Scheduling Basic Jobs

August 20, 2025

Brandon Reyes

Research Computing
UNIVERSITY OF COLORADO BOULDER

Slides & Exercises

https://github.com/ResearchComputing/hpc_fundamentals_micro_credential

- In “scheduling_jobs” directory

Learning Objectives

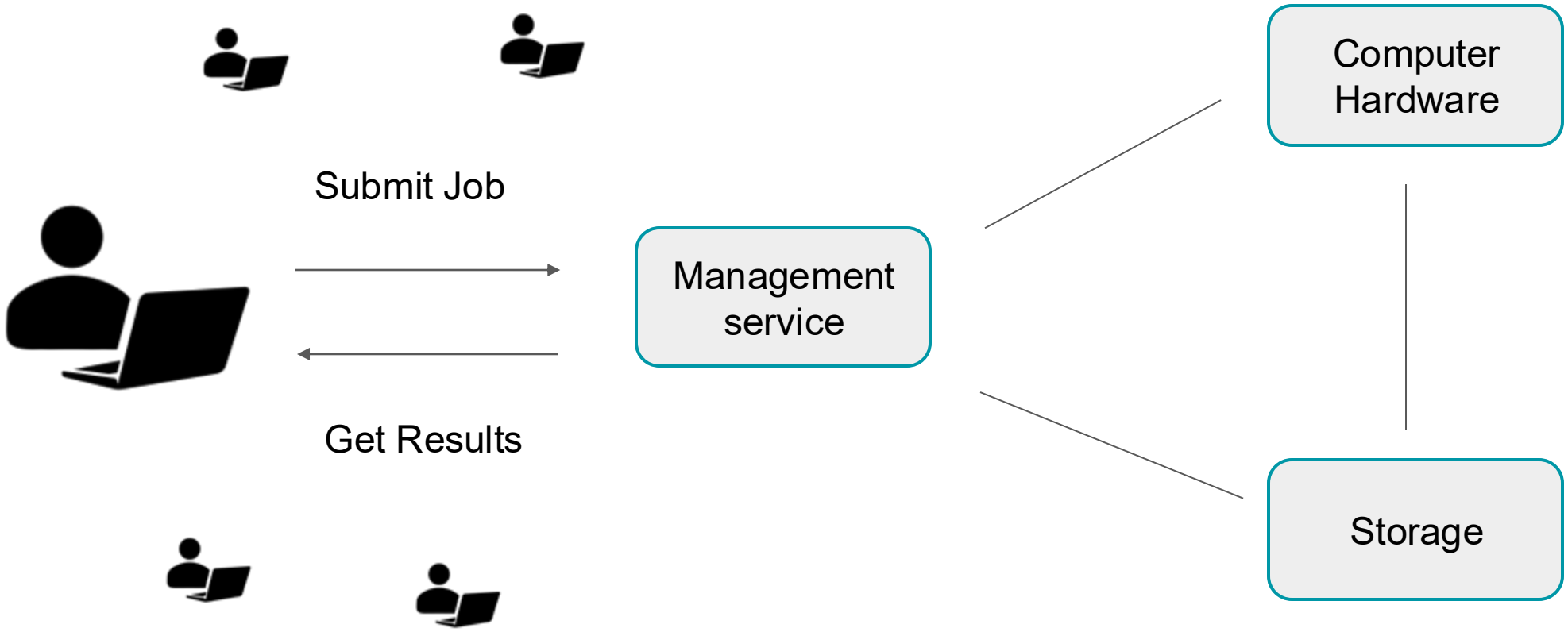
- Obtain a clear overview of job submission on an HPC system
- Learn about submitting both a batch and an interactive job to Alpine using the terminal

Session Overview

- General overview of Job submission
- Introduction to batch jobs
  - Submission of a batch job
- Checking/monitoring jobs
- Utilizing software in a job
- Introduction to interactive jobs

Remember, when you login to the HPC system you are put on a login node. You need to then gain access to a compute node to run software.

