Computação em Larga Escala

Slurm - Workload Manager

Eurico Pedrosa

António Rui Borges

Universidade de Aveiro - DETI

2025-04-13

Introduction to Slurm

What is Slurm?

- Initially known as **S**imple **L**inux **U**tility for **R**esource **M**anagement
- An open-source workload manager and job scheduler
- Designed for Linux clusters
- Simple, scalable, and fault-tolerant
- Responsible for queuing and executing jobs on shared computing resources
- A critical tool in high-performance computing (HPC) environments

Role in HPC: Key Functionalities

- **Resource Allocation**
  - Grants exclusive or shared access to compute nodes
  - Time-bound access for running user jobs
  - Ensures efficient use of hardware
- **Job Launch & Monitoring**
  - Starts and manages parallel jobs across nodes
  - Tracks job status and performance
  - Enables users to monitor job execution
- **Queue Management**
  - Maintains a queue of submitted jobs
  - Arbitrates access to computing resources
  - Applies scheduling policies to determine job order
  - Handles job priorities and dependencies

Who Uses Slurm?

- Academic and research HPC centers
- National laboratories
  - e.g., Lawrence Livermore National Lab (LLNL), NERSC
- Industrial HPC clusters
- Currently used on 65% of the world's top 500 supercomputers

Why It's Popular

- Scales efficiently from small clusters to national supercomputers
- Supports a large and active user community
- Offers robust documentation and support infrastructure

Background and Workflow

Origin & Development

- Initially developed at **Lawrence Livermore National Laboratory (LLNL)**
- Created around **2002** to replace proprietary job schedulers
- Now maintained by **SchedMD** and the open-source community
  - https://www.schedmd.com/

What Slurm Does

- Converts a cluster of networked Linux machines into a **unified batch computing resource**
- Manages how users interact with the cluster through **job submission**, not direct execution

Background and Workflow

Typical Usage Flow

- Users **do not run heavy computations** on login nodes
- Instead, they:
  - Write a **job script** defining resources and commands
  - Submit it to Slurm via commands like `sbatch`
  - Slurm finds and reserves resources to run the job efficiently

Goals of Slurm

- **Maximize cluster utilization**
- **Ensure fair resource sharing** among users
- **Automate job scheduling** based on defined policies

