Submitting Jobs on HPC with SLURM

02-SLURM and the Cluster

Setup assumptions: You have SSH access to the cluster and R is available via modules or a system install. Replace partition/account names with your site's values.

---

0) Warm-up: Explore the cluster

Question. What partitions/queues can you use, and what's the default time limit and CPU/memory policy?

Show solution

```
# Partitions and key limits
sinfo -o '%P %l %c %m %a'   # Partition, time limit, CPUs, memory, availability

# Default Slurm config snippets
scontrol show config | egrep 'DefMemPerCPU|DefMemPerNode|SchedulerType|SelectType'

# Your account/QoS
sacctmgr show assoc user=$USER format=Cluster,Account,Partition,MaxWall,MaxCPUs,MaxJobs,Grp*
```

Notes: `sinfo` gives active partitions. `scontrol show config` reveals defaults like memory policy per CPU or per node.

---

1) Submit a minimal R job (single task)

Question. Write a Slurm batch script that runs a one-liner in R and writes to `slurm-%j.out`.

Show solution

```
cat > r_minimal.sbatch <<'SB'
#!/usr/bin/env bash
#SBATCH -J rmin
#SBATCH -p short              # <-- replace with your partition
#SBATCH -t 00:05:00
#SBATCH -c 1
#SBATCH --mem=1G
#SBATCH -o slurm-%j.out

module load R/4.3.2 2>/dev/null || true  # or: module load R; or skip if R in PATH
Rscript -e 'cat("Hello from R on Slurm!\n")'
SB

sbatch r_minimal.sbatch
```

`%j` expands to the JobID. Inspect output with `tail -f slurm-<jobid>.out`.

---

2) Parameterized simulation in R (script)

Question. Create `sim.R` that accepts command-line args: `n`, `mu`, `sigma`, `seed`, runs a simple simulation (e.g., mean of rnorm), and writes a CSV line to `results/sim_<seed>.csv`.

Show solution

```
mkdir -p results
cat > sim.R <<'RS'
args <- commandArgs(trailingOnly = TRUE)
stopifnot(length(args) == 4)

n     <- as.integer(args[1])
mu    <- as.numeric(args[2])
sigma <- as.numeric(args[3])
seed  <- as.integer(args[4])
```

```
set.seed(seed)
x <- rnorm(n, mu, sigma)
res <- data.frame(n=n, mu=mu, sigma=sigma, seed=seed,
                  mean=mean(x), sd=sd(x))

out <- sprintf('results/sim_%d.csv', seed)
write.csv(res, out, row.names = FALSE)
cat('Wrote', out, '\n')
RS
```

---

3) Batch script that passes parameters to R

Question. Write `sim.sbatch` that runs `Rscript sim.R 10000 0 1 42`.

Show solution

```
cat > sim.sbatch <<'SB'
#!/usr/bin/env bash
#SBATCH -J sim
#SBATCH -p short
#SBATCH -t 00:05:00
#SBATCH -c 1
#SBATCH --mem=2G
#SBATCH -o slurm-%j.out

module load R/4.3.2 2>/dev/null || true
Rscript sim.R 10000 0 1 42
SB

sbatch sim.sbatch
```

---

4) Create a parameter grid in R with `expand.grid`

Question. Use R to create a grid of `(n, mu, sigma, seed)` and save it as `params.csv` for an array job (no header, one row per task).

Show solution

```
Rscript - <<'RS'
params <- expand.grid(
  n    = c(1e4, 5e4),
  mu   = c(0, 0.2),
  sigma= c(1, 2),
  seed = 1:8
)
# optional shuffle for load balance
set.seed(1); params <- params[sample(nrow(params)), ]
write.table(params, file='params.csv', sep=',', row.names=FALSE, col.names=FALSE)
cat(nrow(params), 'parameter rows written to params.csv\n')
RS

head -n 5 params.csv
wc -l params.csv
```

Notes: We removed headers so each array index can read the `SLURM_ARRAY_TASK_ID`-th line directly.

