An Introduction to GPU clusters in general and the MLP GPU Cluster in particular

By Antreas Antoniou

Talk Summary

1. GPU Clusters and Slurm:
    a. Introduce GPU boxes, GPU clusters
    b. Introduce MLP cluster and slurm
    c. Small slurm demo - including a cheat sheet
2. Using the cluster and developing cluster-ready code:

    a. Getting started with the cluster
    b. Good workflows for development
    c. MLP Cluster usage demo
    d. Good cluster etiquette and further tips
    e. How jobs can fail, running jobs in a fault tolerant manner
    f. How to make sure your jobs keep running while you sleep

Why do we need GPUs for our research, what happened to CPUs?

- CPU - Num Cores: 4-24 -> 4 - 24 parallel processes
- GPU - Num Cores: 1024 - 5760 -> 1024 - 5760 parallel processes

- Some simple, back of the envelope math to help us visualize what that means:

  Speed per core: CPU -> 4GHz, GPU -> 1.580 GHz (Titan X)
  Assuming a particular deep learning operation requires 100MHz to complete and that we have to compute 10^9 of such OPs

$$time\_required = \frac{per\_op\_cost}{per\_core\_ops\_executed\_per\_second} \times \frac{1}{num\_cores} \times total\_num\_ops$$

  CPU would need ~ 12.1 days
  GPU would need ~ 0.1 days

Common configurations of Research (GPU) Machines
GPU Boxes

Single server grade machine containing:

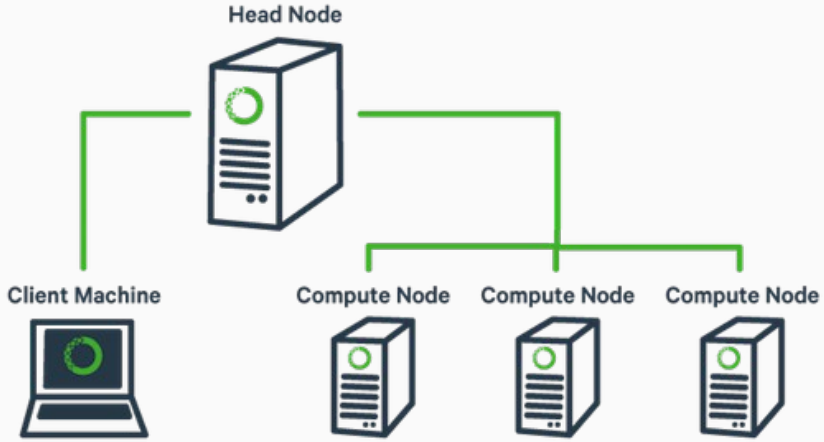
- 4 - 8 GPUs
- 12 - 24 CPU cores
- 64 - 256 GB RAM
- Each machine is used by multiple people, that have to **manually schedule who** will use **what**. For **each box**, each user needs to setup a **separate environment** and they have to **manually launch jobs one by one**.

GPU Box Style Architecture [1]

1. Modified version of https://ucdavis-bioinformatics-training.github.io/2017-June-RNA-Seq-Workshop/monday/cluster.html

Common configurations of Research (GPU) Machines
2. GPU Clusters

GPU Clusters are usually **made of multiple GPU boxes** which are **combined** using **cluster scheduler software** (e.g. Slurm), which abstracts away individual machines and enables users to “submit” jobs, using an **environment they configure once**. **Users can submit multiple jobs (to GPUs located in various different GPU boxes) with a single command.**



GPU Cluster Architecture [1]

