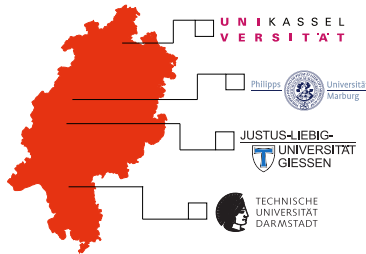


Introduction to the Slurm job manager

Hessisches Kompetenzzentrum für Hochleistungsrechnen (HKHLR)

HKHLR

February 27, 2025



HKHLR is funded by the Hessian Ministry of Sciences and Arts



In this talk:

- ▶ Why do we need batch job scheduling?
- ▶ Introduction to job scripts
- ▶ Introduction to important Slurm commands
- ▶ Job management strategies

Goals

- ▶ Understanding the basics of job scheduling
- ▶ Being able to appropriately submit your own jobs
- ▶ Getting familiar with some advanced controls for submitting multiple jobs

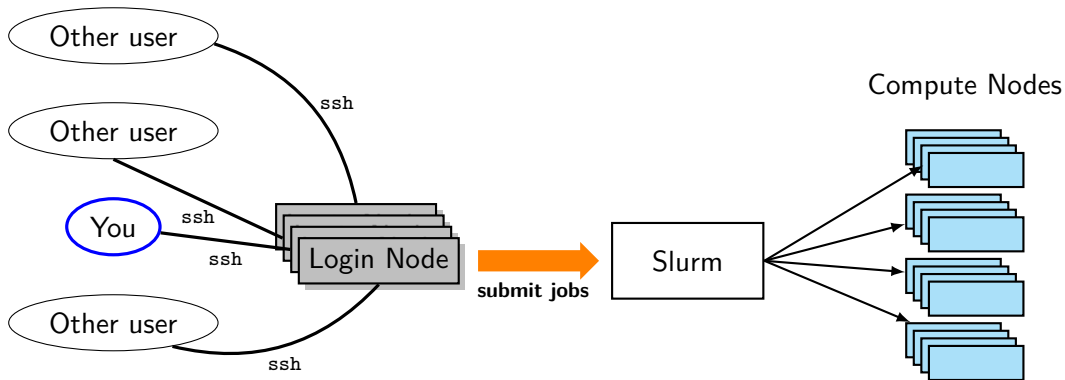
- ▶ For exercises and examples, we will solely focus on the Slurm scheduler
- ▶ The Slurm exercises should run on all clusters that are using Slurm (all Hessian university clusters; a large percentage of German and international clusters)

What does "efficiency" mean in an HPC context?

- ▶ The aim of an HPC cluster is to provide computational resources (i.e. CPUtime, RAM, etc.) on demand
- ▶ "On demand" means as little as possible downtime: **always on**
- ▶ "Always on" means steady power consumption and need for cooling, **even if the resources are idle**
- ▶ Therefore "efficiency" in an HPC context mean **maximizing resource utilization over cluster uptime**

- ▶ *Maximizing resource utilization manually would require perfect coordination between **all** users, at all times!*
- ▶ A job scheduler **automatically** aims for maximizing resource utilization
- ▶ Enforce fair sharing of resources between all eligible users
- ▶ Enforce resource shares granted to each user/group

- ▶ The scheduler keeps track of resources (CPU slots, RAM, etc.) that are:
 - ▶ available
 - ▶ currently in use
 - ▶ reserved by waiting jobs
- ▶ Each resource reservation (i.e. submitting a job) is assigned a **priority**
 - ▶ Details depend on local cluster configuration
 - ▶ Priority progressively gets higher while waiting
 - ▶ It might also get higher or lower according to past resource usage
- ▶ If priority is high enough and resources are available, the job starts
- ▶ Key statistics are recorded for future job prioritization and monitoring statistics



