SLURM User Guide for CPU & GPU Servers

SLURM (**Simple Linux Utility for Resource Management**) is an **open-source workload manager and job scheduler** designed for **high-performance computing (HPC) clusters**.

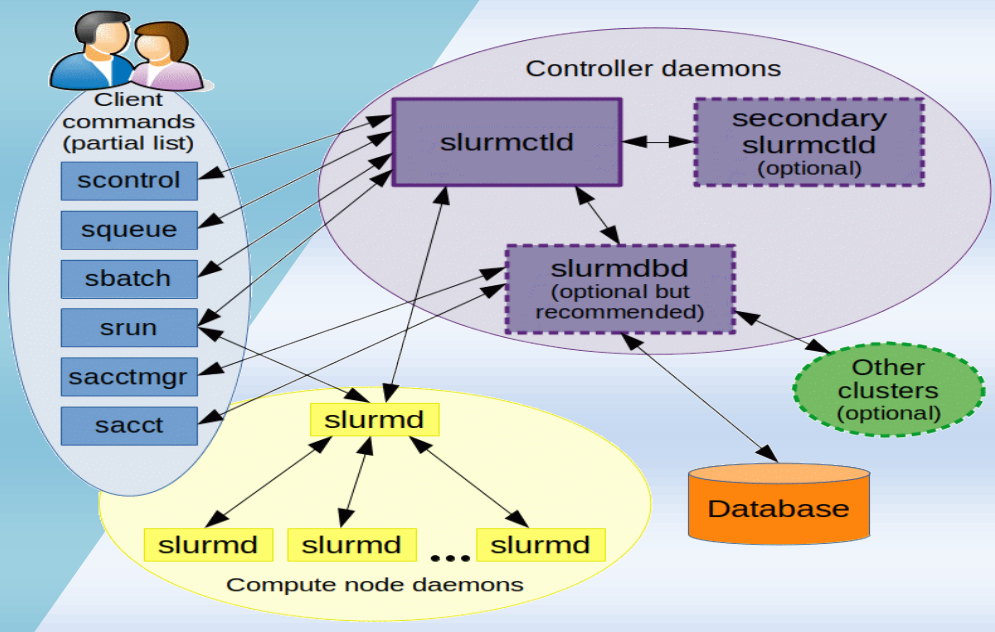
- Developed by Lawrence Livermore National Laboratory (LLNL)
- Widely used in High-Performance Computing (HPC) environments
- Capable of managing workloads on systems ranging from modest clusters to the most powerful computing servers.

Why SLURM?

- Efficient resource allocation
- Fair job scheduling
- Supports CPU, GPU, and memory management
- Open-source and highly configurable

SLURM Architecture

- **Controller (slurmctld):** Manages resources and job scheduling
- **Compute Nodes (slurmd):** Executes jobs
- **Database (slurmdbd):** Stores accounting info (optional)
- **User Commands:** Submit and manage jobs



Job Submission and Management on CPU & GPU Servers with SLURM

1. Basic SLURM Commands

Command	Description
sinfo	View available nodes and their states.
squeue	Check the job queue and running jobs.
sbatch job.sh	Submit a job script to SLURM.
scancel <jobid>	Cancel a running or pending job.
sacct	Check job accounting information.
srun	**U**sed to submit a job for execution or initiate job steps in real time.

2. Workflow for job submission

- Create job script
- Submit the job script with sbatch command
- If job submission is successful, you will see a job ID printed on the command line. Else, check the script and submit again
- Check job status with squeue command
- Use sacct, seff commands to check job information, if needed

3. Submitting a CPU Job

To run a job on CPU nodes, request CPU resources explicitly. Below is an example job script (cpu_job.sh):

```
#!/bin/bash
#SBATCH --job-name=cpu_test
#SBATCH --partition=cpu
#SBATCH --time=01:00:00  # 1 hour job
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=8G
#SBATCH --output=cpu_output.txt

python3 my_program.py
```

---

3. Submitting a GPU Job

To submit a job on GPU nodes, request GPU resources explicitly. Example (gpu_job.sh):

```
#!/bin/bash
#SBATCH --job-name=gpu_test
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --time=01:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=16G
#SBATCH --output=gpu_output.txt

module load cuda/11.8
python3 train_model.py4. Interactive Jobs
```

For debugging or testing, request an interactive session:

```
# For CPU interactive session
srun --partition=cpu --ntasks=1 --cpus-per-task=2 --mem=4G --
time=01:00:00 --pty bash

# For GPU interactive session
srun --partition=gpu --gres=gpu:1 --cpus-per-task=4 --mem=16G --
time=01:00:00 --pty bas
```

5. Monitoring Jobs

- Use `squeue -u <username>` to check your jobs.
- Use `sacct -j <jobid>` to see detailed job statistics.
- Job output and error files will be written as specified by `--output` and `--error` options.

6. Best Practices

- Always request only the resources you need (GPUs, CPUs, memory).
- Use `--time` to specify realistic job duration.
- Run short test jobs before submitting long training jobs.
- Store output files in your home or project directory.