---

5) Array job to run the grid

Question. Write `sim_array.sbatch` that runs one row of `params.csv` per array task and writes logs as `slurm-%A_%a.out`.

Show solution

```
cat > sim_array.sbatch <<'SB'
#!/usr/bin/env bash
#SBATCH -J simarr
#SBATCH -p short
#SBATCH -t 00:10:00
#SBATCH -c 1
#SBATCH --mem=2G
#SBATCH -o slurm-%A_%a.out
#SBATCH --array=1-32           # <-- set to nrow(params.csv)

module load R/4.3.2 2>/dev/null || true

# Read the line matching this array index
```

```
IFS=',' read -r n mu sigma seed < <(sed -n "${SLURM_ARRAY_TASK_ID}p" params.csv)

echo "Task ${SLURM_ARRAY_TASK_ID}: n=$n mu=$mu sigma=$sigma seed=$seed"
Rscript sim.R "$n" "$mu" "$sigma" "$seed"
SB

# Submit after matching the array range to your params
lines=$(wc -l < params.csv)
sbatch --array=1-$lines sim_array.sbatch
```

Key env vars: `SLURM_ARRAY_TASK_ID` (index), `%A` (ArrayJobID), `%a` (TaskID) for log naming.

---

6) Monitor, inspect, and cancel jobs

Question. How do you check running jobs and see finished job states? Cancel a stuck task 5 of an array.

Show solution

```
# Running/queued jobs for you
squeue -u $USER -o '%A %j %t %M %D %R'

# Completed job accounting (after finish)
sacct -u $USER --starttime today \
  --format=JobID,JobName%30,State,Elapsed,MaxRSS,ExitCode

# Show details for a specific job
a_jobid=123456
scontrol show job $a_jobid | less

# Cancel a whole job or a single array task
scancel 123456          # whole job
scancel 123456_5        # only task 5
```

Use `tail -f slurm-123456_5.out` to live-watch a specific task's output.

---

7) Post-processing with job dependencies

Question. Submit a combine job that runs **after all array tasks succeed**, binding its dependency to the array's JobID.

Show solution

```
cat > combine.R <<'RS'
fl <- list.files('results', pattern='^sim_\\d+\\.csv$', full.names=TRUE)
if (length(fl)==0) stop('No results found')
all <- do.call(rbind, lapply(fl, read.csv))
write.csv(all, 'results/combined.csv', row.names=FALSE)
cat('Wrote results/combined.csv with', nrow(all), 'rows\n')
RS

cat > combine.sbatch <<'SB'
#!/usr/bin/env bash
#SBATCH -J combine
#SBATCH -p short
#SBATCH -t 00:05:00
#SBATCH -c 1
#SBATCH --mem=2G
#SBATCH -o slurm-%j.out
module load R/4.3.2 2>/dev/null || true
Rscript combine.R
SB

# Submit array and capture JobID, then submit dependent combine
jid=$(sbatch --parsable sim_array.sbatch)
# If sim_array used --array, dependency should reference the parent array ID only
parent=${jid%%.*}
sbatch --dependency=afterok:$parent combine.sbatch
```

Notes: Use `afterok:` to run `combine` only if all array tasks complete successfully. Use `${jid%%.*}` to strip the task suffix.

---

8) Resource requests & usage diagnostics

Question. Request 2 CPUs and 4G RAM per task; later, inspect actual usage.

Show solution

```
# In your .sbatch header
#SBATCH -c 2
#SBATCH --mem=4G

# Inspect after completion
sacct -j 123456 -o JobID,JobName%30,AllocCPUS,Elapsed,MaxRSS,State,ExitCode
```

Tip: Prefer `--mem=` (per node) vs `--mem-per-cpu=` depending on your site policy.

---

9) Resubmit only failed array tasks

Question. Find which tasks failed from a previous array job and resubmit only those indices.