- ▶ Accepts resource requirements via job scripts and distributes jobs to the compute nodes
- ▶ Captures stdout and stderr in files
- ▶ Sends E-Mails when jobs start, end, or terminate with an error

scripts/basic_job_script.sh

```
1 #!/bin/bash
2
3 # SPDX-FileCopyrightText: 2021 Competence Center for High Performance Computing in He
4 # SPDX-License-Identifier: MIT
5
6 # The following lines are interpreted by Slurm
7 #SBATCH --ntasks 1
8 #SBATCH --mem-per-cpu=10
9 #SBATCH --time 00:00:05
10
11 #SBATCH --job-name Hello_World
12 #SBATCH --output job.out
13
14
15 # Job script starts here (interpreted by Shell, e.g. 'bash')
16
17 echo "Hello from $HOSTNAME"
```


- ▶ Note the "shebang" `#!/bin/bash`: This is a shell script!
- ▶ Slurm parameters are prefixed by `#SBATCH` (ignored by shell due to `#`)
 - ⇒ Slurm will configure the job settings based on the parameters.
 - ▶ All job properties are controlled by setting parameters
- ▶ Rest of job script is read by shell interpreter (must contain valid shell commands!)

Mandatory parameters

- ▶ Most clusters have parameters that *must always be set* in the job submission
- ▶ Knowing which parameters are mandatory and what defaults are set for non-mandatory parameters is crucial

Mandatory Parameters on Lichtenberg

- ▶ `-n` or `--ntasks`: Number of processes to start
- ▶ `--mem-per-cpu`: Maximum memory per core (in Mbytes)
- ▶ `-t` or `--time`: Time limit for this job (format: hh:mm:ss or dd-hh:mm:ss or seconds)

While mandatory parameters vary from cluster to cluster some are important in general:

- ▶ `--job-name`: Meaningful names help identify jobs while running
- ▶ `--time`: Generally a good idea to give a reasonable estimate for the wall-clock time consumed by the job.
- ▶ `--nodes` and `--ntasks-per-node`: How many compute nodes to use and how many process to start *per* node. Particularly important when nodes are granted exclusively on the cluster. Total number of processes (`--ntasks`): $\#tasks = \#nodes \times \#tasks-per-node$
- ▶ `--partition`: On which "part(s)" of the cluster to run. A "partition" typically specifies certain resource limits (max. run-time, max. number of compute nodes per job.)
- ▶ `--account`: To which account to assign consumed resources. Users can have multiple accounts within Slurm.

- ▶ `-o` or `--output`: File where stdout of the job will be redirected
- ▶ `-e` or `--error`: File where stderr of the job will be redirected
- ▶ `--mail-type`: One of NONE, BEGIN, END, FAIL, REQUEUE, ALL
- ▶ `--mail-user`: *Valid* mail address where notifications will be sent to.
- ▶ `-c` or `--cpus-per-task`: Allocate more CPUs per process (threads)
 - ▶ Needed for e.g. shared memory parallelism (OpenMP) together with environment variables `OMP_NUM_THREADS` or `MKL_NUM_THREADS`
- ▶ `-C` or `--constraint`: If you need a special type of nodes
 - ▶ e.g. one that has a graphic accelerator

Use long names for better readability

E.g., parameter `-c` is not the same as `-C`! Using long names avoids confusion.

For an exhaustive list of all job parameters see the manual pages of the `sbatch` command.

```
1 $ man sbatch # '$' is the shell prompt
```

The `sbatch` command is used to submit jobs to the queue:

```
1 $ sbatch <job script>
```

Advice

It is recommended to put *all* parameters needed for a particular job inside the job script. This will greatly enhance the reproducibility of your jobs.

- ▶ Choosing parameter values, especially for max. runtime and RAM, requires some educated guessing
 - ▶ For exclusive node allocation setting the RAM is not necessary.
- ▶ *Guiding principle*: As high as necessary, as low as possible.
- ▶ When in doubt, a more generous reservation ensures your job won't be killed prematurely.