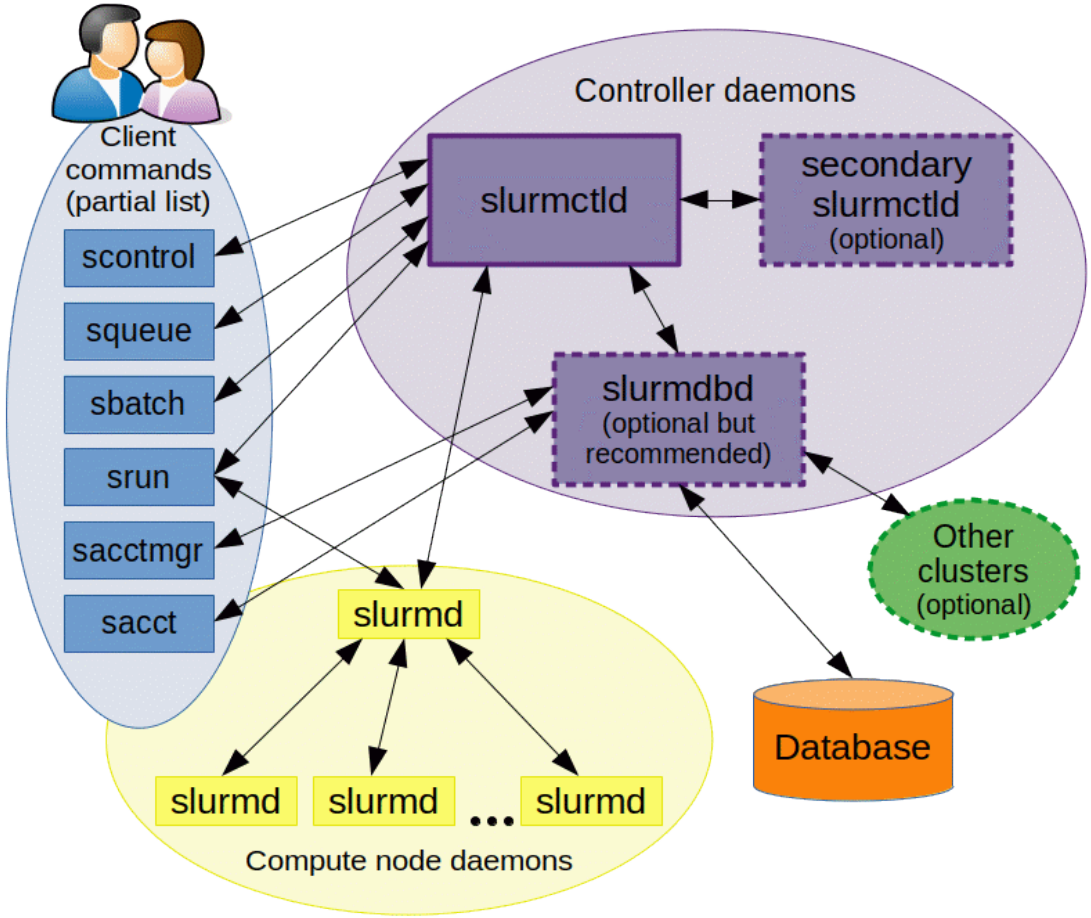
Architecture and Core Components

Architecture Overview

Slurm uses a **centralized controller / distributed agent** model

- A **central control node** runs the main management service
- Multiple **compute nodes** each run lightweight agents
- The system is designed for **scalability, fault tolerance, and modularity**

Architecture Overview



From https://slurm.schedmd.com/overview.html

Key Components

slurmctld – Controller Daemon

- Acts as the **central brain** of Slurm
- Runs on the **management/head/login node**
- Responsibilities:
  - Manages the **job queue** and **scheduling**
  - Allocates resources to jobs
  - Handles all user job requests (`sbatch`, `srun`, etc.)
- Supports a **backup controller** for failover and high availability
- Communicates with compute nodes via **RPC**

Key Components

slurmd – Node Daemon

- Runs on each **compute node**
- Acts as the **execution agent**
- Responsibilities:
  - Receives tasks from `slurmctld`
  - Launches and monitors job processes (like a **remote shell**)
  - Reports job status and resource usage
- Communicates hierarchically for **scalability** (tree-based message routing)

Key Components

slurmdbd – Database Daemon (Optional)

- Collects and stores **accounting and usage data**
- Used for:
  - **Job history**, **resource usage**, **user tracking**
  - **Fair-share scheduling**
  - **Reporting and analysis**
- Requires a backend database (e.g., **MySQL** or **MariaDB**)
- Supports **multi-cluster** accounting
- Interacts with `slurmctld` to log job data
  - **Source: slurm.schedmd.com**

Key Components

slurmrestd – REST API Daemon (Optional)

- Exposes a **RESTful API** for programmatic access
- Useful for:
  - **Web interfaces**
  - **Custom automation tools**
- Not required for basic usage

Authentication and Security

- Slurm uses **MUNGE** (MUNGE Uid 'N' Gid Emporium) for secure authentication
- Ensures:
  - Only **authorized users** can submit/control jobs
  - **Secure communication** between `slurmctld`, `slurmd`, and Slurm commands

