Batch Job Scheduling with SLURM

Hessisches Kompetenzzentrum für Hochleistungsrechnen (HKHLR)

Marcel Giar, René Sitt, Anja Gerbes, Nikolas Luke

16.03.2021

HKHLR is funded by the Hessian Ministry of Sciences and Arts

Content

In this Talk:

- Why do we need batch job scheduling?
- Introduction into jobscripts
- Introduction into 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

Content

Exercises

- For exercises and examples, we will solely focus on the SLURM scheduler
- The SLURM exercises should run on all clusters that are using SLURM (Darmstadt, Frankfurt, Gießen, Kassel)
- Since Marburg's MaRC2 cluster uses SGE instead of SLURM, we will occasionally show some SGE commands that are mostly equivalent to their SLURM counterparts
- Marburg's new MaRC3 cluster will also run Slurm

SGE

- Informations about SGE specifics will be given in green text blocks

Why do we need a Job Scheduler?

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 few 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**

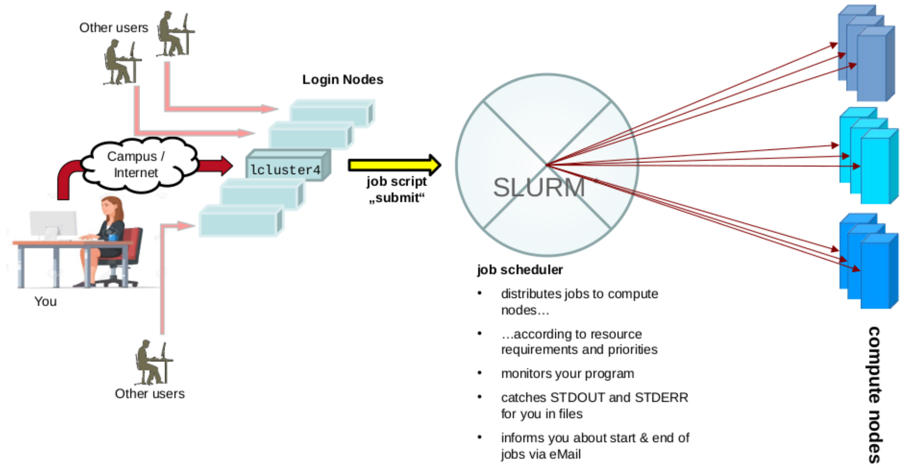
Goals of scheduler usage

- *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

What does a scheduler do?

- 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
  - 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

What does a scheduler do?



job scheduler

- distributes jobs to compute nodes…
- …according to resource requirements and priorities
- monitors your program
- catches STDOUT and STDERR for you in files
- informs you about start & end of jobs via eMail

* **S**imple **L**inux **U**tility for **R**esource **M**anagement

Job script skeleton

```bash
#!/bin/bash

# The following lines are interpreted by Slurm
#SBATCH --ntasks 1
#SBATCH --mem-per-cpu=10
#SBATCH --time 00:00:05

#SBATCH --job-name Hello_World
#SBATCH --output job.out


# Job script starts here (interpreted by Shell, e.g. `bash`)

echo "Hello from $HOSTNAME"
```

Parameters of a Job

- 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!)

SGE parameters

- SGE parameters are prefixed by `'#$'` instead

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 parameter examples

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)

Giessen

Sample job scripts are provided (can only be reached from intra-net):
`https://justhpc.hpc.uni-giessen.de/trac/wiki/SlurmExamples`

- Contain parameters needed to run on different partitions
- Cover "typical" types of calculcations

Important parameters

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

- `--job-name`: Meaning names help identify jobs while running.
- `--time`: Generally 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 x #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 ressources. Users can have multiple accounts within Slurm.

Other Parameters

- `-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.

Parameters of a job

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

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

Spoiler alert

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

```
$ 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. Do *not* use `sbatch` with command line arguments to submit jobs.

Parameters of a job (SGE)

SGE Parameters

- For most SLURM parameters, there is a corresponding SGE expression
- Most resources (e.g. runtime, memory) are reserved with `-l <keyword>=<value>` instead of a dedicated parameter
- Use `-cwd` for being able to use paths relative to the current working directory
- Amount of processes and their distribution is set via *parallel environments* rather than directly (check `qconf -spl` or the cluster reference sheet for a list)

How to choose parameter values

- 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 sumitting a "swarm" of jobs.
- Take scheduler defaults into account.

How to choose parameter values

Be aware of the hardware configuration

- 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`

Slurm Commands

- `sbatch <job_script_file>`: Submit a job and return jobID
  - Command line parameter 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

Equivalent SGE commands

- `sbatch <job_script_file>` ↔ `qsub <job_script_file>`
- `squeue` ↔ `qstat`
- `scancel <jobid>` ↔ `qdel <jobid>`
- `sinfo`: no direct equivalent. Queue list: `qstat -g c`, node list: `qhost`

How to check resource usage of past jobs?

- 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>`
- 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"
```

Module system

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

```
#!/bin/bash
# Slurm parameters
...

# Prepare environment for current job
module purge # clean environment: no version conflicts or unwanted defaults
module add gcc/9.2.0
module add name/of/module
```

Module environment

Initializing the module environment

- Some clusters need an additional script line to access the module environment from compute nodes (check the Quick Reference Sheet!)
- Usually: `source /etc/profile.d/modules.sh`
    - You can put this line inside your `$HOME/.bashrc` file

