```

You should see a confirmation that your job has been submitted, that looks something like this:

```
Submitted batch job 20350
```

5. Your job has now been submitted to a compute node, but we need to know when it is done running. We are going to use the “squeue” command to do this. We can achieve this by simply executing the command:

   ```
   >> squeue
   ```

   This will show you the jobs currently running, and you will see that one of them is yours. This should look something like this:

   ```
   JOBID PARTITION     NAME     USER    ST       TIME  NODES NODELIST(REASON)
   20356 smallcard     test user_name    R       0:02      1 ece-nebula06
   ```

   This tells us that job 20356 is running on the “smallcard” partition and has been running for 2 seconds.
6. It is very inconvenient to keep checking squeue, so instead we can set our terminal up to watch it, like this:

   ```
   >> watch -n 1 squeue
   ```

   This will continually call squeue every second and update the terminal to show the output of the command. Execute the watch and keep it open until your job goes away. This indicates it is finished. Use CTRL+C to exit the watch
7. Type the command:

   ```
   >> ls
   ```

   And you will see a new file has been created. This file starts with the text “slurm”, then your job ID. For instance, my file for this job is called “slurm-20356.out”.
8. Now, let’s look at that file, we’ll see that it captured the entire output from the python script (including warnings, that I intentionally left in so you could see it). Execute the command:

   ```
   >> cat slurm-20356.out
   ```

   And you will see the output from your job

If you got that far, congratulations! You have successfully submitted, executed, and read the output of a GPU job on the ece-nebula cluster! Read on to understand how that job is set up so that you can start making your own jobs.

Understanding How Jobs Run

The ece-nebula cluster is made up of two types of computers, from the user's perspective: the login node and the compute nodes. Users are allowed to log in only to the login node, but that node does not have any GPUs and should never perform computation for your job. When you submit a job, the login node searches for a node that meets your resource requirements, and if one is found it will allocate that node to your job. If none are currently available but your resource request is valid, your job gets put into the queue and will run once resources are free. Finally, if no node exists on the cluster that meets your requirements, you will see an error when you try to submit the job. We'll go into requesting resources in more detail below, but first we need to get something important out of the way:

The Network File Share

Compute nodes and the login node are physically different computers. All of your work, files, and datasets must be on a networked file share in order for the login node and the compute nodes to perform your job. For instance, if you were to put a python script you wanted to run on a folder that the compute node can't see, there is no way for the node to run your job, and you will see an error in the output file.

This is why we had to navigate to the specific folder. The folder /slurm_nfs/user_name is your folder that is accessible to all nodes in the cluster. You can make subfolders in this folder, but you need to make sure that all of your code goes there.

The administrators of the cluster will periodically remove accounts that have not been accessed in six months. Therefore, it is best not to store data on this cluster long-term.

The /datasets_nfs Share

The folder /slurm_nfs physically sits on the login node, which (as of January 2024) has limited storage of about 400GB. Although that may sound like a lot, for a multi-user system like this one it is in fact very small. If you are going to work with large datasets (even a few GB is considered "large" given the number of users), you should put your datasets into the /datasets_nfs folder instead of /slurm_nfs. This is as simple as copying the dataset into the /datasets_nfs folder from your local machine.

The /datasets_nfs folder is also networked to all of the compute nodes and the login node. The major difference is that this folder sits on a computer with 1TB of storage space, and cluster administrators will clean it more frequently – any file not accessed within the last week may be subject to deletion if the drive gets full. Therefore, /datasets_nfs should **never** be used for long-term storage.

In the future we hope to expand the storage capacity of the cluster, but you should be aware that the cluster will never be a place to store huge amounts of data and AI models. We simply don't have the resources to provide that service to all UW engineering undergrads.

Why Jobs Load Slowly

Since everything is on the NFS, jobs, models, and data must be transferred to the compute node you are working on before the job can start running. For this reason, running a job that takes a lot of data can take quite a while. Once the job is fully loaded, however, it will run as fast as the GPU can run. Therefore, it is best to use the cluster to load jobs that process a batch of inputs at once, rather than only a single input. For instance, the test job, test_t5.py, is in fact quite poorly set up, because it loads a model, runs a single query on it, and ends. This is not ideal, because the loading happens over the network.

Instead, you should consider processing a large number of inputs at once. A simple way to do this is to create a file or a folder on the /datasets_nfs share, and process all of the inputs at once.

Requesting Resources

SLURM's purpose is to allocate nodes to jobs[1]. The first lines in the submission script tell SLURM the type of resources you need. Whenever two jobs request the same resources, SLURM is set up to schedule them to run in the order in which they arrive. The later-arriving job will be held in a job queue until the first-to-arrive job is complete.