Good practise

- ▶ Check resource requirements with some test jobs before submitting a "swarm" of jobs.
- ▶ Take scheduler defaults into account

The hardware available on an HPC system is important when configuring your job scripts. Be aware of the hardware configuration your current system:

- ▶ How many cores per node?
- ▶ How much available RAM per node?
- ▶ How much available RAM per core?
- ▶ If the cluster runs in *exclusive* mode (i.e. only one running job per node): Try to utilize all cores on that node
- ▶ Hardware configuration of (most) of the Hessian clusters can be found at <https://www.hkhlr.de/en/cluster>

A Slurm installation comes with several command line tools to handle jobs:

- ▶ `sbatch <job_script_file>`: Submit a job and return a jobID
 - ▶ Command line parameters will override parameters set in jobscript
- ▶ `squeue`: List all your jobs and their status
 - ▶ Estimating the start time for a pending job: `squeue --start -j <jodid>`
 - ▶ Formatted output (see `man squeue` for more parameter details):
`squeue -o "%.7i %.9P %.20j %.8u %.2t %.15M %.6D %.4C %25R %.10Q %.20S"`
- ▶ `scancel <jobid>`: Cancel a job
- ▶ `sinfo`: Get an overview of cluster nodes and partitions
- ▶ `srun` can be used to start an interactive session but this is **not recommended** on most clusters

- ▶ For a general summary, use the "send mail at job completion" scheduler option
 - ▶ `#SBATCH --mail-type=END`
 - ▶ You can specify a different user to receive the mail than the default (user who submitted the job) by also setting `#SBATCH --mail-user=<user>`
- ▶ If the 'seff' tool is installed, `seff <jobID>` also provides the same informations as the END mail
- ▶ For an extensive summary, the `sacct` command offers a large amount of options
- ▶ Simple `sacct` example: `sacct --job=<JobID> --format=JobID,CPUTime,Elapsed,MaxVMSize`
- ▶ Alternatively, you can use the `/usr/bin/time` command as a wrapper for your program call (see `man time` for details)
- ▶ Simple `/usr/bin/time` example: `/usr/bin/time --format "MaxMem: %Mkb, WCT: %E seconds"`

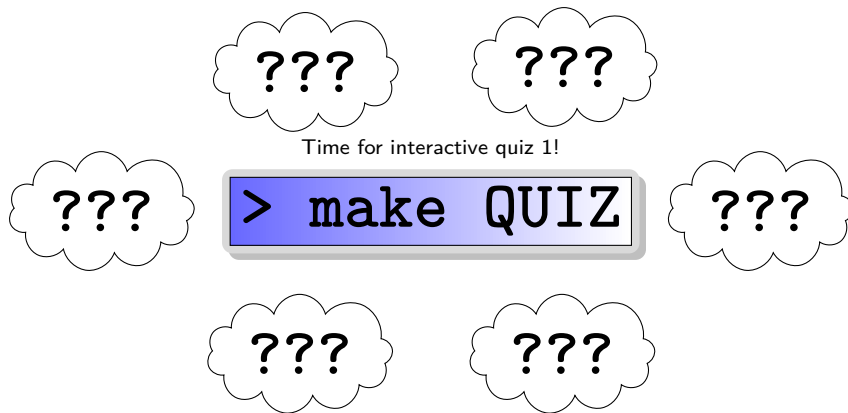
The module system is explained in the Linux/Shell course (ProTHPC Module 1), only a short recap here:

- ▶ On most clusters, each job starts with an empty environment
- ▶ Load modules *inside the job script* to defining the job's environment
- ▶ Using `module purge` as the first command

```
1 #!/bin/bash
2 # Slurm parameters
3 ...
4
5 # Prepare environment for current job
6 module purge # clean environment: no version conflicts or unwanted defaults
7 module add gcc/9.2.0
8 module add name/of/module
```

Writing and submitting a first basic job script:

- ▶ We want to use the `hostname` command to get info on which node our job has been executed
- ▶ Different clusters may have different *mandatory* parameters (e.g. Lichtenberg: `--ntasks`, `--mem-per-cpu`, and `--time`)
- ▶ Different clusters may have different parameter *limits* (e.g. max. runtime)



Are Job Schedulers an essential part of HPC systems?