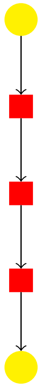
Demo

- 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)
  - All the Hessian university clusters have a quick reference sheet where these specific restrictions and limits are listed; these can be downloaded from `https://www.hkhlr.de/en/material`

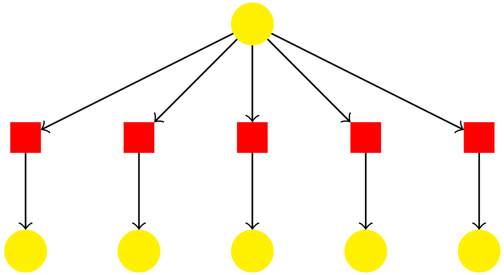
First Short Break & Quiz

- 10 Minutes break to grab a coffee, stretch a bit, etc.
- Try to complete the very short self test on 'Job Scheduling Basics' (Zoom poll) - ask in the chat if something seems unclear!

Workflow with multiple jobs



Linear workflow with each job depending on its predecessor.



"Trivially" parallel workflow with independent jobs. Work is parallelised "by hand". The oftentimes is the most efficient way to parallelise a workflow.

Job Arrays

`--array` option:

- 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)

Job Arrays: Example script

```
#!/bin/bash

#SBATCH --ntasks 1
#SBATCH --mem-per-cpu=10
#SBATCH --time 00:05:00

# #SBATCH --array=0-10
# #SBATCH --array=0-10%3 # %3 --> maximally 3 tasks from array running simulaneously
# #SBATCH --array=0-10:2 # :2 --> step size of 2 (same as 0,2,4,6,8,10)
#SBATCH --array=1,3,5,6-10:2%3

#SBATCH --output job%A_%a.out

sleep 30
echo "Hello from $SLURM_ARRAY_TASK_ID of $SLURM_ARRAY_TASK_COUNT"
echo "Lowest task index: $SLURM_ARRAY_TASK_MIN , highest task index: $SLURM_ARRAY_TASK
  _MAX"
```

SGE Array Jobs

- On SGE, the array command is `-t <start>-<end>:<stepsize>`
- The max amount of concurrent tasks is set by an extra parameter `-tc <amount_of_concurrent_tasks>`
- Usable variables in task arrays are named `$SGE_TASK_ID`, `$SGE_TASK_FIRST`, `$SGE_TASK_LAST`, and `$SGE_TASK_STEPSIZE`
- SGE task IDs *must be positive integers*, 0 is *not allowed*
- Tasks that use parallel environments should be started in binary mode (`chmod u+x <jobscript>`, then `qsub -b y <SGE parameters> <jobscript>` due to a bug

Job dependencies

`--dependency` option:

- 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

Example:

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

Job Dependencies with SGE

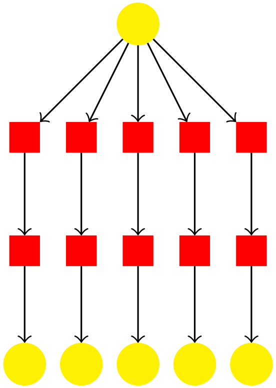
SGE Dependencies

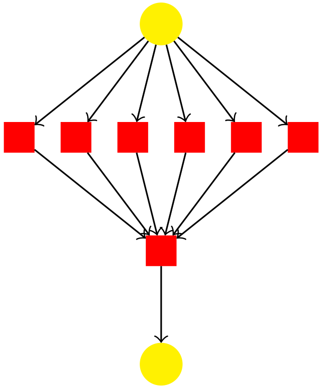
- On SGE, dependencies are set with `-hold_jid <job_ID>`
- Instead of a job ID, you can also use the job name or even patterns with placeholders like `*` or `?`
- Dependency jobs are started when all referenced jobs have completed (successfully or not!), *except* if any finished with exit code 100 (i.e. a "queue error")

Second Short Break & Quiz

- Another 10 minutes break before the last part of the talk
- Try to complete the very short self test on 'Task Arrays and Dependencies' (Zoom poll) - ask in the chat if something seems unclear!

Complex Workflows





Complex 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 precedessor 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)

Dependencies wrapper: Example script

```
#!/bin/bash

jid1=`sbatch dependency_1.slurm | head -n 1 | awk '{FS=" ";print$4}'`
echo "Submitted job ID $jid1"
jid2=`sbatch --dependency=afterany:$jid1 dependency_2.slurm | head -n 1 | awk '{FS="
  ";print$4}'`
echo "Submitted job ID $jid2"
```

SGE array dependencies

- On SGE, task-on-task dependencies can be set with `-hold_jid_ad <job_ID>`
- A wrapper script works similar (but job ID has to be read from index 3 instead of 4 due to different output)
- Alternatively you can also use the job name as an identifier

Summary

- Calculation is done with batch jobs
- Execution of the Jobs is managed by a scheduler

- Introduction into slurm jobscripts
- Introduction into important slurm commands
- Job arrays are useful to manage multiple jobs
- Job dependencies, possibly in combination with job arrays, can be used to model complex workflows

Noon Break Exercise

- 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 elements to put it together
- Problems or questions that pop up during this exercise, as well as any other questions you may have, can be discussed in the afternoon session