Running Jobs

How to submit, monitor, and manage jobs on the TAU HPC cluster.

- Slurm Basics
  - Submitting a Batch Job
  - Partitions, QoS & Preemption
  - Interactive Jobs
  - Managing & Monitoring Jobs
  - Memory & Resource Estimation
  - GPUs, Constraints & Features

Slurm Basics

Core Slurm concepts — batch jobs, interactive sessions, monitoring, and memory.

Submitting a Batch Job

Use `sbatch` to submit a script for batch processing. Your script runs when resources become available — you don't need to stay connected.

Basic Job Script

```
#!/bin/bash
#SBATCH --job-name=my_job                          # Job name
#SBATCH --account=public-users_v2                  # Account name
#SBATCH --partition=power-general-shared-pool      # Partition name
#SBATCH --qos=public                               # QOS type
#SBATCH --time=02:00:00                            # Max run time (hh:mm:ss)
#SBATCH --ntasks=1                                 # Number of tasks
#SBATCH --nodes=1                                  # Number of nodes
#SBATCH --cpus-per-task=10                         # CPU cores required
#SBATCH --mem-per-cpu=4G                           # Memory per CPU
#SBATCH --output=my_job_%j.out                     # Output file (%j = job ID)
#SBATCH --error=my_job_%j.err                      # Error file
#SBATCH --mail-user=your@email.tau.ac.il           # Email address
#SBATCH --mail-type=END,FAIL                       # Notify on completion or failure

module purge
module load gcc/rocky9-gcc-fortran-14.2.0

# Your commands here
./my_program
```

Submit it:

```
sbatch my_job.sh
```

*The maximum time an sbatch job can run is 7 days #SBATCH --time=7-00:00:00

Job Arrays

If you need to submit many similar jobs, use a job array instead of separate `sbatch` calls. Arrays reduce scheduler load and are easier to manage.

```
#!/bin/bash
#SBATCH --job-name=array_job
#SBATCH --account=public-users_v2
#SBATCH --partition=power-general-shared-pool
#SBATCH --qos=public
#SBATCH --time=02:00:00
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=4G
#SBATCH --array=1-100                          # 100 tasks
#SBATCH --output=array_job_%A_%a.out           # %A = job ID, %a = task ID
#SBATCH --error=array_job_%A_%a.err

# Use SLURM_ARRAY_TASK_ID to differentiate tasks
./my_program input_${SLURM_ARRAY_TASK_ID}.txt
```

Each task runs independently with a different `SLURM_ARRAY_TASK_ID`. Output and error logs are separated per task.

Excluding Nodes

To exclude specific nodes from your job:

```
#SBATCH --exclude=compute-0-[100-103],compute-0-67
```

Chain Jobs

Use `--depend` to run a job only after another completes:

```
sbatch --depend=afterok:45001 next_step.sh
```

Partitions, QoS & Preemption

Each partition (or "pool") has several QoS tiers that determine job priority and preemption behavior. To see which partitions, accounts and QoS you can use, run `check_my_partitions` on the login node (see Finding Your Account and Partition).

QoS types

QoS	Purpose	Preempts	Can be preempted by
Share-type QoS (e.g. `0.125_48c_8g`, `0.75_48c_8g`)	For multi-owner pools; defines each owner's guaranteed slice (CPU/GPU portion).	`owner`, `public`	--
`owner`	Used on your lab's pool to run above your guaranteed slice (higher than `public`).	`public`	Share-type QoS
`public` (partition: `power-general-shared-pool`)	Used on cluster-wide shared pools for friendly or opportunistic runs.	--	`owner`, share-type QoS
`public` (partition: `power-general-public-pool`)	Used on a small, cluster-wide shared group of nodes. Not preemptable.	--	--

Preemption rules

Share-type QoS > `owner` > `public`

- A share-type QoS job can preempt `owner` or `public` jobs on the same pool.
- An `owner` job can preempt `public` jobs.
- `public` jobs cannot preempt any other jobs.

Billing and priority

Each user has a "billing" value, calculated from the amount of resources they request. When you ask for more memory, your billing increases.

Billing affects priority: users who asked for fewer resources in the past get higher priority in the future.

Submitting with the correct QoS

Below are examples of how to use each QoS tier with your account.

Owner QoS (on your lab's pool):

```
sbatch -A UIDHERE-users_v2 -p UIDHERE-pool --qos=owner --time=02:00:00 run.sh
```

Share-type QoS (on a multi-owner pool, for your guaranteed slice):

```
sbatch -A UIDHERE-users_v2 -p gpu-LAB1-LAB2-pool --qos=0.125_48c_8g --gres=gpu:A100:1 run.sh
```

Public QoS (friendly, cluster-wide):

```
sbatch -A public-users_v2 -p power-general-shared-pool --qos=public --time=01:00:00 run.sh
```

Small, protected CPU pool:

```
sbatch -A public-users_v2 -p power-general-public-pool --qos=public --time=01:00:00 run.sh
```

Handy checks

Monitor your jobs and see their QoS and pending reasons:

```
squeue --me -O "JOBID,ACCOUNT,PARTITION,QOS,STATE,REASON"
sprio -w
```

If your job was preempted, check:

```
sacct -j <jobid> --format=JobID,State,Reason
```

Interactive Jobs

Interactive jobs give you a shell directly on a compute node — useful for testing, debugging, or running applications that need direct input.