A: No, they are just nice to have.

B: Yes

Are Job Schedulers an essential part of HPC systems?

A: No, they are just nice to have.

B: **Yes**

In a job script, what is the prefix for scheduler parameters?

- A: `#$` in Slurm
- B: No prefix needed at all.
- C: `#SBATCH` in Slurm

In a job script, what is the prefix for scheduler parameters?

- A: `#$` in Slurm
- B: No prefix needed at all.
- C: `#SBATCH` in Slurm

Which are the **mandatory** scheduler parameters on the Lichtenberg cluster?

- A: Max. runtime, memory per core, job name (`--time`, `--mem-per-cpu`, `--job-name`)
- B: Memory per core, number of tasks, output file (`--mem-per-cpu`, `--ntasks`, `--output`)
- C: Max. runtime, memory per core, number of tasks (`--time`, `--mem-per-cpu`, `--ntasks`)
- D: No parameters are mandatory

Which are the **mandatory** scheduler parameters on the Lichtenberg cluster?

- A: Max. runtime, memory per core, job name(--time, --mem-per-cpu, --job-name)
- B: Memory per core, number of tasks, output file (--mem-per-cpu, --ntasks, --output)
- C: **Max. runtime, memory per core, number of tasks** (--time, --mem-per-cpu, --ntasks)
- D: No parameters are mandatory

How many CPU cores are reserved by setting `--ntasks=4` and `--cpus-per-task=6`?

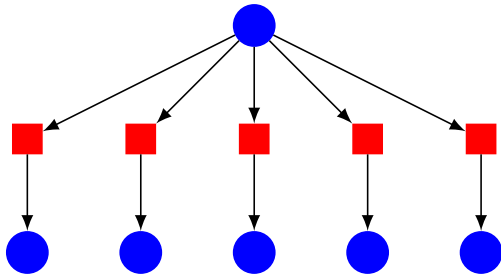
- A: 4
- B: 6
- C: 10
- D: 24

How many CPU cores are reserved by setting `--ntasks=4` and `--cpus-per-task=6`?

- A: 4
- B: 6
- C: 10
- D: **24**



Linear workflow with each job depending on its predecessor.



"Trivially" parallel workflow with independent jobs. Work is parallelized "by hand". Oftentimes the the most efficient way to parallelize a workflow.

Slurm supports a feature called *array jobs*. To make use of this we need the `--array` option inside our job script.

- ▶ Submitting multiple jobs at once, with a single job script and a single command
- ▶ Each array task will reserve the full amount of resources that are requested by the job script (e.g. `--ntasks` is the amount of processes that *every* array task can start)
- ▶ Each array task has a unique ID, which can be referenced in the job script via the `SLURM_ARRAY_TASK_ID` variable
- ▶ Task IDs can be set explicitly and as a range, with or without a stepsize (see examples)
- ▶ The number of maximum tasks that can execute simultaneously can also be set (see examples)

This is a basic job script that can be used to submit an array of jobs.

scripts/array.slurm

```
1 #!/bin/bash
2
3 # SPDX-FileCopyrightText: 2021 Competence Center for High Performance Computing in Hessen (HKHLR)
4 # SPDX-License-Identifier: MIT
5
6 #SBATCH --ntasks 1
7 #SBATCH --mem-per-cpu=10
8 #SBATCH --time 00:05:00
9
10 # #SBATCH --array=0-10
11 # #SBATCH --array=0-10%3 # %3 --> maximally 3 tasks from array running simultaneously
12 # #SBATCH --array=0-10:2 # :2 --> step size of 2 (same as 0,2,4,6,8,10)
13 #SBATCH --array=1,3,5,6-10:2%3
14
15 #SBATCH --output job%A_%a.out
16
17 sleep 30
18 echo "Hello from $SLURM_ARRAY_TASK_ID of $SLURM_ARRAY_TASK_COUNT"
19 echo "Lowest task index: $SLURM_ARRAY_TASK_MIN , highest task index: $SLURM_ARRAY_TASK_MAX"
```