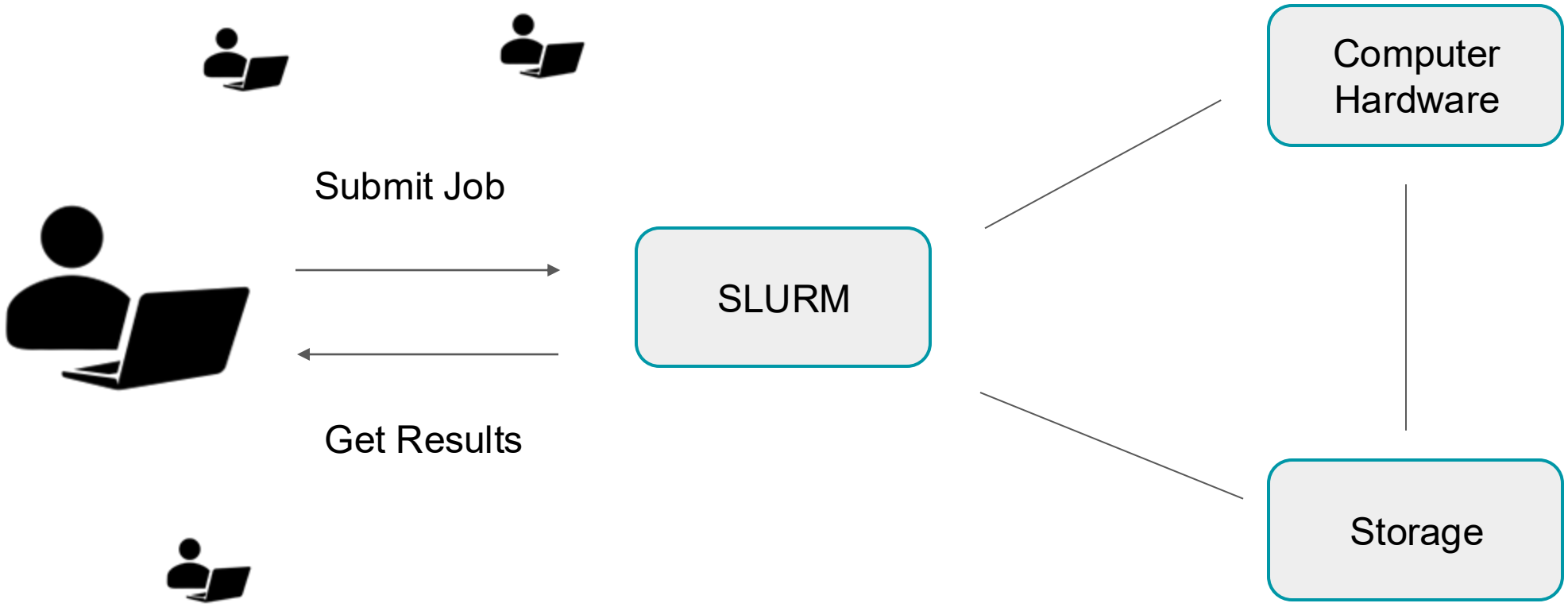
General Overview of Job Submission



Introduction to Jobs

- Because our clusters are shared resources with many users trying to utilize available compute with their applications, we need a system to divide compute in a simple and fair system
- SLURM
  - **S**imple **L**inux **U**tility for **R**esource **M**anagement
- Through SLURM, users can grab allotments of compute resources called Jobs
- 2 Types of Jobs
  - **Batch Jobs**
  - **Interactive Jobs**

General Overview of Job Submission



Job directives

- Directives are special flags that specify what type of HPC resources you would like to use
  - “I want to run on 1 CPU for 1 hour”
- Used for both interactive and batch jobs
- Can be specified via the command line or in a job script
- All directives shown are specific to SLURM

Common directives

- ★ **Partition – A collection of compute nodes**
  - `--partition=<partition_name>`
- ★ **Quality of service (QoS) – System defined constraints for a job (more on this later!)**
  - `--qos=<qos>`
- **Allocation – Account to "charge to"**
  - `--account=<account_name>`
- **Number of nodes to run on**
  - `--nodes=<nodes>`
- **Number of cores to run on**
  - `--ntasks=<number-of-tasks>`

★ - Must be specified for every job

Common directives

- ★ **Wall time – How long you want to run on these resources**
  - `--time=<wall time>`
- **Job name**
  - `--job-name=<jobname>`
- **Output – Where all output that would be written to the terminal should go**
  - `--output=<name>`
- **Send an email when events happen in the job**
  - `--mail-type=<type>`
- **Email address to send updates to**
  - `--mail-user=<user>`

★ - Must be specified for every job

Alpine Partitions

- Partitions are a collection of compute nodes e.g. computers with common characteristics

Partition	Description	# of nodes	RAM/core (GB)	cores/node	GPUs/node
amilan	General Compute Node: AMD Milan	406	3.75	32,48,64	0
ami100	GPU Node: 3x AMD MI100	8	3.75	64	3
aa100	GPU Node: 3x Nvidia A100	12	3.75	64	3
amem	High-memory node	24	21.5	48,64	0

Alpine Testing Partitions

- For many of the partitions we have testing partitions that provide quicker access to resources.

Note: there are restrictions on the amount of resources you can request. These restrictions can be found in our documentation.