System Interactions

Typical Job Workflow

- A user submits a job via commands like:
  - `sbatch` (batch job submission)
  - `srun` (interactive or parallel job execution)
- Workflow steps:
  - **1. User command contacts `slurmctld`**
    - The central controller evaluates policies and resource availability
  - **2. `slurmctld` assigns resources**
    - Determines which compute nodes will run the job
  - **3. `slurmctld` instructs `slurmd` daemons**
    - These daemons launch the job's processes on assigned nodes
  - **4. `slurmd` monitors execution**
    - Reports job start, status, completion, and errors back to `slurmctld`
  - **5. (Optional)** `slurmctld` logs job completion to `slurmdbd`
    - Used for accounting and reporting if enabled

System Interactions

Design Insight

- **Submission is decoupled from execution**
  - Users interact only with the controller (`slurmctld`)
  - Controller delegates execution to compute nodes (`slurmd`)

Configuration Files

slurm.conf – Main Configuration File

- Defines the cluster's architecture and policies
- Common configuration elements:
  - `ControlMachine=` – hostname of the Slurm controller
  - `NodeName=` – list of compute nodes and their properties
  - `PartitionName=` – defines partitions (aka queues) and node groupings
  - Scheduler policies (e.g., backfill, priority, fair-share)
  - Time limits, node states, authentication settings, etc.

Configuration Files

```
# Example slurm.conf snippet
ControlMachine=login-node1          # Host running slurmctld
NodeName=node[001-100] CPUs=32 State=UNKNOWN
PartitionName=debug Nodes=node[001-010] MaxTime=01:00:00 State=UP
PartitionName=main  Nodes=node[011-100] MaxTime=7-00:00:00 State=UP
```

Cluster Setup

- Total of **100 compute nodes**: `node001` to `node100`

Defined Partitions (Queues):

- **debug**
  - Nodes: `node001` to `node010`
  - Max time: **1 hour**
  - Purpose: **Short test/debug jobs**
- **main**
  - Nodes: `node011` to `node100`
  - Max time: **7 days**
  - Purpose: **Standard production workloads**

Configuration Files

What Are Partitions?

- Partitions in Slurm = **job queues**
- Used to:
  - Categorize nodes based on purpose, time limits, or hardware type
  - Apply different scheduling policies
- Users **must specify a partition** when submitting jobs, if multiple are defined

User Interaction

Submitting Jobs

Slurm handles job execution—**users don't log into compute nodes directly.** Instead, users submit jobs to the scheduler using specific commands.

Submitting Jobs

Key Job Submission Commands

- `sbatch` – **Submit a batch job**
  - Most common method for long-running jobs
  - Submits a **job script** (shell script with Slurm directives)
  - Script defines:
    - Resources (time, CPUs, memory, partition, etc.)
    - Commands to run (e.g., `srun`, `python`, `mpirun`)
  - Slurm queues and schedules the job
  - Ideal for:
    - **Production workloads**
    - **Reproducible runs**
  - **Source: hpc-wiki.info**

Submitting Jobs

- `srun` – **Run a job or command directly**
  - Used **inside a batch script** to launch tasks across allocated nodes
  - Can also be used **on the command line** for:
    - Quick tests
    - Launching MPI or OpenMP programs without a script
  - Requests resources and executes immediately (if available)
- `salloc` – **Request an interactive session**
  - Allocates resources and gives you a **shell** on compute nodes
  - Useful for:
    - **Interactive debugging**
    - **Testing GUI applications**
    - Running **interpreters** or **live tools** (e.g., Jupyter, MATLAB)
  - Job ends when session ends or time expires

Job Script Basics

- **Batch jobs in Slurm are driven by shell scripts**
- These scripts contain:
  - Slurm **directives** (prefixed with `#SBATCH`)
  - Regular **shell commands**
- Submitted with the `sbatch` command
  - Slurm queues and executes them when resources become available

Example: Simple Slurm Job Script

```
#!/bin/bash
#SBATCH --job-name=example_job        # Name of the job
#SBATCH --output=example_job.out      # Output file (stdout and
stderr)
#SBATCH --time=0-00:30:00             # Time limit (30 minutes)
#SBATCH --partition=debug             # Queue name
#SBATCH --nodes=1                     # Number of nodes
#SBATCH --ntasks=1                    # Number of tasks (processes)
#SBATCH --mem=1G                      # Memory per node

echo "Running on host: $HOSTNAME"
./my_program arg1 arg2                # Commands to run
```