To make job allocation easier, we have combined the compute nodes into partitions based on VRAM. The partitions have names to remember them. They are:

- **smallcard:** 2 nodes with 8GB VRAM each.
- **midcard:** 1 node with 20GB VRAM (midcard will be expanding to 2 nodes in S2024)
- **dualcard:** 2 nodes with 2 GPUs, each with 20GB VRAM
- **bigcard:** (Planned for Feb 2024) 1 node with 2 GPUs, each with 48GB VRAM

Open your job submission script (if you're working with the starter scripts, this would be slurm_job_t5.sh). The comments after the lines starting with #SBATCH indicate the resources you are requesting. There are a few of interest:

- **#SBATCH --partition=smallcard**: this is where you set the partition you want to work on. Which partition your job goes to determines the resources available to you
- **#SBATCH --cpus-per-task=1**: for a job that is CPU-heavy, you can change this to a higher number. Generally, you can allocate all of the available CPUs of a node, but SLURM usually gives one to the OS so it's best to allocate a few less than maximum:
    - smallcard nodes have 4 CPUs
    - all other partitions' nodes have 24 CPUs

  For most AI workloads, a single CPU is fine since the compute happens on GPU.
- **#SBATCH --gres=gpu:1**: if you are running a job on smallcard or midcard, all nodes have only 1 GPU, so leave this unchanged. If you are running a job on the other partitions, they have dual GPUs, so you can increase this to 2 if you are using both GPUs. Note: if you are

[1] We can argue if it should be "jobs to nodes", but saying nodes to jobs is more accurate since limited resources are given to potentially unlimited jobs, not jobs to resources.

not using both GPUs, setting this to 1 lets SLURM allocate someone else's single-GPU job to the other GPU.

- **#SBATCH --mem-per-cpu=12G**: this is allocating 12GB of RAM (not VRAM) to your job. This is best set to 1.5x the VRAM of your GPU, since many AI workloads will first load data into the RAM then move it to VRAM, and the OS needs some as well. All nodes have at least 1.5x their VRAM in normal RAM.
- **#SBATCH --time=00:10:00**: time, in HH:MM:SS format. You should set this to a reasonable, finite number. When in doubt, make it bigger, but do not make it infinite. Most of the time your job will finish before it times out, but if something goes wrong and it gets stuck it is best to automatically end the job.

How to Choose Resources

You may have never had to think about the resources you need for a job before – normally, if you only have access to a single computer, you just use what's there. It's not a huge deal if you request the wrong resources, but it's generally better to request more resources than less until you are certain of what your job needs. This is because running a smaller job on a bigger machine will succeed, but be wasteful, while running a job that is too big for the machine will end in an error.

Normally, when running an AI workload, I ask myself the following questions:

- How big is the AI model I want to run? Normally this is well-known, and can range from a few MB to tens of GB.
- Do I need a lot of CPU? Normally you don't, but if your job requires, say, preprocessing a large dataset then loading it into a GPU for learning, you may use a lot of CPU
- How long will this job run for? The only thing that teaches you this is experience. Most of my AI jobs either run for under a minute or several hours, with very little in between

You should try your best to allocate as few resources as possible to your job, so that if someone who needs more resources than you wants to run a job they are able to. However, the cluster is for learning, so it's OK if you allocate more than you need.

Checking if Resources are Available

There are two SLURM commands that are useful to check if resources are available. For instance, if you want to run a small job that can easily fit on a smallcard machine, but both machines are in use and no other machines are, you can request resources on a bigger machine instead. To see what is available, you can use the commands "sinfo" and "squeue".

- **sinfo** tells you the state of the machines. Any machine in an "idle" state can be allocated, otherwise it is either already in use or it is down
- **squeue** tells you how many jobs are in the queue for the partition. If all nodes are allocated, squeue can give you an idea of where to submit your job. For instance, if all nodes are allocated but squeue shows that midcard has only one job and all of the others have ten jobs, it makes sense to allocate to midcard, since it is likely that you will get to run your job sooner.

The scancel Command is your Friend

If you start a long-running job but then realize that something is wrong (you downloaded the wrong dataset, say), you can cancel a job by its ID or you can cancel all jobs belonging to your username.

- **Cancel a job by its id:** scancel JOB_ID (for instance, to cancel job 1234, “scancel 1234”
- **Cancel a job by your username:** scancel -u user_name. Note that this cancels ALL of your jobs, so if you have submitted more than one it will stop them all