1. Modified version of https://ucdavis-bioinformatics-training.github.io/2017-June-RNA-Seq-Workshop/monday/cluster.html

The MLP GPU Cluster

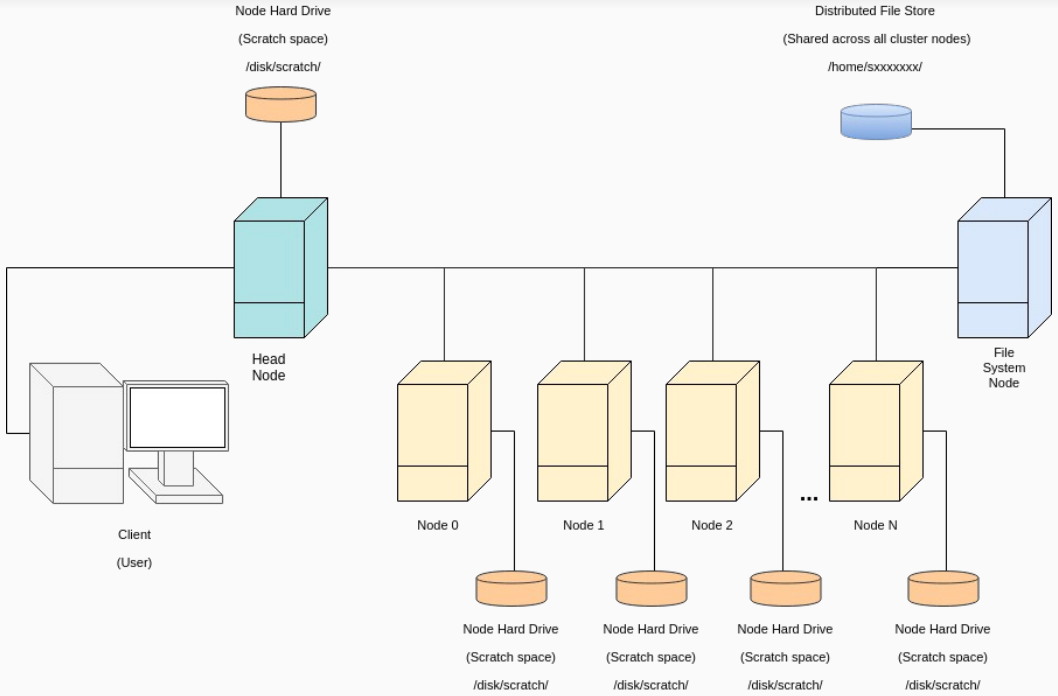
- GPU Availability: 25 GPU boxes with 8x 1060 Ti GTX each, for a total of 200 GPUs.
- Node Hard-drives: Each node has their own 2TB hard-drive.
- File-system: This is a distributed file store that enables users to store data to a central location. Our Cluster has about 10 TBs of storage with 1GBps data transfer speed shared across all nodes and users.
- Scheduler: Our cluster uses the Slurm Scheduler - https://slurm.schedmd.com/

Slurm

Key Features of Slurm

- Scales to millions of cores and tens of thousands of GPGPUs
- Military grade security
- Heterogeneous platform support allowing users to take advantage of GPGPUs.
- Flexible plugin framework enables Slurm to meet complex customization requirements
- Topology aware job scheduling for maximum system utilization
- Open Source
- Extensive scheduling options including advanced reservations, suspend/resume, backfill, fair-share and preemptive scheduling for critical jobs
- No single point of failure

MLP Cluster Slurm Architecture Overview



MLP Cluster’s Partitions

To better manage jobs of different types, we partitioned our cluster into 4 partitions:

1. interactive -> max job length: 2 hours (2 nodes) -> Used only for interactively debugging your code.
2. short -> max job length: 4 hours (2 nodes) -> Used for short jobs up to 4 hours. (i.e. Small conv-nets)
3. standard -> 8 hours (15 nodes) -> The largest partition in the cluster, can run jobs for up to 8 hours at a time.
4. longjobs -> 3 days 8 hours (6 nodes) -> Used for very long jobs up to 3 days and 8 hours. This can be massive deep nets on very large datasets.

How do I get started?

1. Checkout the mlp_cluster_tutorial branch from the mlp github.
2. Follow the instructions in the quick-start-guide.
3. Read the README.md in the mlp_cluster_tutorial for further instructions and tips regarding the extra tools in the repo.

What’s a good workflow for developing cluster-ready code?

