SLURM GUIDE: AMD TEAM

Cluster structure and terms

Slurm consists of a slurmd daemon running on each compute node and a central slurmctld daemon running on a management node (with optional fail-over twin). The slurmd daemons provide fault-tolerant hierarchical communications.

A **node** in a SLURM-managed environment is a single computer within a larger group of computers (a cluster). Each node typically includes its own processors (CPUs), memory (RAM), and often local storage.

- Nodes are the fundamental compute resources where actual workloads (jobs, tasks) run
- Each node runs a Slurm **daemon** (slurmd), allowing it to communicate with the central Slurm controller and be managed as part of the cluster.
- Consists often of multiple GPUs / CPUs linked together

A **cluster** is a collection of interconnected nodes (computers) that work together under centralized management.

- The **cluster** acts as a unified resource pool, managed by software like SLURM, which schedules and allocates jobs across available nodes.
- Besides compute nodes, clusters may include login nodes, management nodes, and shared file servers.

A **directory** is a location in a file system where files and other directories are stored.

- Function: Directories organize data on disk, whether on a single node or across a shared file system accessible by multiple nodes in a cluster.
- Relation to Nodes/Clusters: Directories are not compute resources but are used to store input/output data, scripts, and results for jobs running on nodes or across the cluster

A **daemon** is a little helper that carries out and executes jobs. Entities managed by these Slurm daemons include **nodes**, the compute resource in Slurm, **partitions**, which group nodes into logical (possibly overlapping) sets, **jobs**, or allocations of resources assigned to a user for a specified amount of time, and **job steps**, which are sets of (possibly parallel) tasks within a job.

List of Commands:

I have attempted to list these in order of importance

squeue reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.

srun is used to submit a job for execution or initiate job steps in real time. srun has a wide variety of options to specify resource requirements, including: minimum and maximum node count, processor count, specific nodes to use or not use, and specific node characteristics (so much memory, disk space, certain required features, etc.). A job can contain multiple job steps executing sequentially or in parallel on independent or shared resources within the job's node allocation.

salloc is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.

sbatch is used to submit a job script for later execution. The script will typically contain one or more srun commands to launch parallel tasks.

sstat is used to get information about the resources utilized by a running job or job step.

scancel is used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.

sview is a graphical user interface to get and update state information for jobs, partitions, and nodes managed by Slurm.

sinfo reports the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options.

sacct is used to report job or job step accounting information about active or completed jobs.

sattach is used to attach standard input, output, and error plus signal capabilities to a currently running job or job step. One can attach to and detach from jobs multiple times.

sbcast is used to transfer a file from local disk to local disk on the nodes allocated to a job. This can be used to effectively use diskless compute nodes or provide improved performance relative to a shared file system.

scontrol is the administrative tool used to view and/or modify Slurm state. Note that many scontrol commands can only be executed as user root.

sprio is used to display a detailed view of the components affecting a job's priority.

sshare displays detailed information about fairshare usage on the cluster. Note that this is only viable when using the priority/multifactor plugin.

strigger is used to set, get or view event triggers. Event triggers include things such as nodes going down or jobs approaching their time limit.

OTHER BASH (not-SLURM) commands:
pwd: displays current working directory
ls: lists current working directory
echo: prints a string of text, or variable value, into the terminal
man <command>: lets us inspect a given command, and see how it works

Inspecting nodes

To inspect a node, and see the hardware on it, run: scontrol show node <node-name>
This way, you can see if the node is running any gpus or nah

> Nodes c00, c01, c02 all have MI210s - *we should use these nodes*
> I'm pretty sure we specify these by saying –gpus:mi210:1 [says we need one MI210, and slurm daemon allocates us one of these nodes as a result]

To inspect a job, use `scontrol show job`

Running Scripts

Scripts are 'bash scripts', which are meant to execute tasks. We create bash scripts to then carry out commands or tasks
https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/
Bash scripts *always* start with a *shebang statement* - in our case, it looks like *#!/usr/bin/bash* on the first line

We can set a number of parameters to change. See this example bash script for an example:

```
#!/bin/bash
#SBATCH -J matmul-py          #Job name
#SBATCH -o %x.%J.out          #Job output file
#SBATCH --error=%J.err        # File to save job's standard erro
#SBATCH -p debug              # -p sets the partition node - here is debug
#SBATCH --gpus=mi210:1 -N1 -n1 -c1          # 1 GPU, 1 node, 1 task, 1 cpu
#SBATCH -t 00:30:00            # Time 30 mins
#SBATCH --mem=32G             #We typically need quite a lot of memory lol
- our max is 80G
#memory blowup is real, esp with matrix multiplication.... #justsaying :3


set -x #enables verbose output
```

```
hostname #prints name of compute node we're running on
rocm-smi #ROCM command that shows diagnostic info about hardware

rocminfo | grep "Device Type" -A 5

module load py-torch/2.5.1-hip6.2.4 #Load torch rn

source pdhg_env/bin/activate #activate environment

# Run your Python script
python -u 'matmul.py'
```

Crucially, we load torch from the cluster's own 'version', and we also have our own python environment, with numpy pandas matplotlib etc all installed…

Currently, we can't run python files inside scratch folder, even if we can run scripts inside here

For more scripts, look in the github lol

Another script example, taken from https://nebius.com/blog/posts/model-pre-training/slurm-vs-k8s

```
# General job parameters: name, output file, time limit.
#SBATCH --job-name=my_job
#SBATCH --output=/home/bob/my_job.out
#SBATCH --time=10:00

# First allocation request: 2 nodes with 8 H100 GPUs on each.
#SBATCH --nodes=2
#SBATCH --gres=gpu:h100:8

# Second allocation request: 2 nodes with 8 A100 GPUs on each.
#SBATCH --nodes=2
#SBATCH --gres=gpu:a100:8

# Other resources, common for both allocation requests.
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=4G

# Print info about H100 GPUs.
# It runs the same command on 2 nodes with allocating 1 CPU,
# 4G of memory and 8 H100 GPUs on each.
srun --nodes=2 --cpus-per-task=1 --mem=4G --gres=gpu:h100:8 \
echo "Info about H100 GPUs on node $(hostname):" && \
nvidia-smi & # <- Continue without waiting

# The same step as above but for nodes with A100 GPUs.
srun --nodes=2 --cpus-per-task=1 --mem=4G --gres=gpu:a100:8 \
echo "Info about A100 GPUs on node $(hostname):" && \
nvidia-smi & # <- Continue without waiting

wait # Wait for the previous steps to complete.

# Run NCCL benchmarks on all 4 nodes with allocating 8 CPUs,
# 4G of memory for each CPU, and 8 GPUs of any model.
# Resource allocations for CPUs and memory aren't specified
# here so values from #SBATCH comments are used.
srun --nodes=4 --gpus=8 \
    echo "Run NCCL all-reduce benchmark:" && \
    /usr/bin/all_reduce_perf -b 512M -e 8G -f 2 -g 8
```