Batch Job / Workflow Management: SLURM & Snakemake

SubMIT Workshop
Jan 23, 2026

Alex Avdoshkin, Project Team

What I'll cover

Slurm

- Login node vs compute nodes
- Core commands: sinfo, squeue, sbatch, srun/salloc, scancel
- Resource requests: time, CPUs, memory (and GPUs)

Snakemake

- Snakefile basics: rules, inputs/outputs, wildcards
- Workflow DAG
- Profile and Slurm

```
Find more info in
subMIT User's Guide:
submit.mit.edu/submit-
users-guide
```

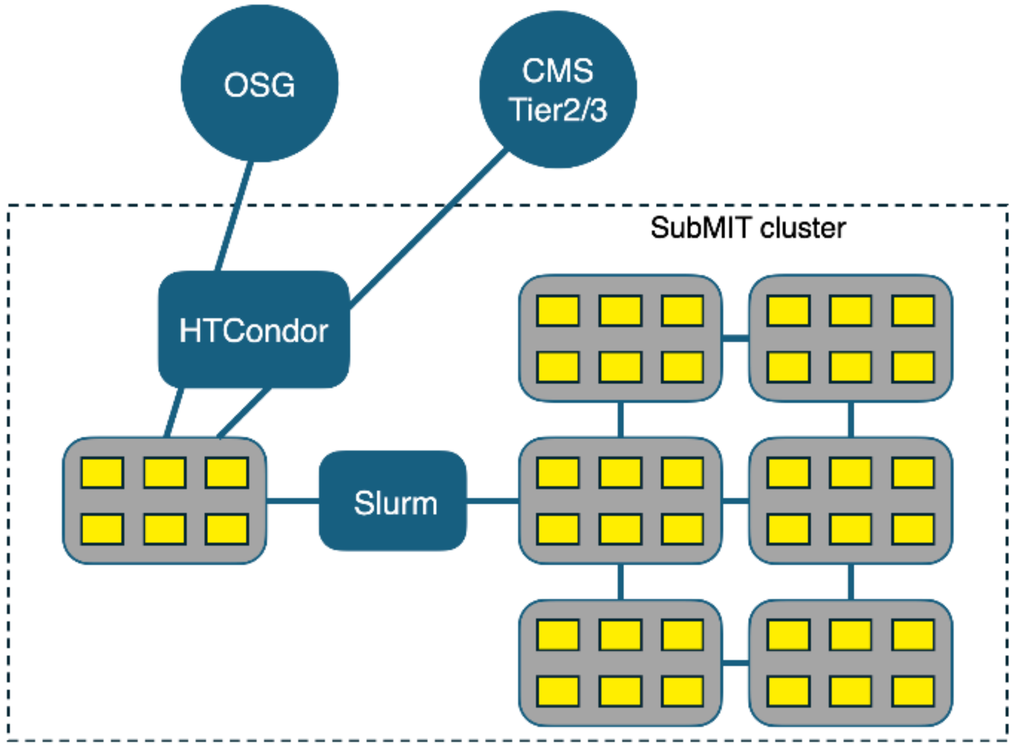
Slurm in one slide

Functions

- Allocate (exclusive/non-exclusive) access to compute resources
- Start/monitor work on allocated nodes
- Arbitrate contention via a queue + priority policies

Architecture overview:

- slurmctld: controller (scheduler brain)
- slurmdbd: database daemon (bookkeeping)
- slurmd: daemon on each compute node (runs tasks)



`See subMIT Tutorial 2: submit.mit.edu/submit-users-guide/tutorials/tutorial_2.html`

Slurm vocabulary

Key concepts

- Partition: a set of nodes + scheduling rules
- Node: a machine providing CPUs/GPUs
- Job: your request for resources + a script/ command to run

Job life cycle

- Jobs have priority. It depends on user's use history
- A job needs to fit with what's available to start
- Small jobs can start early because of "backfill"

Slurm commands

Info/queue:

sinfo	partitions + node availability
squeue	what's running/ pending
squeue -u $USER	user's jobs only
sacct	job history

Run/control:

sbatch	submit a batch script
srun	run a command
salloc	request interactive allocation
scancel	cancel a job

sbatch scripts

Example job.sbatch

```
1 #!/bin/bash
2 #SBATCH —job-name=train
3 #SBATCH --partition=submit
4 #SBATCH --time=02:00:00
5 #SBATCH --cpus-per-task=8
6 #SBATCH --mem=16G
7 #SBATCH --output=logs/out
8 #SBATCH --error=logs/err
9
11 module load your_toolchain
12
13 input=data/sample.fq.gz
14 output=results/sample.bam
15
16 echo "Running on $(hostname)"
17 my_aligner -t $SLURM_CPUS_PER_TASK -i $input -o $output
```

Call as

```
call as $sbatch job.sbatch
```

Run an array

```
$sbatch —array=0-99 job.sbatch
```

Managing dependencies

```
jid1=$(sbatch --parsable jobA.sbatch)
jid2=$(sbatch --parsable
—dependency=afterok:$jid1 jobB.sbatch)

sbatch --dependency=afterok:$jid2 jobC.sbatch
```