1. Implement your code locally, on a lab PC, personal laptop etc.
2. Test and debug locally, on CPU. Make sure your code is indeed running on your local machine, and note the speed (to compare with the GPU later).
3. Transfer your files to the cluster, using a command like rsync.
4. Test your script on the cluster’s GPUs, either using interactive sessions (not recommended), or using an sbatch command (recommended).
5. Never implement your code on the cluster. Running scripts on the cluster will often require queuing, which will reduce your productivity by 1-2 magnitudes. Moreover, issuing commands on the cluster, and modifying files will add minor computation and storage overheads on the file-system. The compound effect of 100 people doing this will be an extremely slow and unusable cluster. Nobody wants that.

Demo

1. Introduction of the mlp cluster branch
2. Slurm Demo Usage:
    a. Cheat Sheet - https://github.com/JIC-CSB/SLURM-cheat-sheet
    b. Partitions - Names, Uses, Technical Differences
    c. Single GPU experiments
    d. Multi GPU experiments

Good cluster etiquette

1. Remember that others on the cluster are fellow humans trying do to their work. No need for any kind of aggressive behaviour.
2. **Never** ever use **dynamic hard-drive sourced data loaders** using the **file-system as the hard-drive**. It is at least 3 magnitudes slower than loading from memory and 2 magnitudes slower than loading from the node's scratch drive. Also, it will slow down the cluster to the point where it might become unusable for others.
3. Make sure you are using **at most 12 GB of RAM**. This can be easily achieved by specifying 12GB as the amount of memory you'll request from slurm. We will explain how to do that next.
4. Use time-expiring jobs. This way the scheduler knows how much your jobs will take and can allocate resources more efficiently.

Additional Tips

1. In terms of cost effectiveness, the cluster is yielding the maximum value when being used at 100% capacity. At that point, queues will begin forming and the fair share system will take care of the rest.
2. Hence, if you see a user using 15 GPUs, there is no need to Witch-Hunt them. They are not commiting a crime. In fact, they are helping improve the cost effectiveness of the cluster.
3. We have tuned the fair share policies to ensure that people’s jobs will be treated fairly. However, for that to happen, you need to queue your jobs. Don’t wait for the queue list to shrink, your job’s priority increases as a function of how much time it has been in the queue (among other things).
4. If you feel that a particular user is somehow abusing the system, please simply raise your concerns in a private piazza post.

Fair Share System - How does slurm allocate resources?

Slurm has built in systems for fair share usage. In other words, depending on the total available compute, and how much a user has used, their jobs get ranked in a priority list. This means that even if one submits 10s of jobs, slurm will try to keep the usage fair among the users. The job priority formula is not fixed, it’s set by the cluster technicians. Below one can see the main factors.

Job_priority =
(PriorityWeightAge) * (age_factor) +
(PriorityWeightFairshare) * (fair-share_factor) +
(PriorityWeightJobSize) * (job_size_factor) +
(PriorityWeightPartition) * (partition_factor) +
(PriorityWeightQOS) * (QOS_factor) + ...

Job Failure Causes

1. Periodic unavailability of otherwise healthy nodes due to:
    a. Memory leaks
    b. Filesystem access breakdown
2. Jobs dying without informing the user due to:
    a. Random node errors (these happen on all types of machines).
    b. Node itself crashes
    c. Node is not crashed, but has an issue that kills jobs immediately upon submission.
3. Machines that can’t read from the file-system due to internal errors.

Overcoming problems that ‘catch you in your sleep’

Issues such as:

1. Jobs being killed without warning, without you knowing, until you observe the state (Schrodinger's cluster job?).
2. Jobs hanging forever without you being aware.
3. Jobs being allocated to nodes that fail or crash.

Can be easily circumvented using the provided python job runner script found in the cluster_experiment_scripts folder.

The scripts circumvent these problems by checking that your jobs keep running, re-submitting if necessary, keeping track of progress and making sure your experiments get done.

What to do if I have a question re: MLP cluster

1. Post on Piazza, let us know what the problem is.
2. Depending on the problem we might advise you to submit a computing support ticket using https://www.inf.ed.ac.uk/systems/support/form/
3. Alternatively, you can do both. However, it will allow the computing support to be more efficient if they only have to worry about actual system/hardware problems, instead of answering questions of the type “How can I run a job”.