Partition	Description	# of nodes	RAM/core (GB)	cores/node	GPUs/node
atesting	General Compute Node: AMD Milan	2	3.75	64	0
acompile	General Compute Node: AMD Milan	4	3.75	64	0
atesting_mi100	GPU Node: 3x AMD MI100	1	3.75	64	3
atesting_a100	GPU Node: 3x Nvidia A100	1	3.75	64	6 MIG instances

Quality of Service (QoS)

- Quality of Service specifies additional constraints for a job
  - On Alpine, QoS can be used to run long jobs, specify testing partitions, and select high-memory nodes

QoS	Description	Max wall time	Max jobs/user	Max nodes/user
normal	Default QoS	24 H	1000	128
long	For jobs needing longer wall times	7 D	200	20
mem	High-memory jobs	7 D	n/a	12
testing	For jobs submitted to testing partitions	1 H	1	n/a
compile	For jobs submitted to compile partition	12 H	1	n/a

Batch Jobs

- **Batch Jobs** are jobs you submit to the scheduler that are run later without supervision
  - By far the most common job on Alpine
  - Requires a job script
- A job script is simply a script that includes **SLURM directives** (resource specifics) ahead of any commands.

Anatomy of a job script

- It’s just a bash script with SLURM specific directives!

```
#!/bin/bash

## Directives
#SBATCH --<option>=<value>

## Software
module load <software>

## User scripting
<command>
```

Directives in a job script

```
#SBATCH --<option>=<value>
```

Example job script

```
#!/bin/bash

## Directives
#SBATCH --ntasks=1                          # Number of requested tasks/cores
#SBATCH --time=0:01:00                      # Max run time
#SBATCH --partition=amilan                   # Specify Alpine CPU node
#SBATCH --qos=normal                        # Specify QoS
#SBATCH --output=test_%j.out                # Rename standard output file

## Software
module purge                                # Purge all existing modules

## User commands
echo "This is a test submitted by $USER"
```

Submitting a Job script

- Once a job script has been constructed you must submit it to the HPC system using SLURM
  - Done using `sbatch`
- If we created the job script “my_first_job.sh” then we would submit it as follows:

```
sbatch /path/to/my_first_job.sh
```

Job output

- Once a job completes its execution, the standard output of the script will be redirected to an output file.
  - Great for debugging!
  - Could be different from output generated by your application
  - File is created in directory job was run unless specified in your --output directive.
  - If the directive --output is not provided, then a generic file name will be used (slurm_xxxxxx.out).

Checking your jobs

- squeue: Monitor your jobs status **in queue and while running**:
  - By default, shows all jobs in queue can specify using:

```
$ squeue -u <username>
$ squeue -p <partition>
```

- sacct: Check back on usage statistics of **previous Jobs**
  - By default, only checks all jobs from the start of the current day can specify using:

```
$ sacct -u <username>
$ sacct --start=MM/DD/YY -u <username>
$ sacct -j <job-id>
```

Checking your jobs

- Another method of checking details of your job while running is with `scontrol`
  - Advanced command usually used by system administrators, but you can use it too!

```
$ scontrol show job <job number>
```

- To check the percentage of CPU and memory usage of a job **after it completes,** use seff

```
$ seff <job number>
```

Software and Jobs

- Okay so running a job is easy, but how do I run a job with my software?
- We can utilize all the software we discussed in the previous talk in the job script!
- Any non-GUI related commands you would run from the command line can be put in your job script!
  - If you would like to use GUI applications, you will need X11 forwarding and an interactive job

Software job script example

```
#!/bin/bash

## Directives
#SBATCH --ntasks=1                          # Number of requested tasks/cores
#SBATCH --time=0:01:00                      # Max run time
#SBATCH --partition=amilan                   # Specify Alpine CPU node
#SBATCH --qos=normal                        # Specify QoS
#SBATCH --output=test_%j.out                # Rename standard output file

## Software
module purge                                 # Purge all existing modules
module load anaconda                         # Load Anaconda
conda activate <my-conda-environment>        # Activate CONDA environment

## Run Python script
python my_cool_script.py
```

Interactive jobs

- Interactive jobs are used to gain access to a compute node in real time
  - Great for testing and debugging!
- Interactive jobs can be subject to normal wait times!
- We can get access to a compute node interactively with `sinteractive`
- We also have a specialized command called acompile that provides you with an interactive session
  - Access to quick resources
  - Limit of 4 CPUs for up to 12 hours

Running an interactive job

- Here we will run a Python script using an interactive job
- First request resources:

```
$ sinteractive --partition=acompile --qos=compile --time=00:10:00
```

- When the job starts you will be put on a compute node and you can execute your commands:

```
$ module load anaconda
$ python my_very_cool_script.py
```

- To quit:

```
$ exit
```

Interactive GUI job

- If you have software that has a Graphical User Interface (GUI) it is sometimes possible to display that GUI on your local machine.
  - This is done using X11 forwarding
  - This GUI may be laggy if you are on a compute node or if your connection is unstable
  - See our documentation: https://curc.readthedocs.io/en/latest/running-jobs/interactive-jobs.html#interactive-gui-applications

Note: you can also use our Open OnDemand “Core Desktop” application to display a GUI. This tends to be more stable than X11 forwarding.

Thank you!