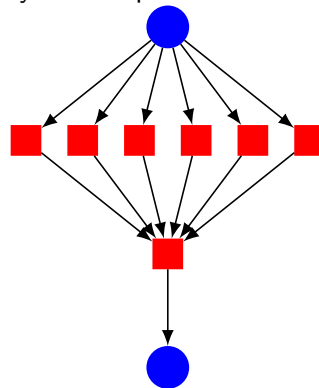
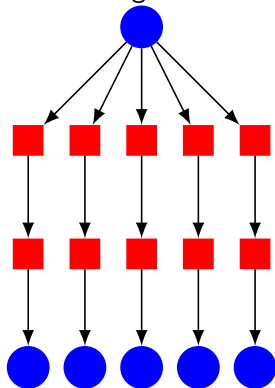
In some workflows jobs depend on each other. Slurm offers the `--dependency` option to make jobs relate to others.

- ▶ Set dependencies from other jobs: `<type>:<job_ID>`
 - ▶ `after`: Can begin after the specified jobs have **begun execution**
 - ▶ `afterany`: Can begin after the specified jobs have **terminated**
 - ▶ `afternotok`: Specified jobs have terminated in some **failed state**
 - ▶ `afterok`: Specified jobs have **successfully** executed
 - ▶ `aftercorr`: "After correlated" (later)
 - ▶ `singleton`: Can begin after **all other jobs with the same job name** have ended

```
1 $ sbatch job1.sh
2 successfully submitted batch job 12345
3 $ sbatch --dependency=afterok:12345 job2.sh
```

Remember that we can use the `--dependency` flag either from the command line or inside our job script.

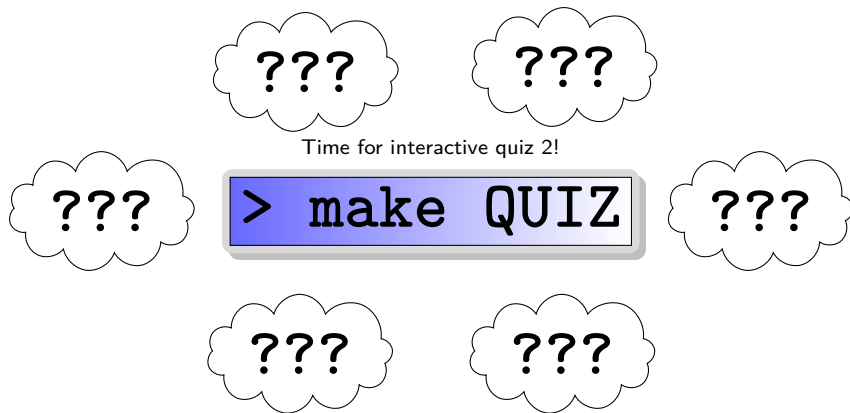
We can also imagine more complicated workflows with arrays and dependencies.



- ▶ A job can depend on multiple jobs, by using a colon-separated list:
`--dependency=afterok:<jobID>:<jobID>:<jobID>`
- ▶ A job depending on a task array with `afterok`, `afterany`, `afternotok` will wait until the *last* task has finished
- ▶ If two task arrays should depend on each other, `aftercorr:<jobID>` can be used; each dependent task will then be able to run *as soon as its predecessor with the same task ID has finished*
- ▶ Dependencies can also be set in a wrapper script that reads the job IDs from the `sbatch` command (see example)

scripts/dependency_wrapper.sh

```
1 #!/bin/bash
2
3 # SPDX-FileCopyrightText: 2021 Competence Center for High Performance Computing in Hessen (HKHLR)
4 # SPDX-License-Identifier: MIT
5
6 jid1=$(sbatch dependency_1.slurm | head -n 1 | awk '{FS=" ";print$4}')
7 echo "Submitted job ID $jid1"
8
9 # Using 'afterany' will make the job start when the last job from the
10 # job array it depends on has completed.
11 jid2=$(sbatch --dependency=afterany:$jid1 dependency_2.slurm | head -n 1 | awk '{FS=" ";print$4}')
12 echo "Submitted job ID $jid2"
```