Job Script Basics

Key Points

- `#SBATCH` lines are **job configuration options** for Slurm
  - They are **comments to the shell**, but parsed by Slurm
- Remaining lines are standard **bash commands**
- In this example:
  - **Job Name**: `example_job`
  - **Resources Requested**: 1 node, 1 task, 1 GB RAM, 30 minutes
  - **Partition**: `debug`
  - **Command Executed**: `./my_program arg1 arg2`

Job Submission Workflow

1. Save your script (e.g., `script.sh`)
2. Submit with:

   ```
   sbatch script.sh
   ```

3. Slurm queues the job
4. When the requested resources are free, the job runs on a **compute node**

Resource Requests in Slurm Job Scripts

When writing a job script, resource requests are specified using `#SBATCH` options. These inform Slurm how much and what kind of resources your job needs.

Key `#SBATCH` Parameters

- `--time=HH:MM:SS`
  - Maximum job runtime
  - Format: `days-hours:minutes:seconds` (e.g., `0-02:00:00` = 2 hours)
- `--mem=4G`
  - Memory **per node**
- `--mem-per-cpu=2G`
  - Memory **per CPU core** (alternative to `--mem`)
- `--nodes=2`
  - Number of compute nodes requested
- `--ntasks=4`
  - Total number of tasks (often = MPI processes)
- `--cpus-per-task=4`
  - Number of threads per task (for OpenMP or multithreaded apps)

Resource Requests in Slurm Job Scripts

Best Practices

✅ **Request accurately** based on your job's needs

❌ Don't over-request

- Requesting **too little** (e.g., not enough memory):
  - May cause job **failure or termination**
- Requesting **too much** (e.g., too many nodes, long time):
  - May result in **longer queue times**
  - Wastes resources and impacts overall system efficiency

💡 Tip: Start Small, Scale Up

- Run test jobs on fewer nodes/shorter time limits
- Use Slurm's job monitoring and logs to analyze actual resource use
  - Tools: `seff`, `sacct`, `sstat`

Job Queues & Monitoring

After submission, jobs enter a **queue** — called a **partition** in Slurm. Users can monitor, manage, and analyze their jobs using Slurm commands.

Monitoring Commands

- `squeue` — **View the job queue**
  - Shows jobs in the system with their status:
    - **PD** = Pending (waiting)
    - **R** = Running
  - Useful flags:
    - `squeue -u <username>` ⟶ your jobs
    - `squeue -j <jobid>` ⟶ a specific job

Job Queues & Monitoring

- `sinfo` — **Cluster and partition status**
  - Shows the state of nodes and partitions:
    - Up / Down
    - Idle / Allocated / Drained
  - Helps identify available resources or bottlenecks

Managing Jobs

- `scancel` — **Cancel a job**
  - Usage: `scancel <jobid>`
  - Stops a job that's pending or currently running

Job History and Stats

- `sacct` — **View completed job info**

Job Queues & Monitoring

- Available if **accounting is enabled**
- Shows:
  - **Start & end time**
  - **Runtime**
  - **Exit code**
  - **Memory and CPU usage**
- Useful for job optimization and debugging

MPI & Slurm

Running MPI Jobs with Slurm

Slurm is **designed with MPI in mind** — it handles resource allocation and process launching for distributed parallel jobs.

What is MPI?