Basic Interactive Session

```
srun --ntasks=1 -p power-general-shared-pool -A public-users_v2 --qos=public --pty bash
```

On a Specific Node

```
srun --ntasks=1 -p power-general-shared-pool -A public-users_v2 --qos=public --
nodelist=compute-0-12 --pty bash
```

With GUI (X11 Forwarding)

Make sure you connected to the login node with `ssh -X` first:

```
srun --ntasks=1 -p power-general-shared-pool -A public-users_v2 --qos=public --x11 --pty bash
```

With Multiple CPUs

```
srun --ntasks=1 --cpus-per-task=4 -p power-general-shared-pool -A public-users_v2 --qos=public
--nodes=1 --pty bash
```

Important

- Interactive jobs consume resources for as long as your session is open — exit when done
- If your connection drops, the interactive job will terminate
- For long-running work, use `sbatch` instead

Managing & Monitoring Jobs

Once your job is submitted, use these commands to track and manage it.

Checking Job Status

```
# Your jobs only
squeue -u your_username

# All jobs in the cluster
squeue

# Specific job
squeue -j job_id
```

Job States

State	Meaning
PD	Pending — waiting for resources
R	Running
CG	Completing
CD	Completed
F	Failed
OOM	Out of memory
TO	Timed out

Job Details

```
# Full details of a job
scontrol show job job_id

# Job accounting and resource usage
sacct -j job_id --format=JobID,JobName,State,MaxRSS,Elapsed
```

Cancelling Jobs

```
# Cancel a specific job
scancel job_id

# Cancel all your jobs
scancel -u your_username

# Cancel a specific array task
scancel job_id_task_id
```

Attaching to a Running Job

Use `sattach` to attach to a running job's input/output streams in real time:

```
# Attach to a job
sattach job_id

# Attach to a specific job step
sattach job_id.step_id
```

For non-interactive jobs this behaves like `tail -f`. For interactive jobs you can provide input directly.

To find job steps:

```
scontrol show job job_id
```

Look for **StepId** in the output.

Memory & Resource Estimation

Specifying the right amount of memory is important. Too little and your job fails with OOM. Too much and you block resources from other users.

Memory Directives

```
#SBATCH --mem=4G           # Total memory for the entire job
#SBATCH --mem-per-cpu=2G   # Memory per CPU core (use one or the other, not both)
```

Prefer `--mem-per-cpu` when your job scales with CPU count. Use `--mem` for fixed memory requirements.

Checking Memory Usage of Past Jobs

```
sacct -j job_id --format=JobID,JobName,MaxRSS,Elapsed
```

`MaxRSS` is the peak memory used. Use this to tune future submissions.

Monitoring a Running Job

```
#!/bin/bash
#SBATCH --job-name=memory_test
#SBATCH --account=public-users_v2
#SBATCH --partition=power-general-shared-pool
```

```
#SBATCH --qos=public
#SBATCH --time=01:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=4G
#SBATCH --output=memory_test.out

echo "Memory before:"
free -m

./your_application

echo "Memory after:"
free -m
```

Tips for Estimating Memory

- Start conservative, check `MaxRSS` via `sacct`, tune upward
- Check application documentation for memory recommendations
- Run a small test job first before scaling up
- Use `free -m`, `top`, or `htop` inside an interactive job to observe live usage
- Plan for peak usage — memory spikes during data loading or processing

OOM Error

If your job fails with out-of-memory, you'll see:

```
sacct -j job_id -o JobID,JobName,State%20

JobID    JobName               State
-------- ------------------- --------------------
71       my_job        OUT_OF_MEMORY
```

Resubmit with a higher `--mem` or `--mem-per-cpu` value.

GPUs, Constraints & Features

Beyond standard CPU/memory resources, the cluster provides special resources you can request via `--gres` and `--constraint`.

GPU Jobs

To request a GPU, use the `gpu-general-pool` partition and add `--gres=gpu:1`:

```
#!/bin/bash
#SBATCH --job-name=gpu_job
#SBATCH --account=my_account
#SBATCH --partition=gpu-general-pool
#SBATCH --qos=my_qos
#SBATCH --time=02:00:00
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=10
#SBATCH --gres=gpu:1
#SBATCH --mem-per-cpu=4G
#SBATCH --output=gpu_job_%j.out
#SBATCH --error=gpu_job_%j.err

module purge
module load python/python-3.8

# Your GPU commands here
```

Constraints

Constraints let you target nodes with specific hardware. Add `--constraint=<feature>` to your job script:

```
#SBATCH --constraint=localscratch,amd
```

Multiple constraints are comma-separated — the job will only run on nodes matching all of them.

Available Features

Run `features` on the login node to see the current list. Current features:

Feature	Description
`Af3`	Nodes that can run AlphaFold3
`AMD`	Nodes with AMD CPU family
`avx`	Nodes with AVX CPU capabilities
`localscratch`	Nodes with at least 700GB local `/localscratch`
`ib_bh`	Nodes with InfiniBand network #1
`ib_dj`	Nodes with InfiniBand network #2
`Intel`	Nodes with Intel CPU family

Using Local Scratch

If your job uses `/localscratch`, you must clean up after yourself — space is shared across all jobs on that node. Add this to your script:

```
export CACHEDIR=/localscratch/${USER}_${SLURM_JOB_ID}
mkdir -p $CACHEDIR

cleanup() {
  rm -rf -- "$CACHEDIR" || true
}
trap cleanup EXIT INT TERM HUP
```

The `trap` ensures cleanup runs even if the job fails or is cancelled.

In order to add /localscratch feature to your job request, please add this parameter to your job's parameters:

```
#SBATCH --constraint=localscratch
```

Available GRES Types

```
GresTypes=gpu,amd,af3,intel,localscratch
```