Requesting resources

What you usually specify

- --time=HH:MM:SS (wall time)
- --cpus-per-task=N (threads)
- --mem=16G or --mem-per-cpu=2G
- --partition=...
- —gres=gpu:1

Constraints

`--constraint=…`
request nodes with a feature, e.g. *nvidia_a30, 100gbs*, …
Use sinfo -o "%N %f" to see the full list

Partitions

- submit
- submit-gpu
- submit-gpu-express

~2000 CPUs and ~60 GPUs

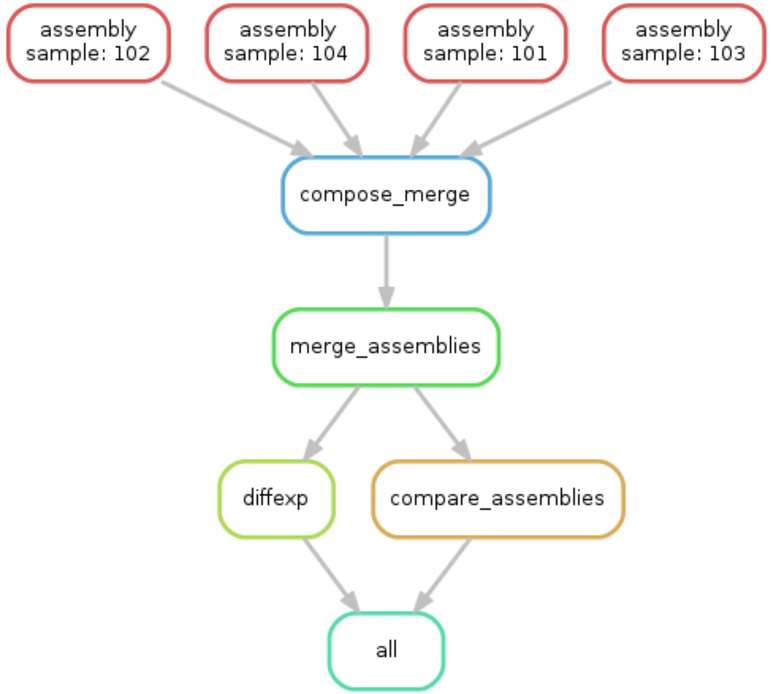
Our nodes (even within in partitions) are heterogenues, differing in the numbers of cores, memory, types of GPUs

Snakemake

You specify results -> Snakemake figures out the steps

Benefits

- Reproducibility: same commands, same inputs → same outputs
- Incremental builds: only rerun what's missing or outdated
- Parallelism: run independent rules concurrently
- Portability: local laptop → Slurm cluster with minimal changes



Snakemake builds a DAG of jobs.
Wildcards (like {sample}) fan out parallel work.

See subMIT Tutorial 7: submit.mit.edu/submit-users-guide/tutorials/tutorial_7.html

Snakefile

Wildcards, rules and all

Snakefile example

```
configfile: “config.yaml"

SAMPLES = config[“samples"]

rule all:
    input:
        expand("results/{sample}.count", sample=SAMPLES)

rule count_lines:
    input:
        "data/{sample}.txt"
    output:
        "results/{sample}.count"
    threads: 1
    resources:
        mem_mb=200
    shell:
        "wc -l {input} > {output}"
```

Mental model

- Rules are templates; wildcards match filenames
- `rule all` defines the targets you want built
- Snakemake runs rules only when outputs are missing/outdated

Best practice: parameterize paths + options in a config.yaml, keep Snakefile logic clean.

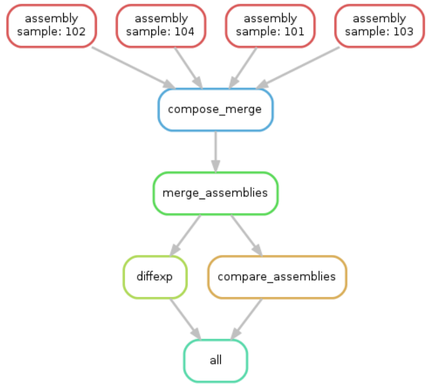
Running Snakmake

Run

```
$ snakemake --cores <number of cores>
```

Build DAG

```
$ snakemake --dag | dot -Tpng > dag.png
```



Run with profile

```
$ snakemake --profile ./slurm_simple
```

profile file

```
# ./slurm_simple/config.yml
cluster:
  mkdir -p slurm_logs/{rule} &&
  sbatch
    --partition={resources.partition}
    --job-name=smk-{rule}-{wildcards}
    --output=slurm_logs/{rule}/{rule}-{wildcards}-%j.out
    --error=slurm_logs/{rule}/{rule}-{wildcards}-%j.err
    --time={resources.time}
    --account=<your submit username>
    --mem={resources.mem_mb}
default-resources:
  - partition=submit
  - time="48:00:00"
  - mem_mb=2000
restart-times: 1
max-status-checks-per-second: 1
latency-wait: 3600
jobs: 5000
keep-going: True
rerun-incomplete: True
printshellcmds: True
use-conda: True
```