- **MPI (Message Passing Interface)** enables parallelism across nodes via **network communication**
- Common in HPC workloads for:
  - Scientific simulations
  - Engineering computations
  - Large-scale data processing

How Slurm Supports MPI

- Slurm allocates:
  - The requested number of **nodes and cores**
  - Launches **MPI ranks** (processes) automatically
- Works seamlessly with `srun`, `mpirun`, or `mpiexec` (depending on the system setup)

Resource Allocation for MPI Jobs

Specify MPI resources using `#SBATCH` directives:

Option 1 – Total number of MPI ranks

```
#SBATCH --ntasks=16        # Run 16 MPI processes
```

Option 2 – Nodes and tasks per node

```
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4   # 4 MPI ranks per node
#SBATCH --ntasks=16           # Total = 4 x 4
```

CPU layout

- For **pure MPI**:

  ```
  #SBATCH --cpus-per-task=1
  ```

- For **hybrid MPI + OpenMP**: Set `--cpus-per-task` to the number of threads per MPI rank (e.g., `--cpus-per-task=8`)

Resource Allocation for MPI Jobs

Example Use Case

To run a 16-rank MPI job across 4 nodes, 4 ranks per node:

```
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
```

Launching MPI Programs in Slurm

There are two primary methods to start MPI jobs under Slurm.

Method 1: Using `srun` (Slurm-native & preferred)

Example Job Script:

```
#SBATCH --ntasks=16
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4

module load openmpi          # Load your MPI implementation
srun ./my_mpi_program arg1 arg2
```

Launching MPI Programs in Slurm

How It Works

- Slurm allocates **4 nodes**, with **4 MPI ranks** on each
- `srun` launches **16 total MPI processes**
- Slurm automatically:
  - Wires up MPI communication
  - Handles node list & rank distribution
  - Sets up process binding and affinity
  - Tracks job accounting & resource usage

Launching MPI Programs in Slurm

Why Use `srun`?

- **No hostfile needed** – Slurm knows the allocated nodes
- Uses **PMI (Process Management Interface)** for efficient MPI setup
- Ensures:
  - **CPU affinity and binding**
  - **Job accounting integration**
  - **Resource tracking**
- Preferred by most HPC centers for **reliability and portability**

Resources

1. **Slurm Official Documentation**
   - https://slurm.schedmd.com/documentation.html
2. **Slurm Quick Start Guide**
   - https://slurm.schedmd.com/quickstart.html
3. **Slurm Configuration Overview**
   - https://slurm.schedmd.com/slurm.conf.html
4. **Slurm Commands Reference**
   - https://slurm.schedmd.com/commands.html
5. **Slurm Presentations (Design & Architecture)**
   - https://slurm.schedmd.com/presentations.html
6. **University of Utah CHPC Slurm Guide**
   - https://www.chpc.utah.edu/documentation/software/slurm.php
7. **NASA NCCS Slurm Best Practices**
   - https://www.nccs.nasa.gov/docs/user-guide/slurm
8. **University of Michigan ARC Slurm Tutorial**
   - https://arc.umich.edu/slurm/
9. **University of Illinois Slurm Overview**
   - https://help.rc.uic.edu/kb/articles/slurm-job-scheduler-overview
10. **OpenMPI FAQ – Slurm Integration**
    - https://www.open-mpi.org/faq/?category=slurm
11. **Intel MPI with Slurm**
    - https://www.intel.com/content/www/us/en/developer/articles/technical/running-intel-mpi-applications-under-job-schedulers.html
12. **MPICH with Slurm**
    - https://wiki.mpich.org/mpich/index.php/Using_the_Slurm_Scheduler
13. **Slurm: Simple Linux Utility for Resource Management (LLNL Paper)**
    - https://computing.llnl.gov/sites/default/files/Slurm_Paper.pdf
14. **NERSC Slurm Documentation**
    - https://docs.nersc.gov/jobs/
15. **HPC University Slurm Tutorials**
    - http://www.hpcuniversity.org/tips/Slurm/
16. **TACC Slurm User Guide and Examples**
    - https://portal.tacc.utexas.edu/user-guides/slurm
17. **Slurm GitHub Repository**
    - https://github.com/SchedMD/slurm