Show solution

```
jid=123456  # parent array job ID
# List failed indices (State not COMPLETED)
fail=$(sacct -j $jid --format=JobID,State -n | awk -F'[_. ]' '$2!="batch" && $3!="COMPLETED"

echo "Failed indices: $fail"
[ -n "$fail" ] && sbatch --array=$(echo $fail | tr ' ' ',') sim_array.sbatch
```

This parses sub-job entries like `123456_7` and extracts `7` when `State != COMPLETED`.

---

10) Interactive work (debugging)

Question. Start an interactive shell on a compute node and verify R sees multiple threads.

Show solution

```
# Allocate and attach to a compute node for 10 min with 2 CPUs and 2G RAM
salloc -p short -t 00:10:00 -c 2 --mem=2G
srun --pty bash

module load R/4.3.2 2>/dev/null || true
```

```
R -q <<'RS'
parallel::detectCores()
sessionInfo()
RS

# Exit when done
exit  # from R
exit  # from shell to release allocation
```

---

11) VS Code / Positron remote dev

Question. Configure VS Code (or Positron) to edit/submit jobs on the cluster via SSH.

Show solution

VS Code (Remote - SSH):

1. Install extensions: *Remote - SSH*, *R*, optionally *Python*, *Bash IDE*.
2. Create `~/.ssh/config` entry on your laptop:

   ```
   Host myhpc
     HostName login.cluster.edu
     User your_netid
     IdentityFile ~/.ssh/id_ed25519
   ```

3. In VS Code: *Remote Explorer* → *SSH Targets* → *myhpc* → *Connect*.
4. Open your home/project directory on the cluster.
5. Ensure R is available in PATH on the cluster; set VS Code R extension (if needed) to use `/usr/bin/R` or your module path.
6. Use VS Code terminal (connected to myhpc) to run `sbatch`, `squeue`, etc. Edit `.sbatch`/`.R` files locally but they execute on the cluster.

Positron:

- Install the *Remote - SSH* (or built-in remote) capability; connect similarly to open a remote workspace.
- Configure the R path in Positron settings to point to the cluster's R binary; use the integrated terminal for `sbatch`.

Optional: set up SSH keys and agent forwarding to enable Git from the cluster.

12) Helpful shell aliases/functions for Slurm

Question. Add helpers to your `~/.bashrc` or `~/.bash_profile` to speed up common tasks.

Show solution

```
cat >> ~/.bashrc <<'BRC'
# Slurm quick views
alias sj='squeue -u $USER -o "%A %j %t %M %D %R"'
alias sa='sacct -u $USER --starttime today -o JobID,JobName%30,State,Elapsed,MaxRSS,ExitCode

# Tail latest log(s)
sl(){ tail -n +1 -f slurm-*.out; }

# Submit and print JobID only
sb(){ sbatch --parsable "$@"; }

# Describe a job
sd(){ scontrol show job "$1" | less; }

# Resubmit failed array tasks for a parent JobID
sref(){ jid="$1"; idx=$(sacct -j "$jid" -n -o JobID,State | awk -F'[_. ]' '$2!="batch" && $3
BRC

# Reload shell config
source ~/.bashrc
```

These helpers give you one-letter shortcuts for listing jobs (`sj`), recent accounting (`sa`), tailing logs (`sl`), describing a job (`sd`), and resubmitting failures (`sref`).

13) Bonus: Make a project scaffold

Question. Create a scaffold with directories and template scripts for sims.

Show solution

```
mkdir -p {scripts,results,logs}

# Template sbatch header you can copy into scripts/
cat > scripts/_header.sbatch <<'H'
#!/usr/bin/env bash
#SBATCH -p short
#SBATCH -t 00:10:00
#SBATCH -c 1
#SBATCH --mem=2G
#SBATCH -o logs/slurm-%A_%a.out
H
```

Now copy `_header.sbatch` into new jobs and append your commands.

Deliverables (for practice)

- `r_minimal.sbatch` and output log
- `sim.R`, `params.csv`, `sim_array.sbatch`
- Evidence of a dependency submission (`combine.sbatch` and combined output)
- Your updated `~/.bashrc` helpers