What is the scheduler parameter for defining task arrays?

A: `-t` in Slurm

B: `--array` in Slurm

What is the scheduler parameter for defining task arrays?

A: `-t` in Slurm

B: `--array` in **Slurm**

You are submitting a job with `--ntasks=4`, `--mem-per-cpu=2G`, and `--array=1-4`. What will happen?

- A: Four jobs are spawned, each having 4 processes and 2GB RAM (500 MB per process).
- B: Four jobs are spawned, each having 1 process and 500 MB RAM (500 MB per process).
- C: Four jobs are spawned, each having 4 processes and 8 GB RAM (2 GB per process).

You are submitting a job with `--ntasks=4`, `--mem-per-cpu=2G`, and `--array=1-4`. What will happen?

- A: Four jobs are spawned, each having 4 processes and 2GB RAM (500 MB per process).
- B: Four jobs are spawned, each having 1 process and 500 MB RAM (500 MB per process).
- C: **Four jobs are spawned, each having 4 processes and 8 GB RAM (2 GB per process).**

Given the parameter input `--array=4,5,6-14:4%3`. How many tasks will be created, what is their task index? How many can run at the same time?

- A: 6 tasks with indices 4, 5, 6, 9, 12, 14 (a maximum of four will run concurrently)
- B: 5 tasks with indices 4, 5, 6, 10, 14 (a maximum of three will run concurrently)
- C: 9 tasks with indices 4, 5, 6, 8, 9, 10, 12, 13, 14 (a maximum of three will run concurrently)

Given the parameter input `--array=4,5,6-14:4%3`. How many tasks will be created, what is their task index? How many can run at the same time?

- A: 6 tasks with indices 4, 5, 6, 9, 12, 14 (a maximum of four will run concurrently)
- B: **5 tasks with indices 4, 5, 6, 10, 14 (a maximum of three will run concurrently)**
- C: 9 tasks with indices 4, 5, 6, 8, 9, 10, 12, 13, 14 (a maximum of three will run concurrently)

How to you set up a job that may if start another job has failed (e.g.: Restart a calculation from a checkpoint after the previous job exceeded its runtime limit.)?

A: It is not possible to set up such a dependency.

B: With `--dependency=afterok:<jobID>`

C: With `--dependency=afternotok:<jobID>`

How to you set up a job that may if start another job has failed (e.g.: Restart a calculation from a checkpoint after the previous job exceeded its runtime limit.)?

A: It is not possible to set up such a dependency.

B: With `--dependency=afterok:<jobID>`

C: **With** `--dependency=afternotok:<jobID>`

- ▶ Calculations on and HPC system are conducted by means of batch jobs
- ▶ Execution of the jobs is managed by a scheduler
- ▶ We have learnt about different types of Slurm jobs scripts and how to write them
- ▶ We have learnt about important Slurm command line tools
- ▶ Job arrays are useful to manage multiple jobs
- ▶ Job dependencies, possibly in combination with job arrays, can be used to model complex workflows

- ▶ During the noon break, try to complete the last exercise on the handout
- ▶ The basic idea is to create a job workflow of two consecutive array jobs and a final 'collector' job
- ▶ The previous demos, which are also included on the exercise sheet, should contain all necessary information solve the tasks
- ▶ Problems or questions that arise while working on the exercise (as well as any other questions you may have) can be discussed in the afternoon session