The Batch System (SLURM)

Objectives

- Get information about what a batch system is and which one is used at HPC2N.
- Learn basic commands for the batch system used at HPC2N.
- How to create a basic batch script.
- Managing your job: submitting, status, cancelling, checking...
- Learn how to allocate specific parts of Kebnekaise: skylake, zen3/zen4, GPUs...

- Large/long/parallel jobs **must** be run through the batch system.
- Kebnekaise is running Slurm.
- Slurm is an Open Source job scheduler, which provides three key functions.
  - Keeps track of available system resources.
  - Enforces local system resource usage and job scheduling policies.
  - Manages a job queue, distributing work across resources according to policies.
- In order to run a batch job, you need to create and submit a SLURM submit file (also called a batch submit file, a batch script, or a job script).

Note

Guides and documentation for the batch system at HPC2N here at: HPC2N's batch system documentation.

Basic commands

Using a job script is often recommended.

- If you ask for the resources on the command line, you will wait for the program to run before you can use the window again (unless you can send it to the background with &).
- If you use a job script you have an easy record of the commands you used, to reuse or edit for later use.

Note

When you submit a job, the system will return the Job ID. You can also get it with `squeue --me`. See below.

In the following, JOBSCRIPT is the name you have given your job script and JOBID is the job ID for your job, assigned by Slurm. USERNAME is your username.

- **Submit job:** `sbatch JOBSCRIPT`
- **Get list of your jobs:** `squeue -u USERNAME` or `squeue --me`
- **Give the Slurm commands on the command line:** `srun commands-for-your-job/program`
- **Check on a specific job:** `scontrol show job JOBID`
- **Delete a specific job:** `scancel JOBID`
- **Delete all your own jobs:** `scancel -u USERNAME`
- **Request an interactive allocation:** `salloc -A PROJECT-ID .......`
  - Note that you will still be on the login node when the prompt returns and you MUST preface with `srun` to run on the allocated resources.
  - I.e. `srun MYPROGRAM`
- **Get more detailed info about jobs:**
  `sacct -l -j JOBID -o jobname,NTasks,nodelist,MaxRSS,MaxVMSize`
  - More flags etc. can be found with `man sacct`
  - The output will be **very** wide. To view in a friendlier format, use
    `sacct -l -j JOBID -o jobname,NTasks,nodelist,MaxRSS,MaxVMSize | less -S`
    - this makes it sideways scrollable, using the left/right arrow key
- Web url with graphical info about a job: `job-usage JOBID`
- More information: `man sbatch`, `man srun`, `man ....`

Example

Submit job with `sbatch`

```
b-an01 [~]$ sbatch simple.sh
Submitted batch job 27774852
```

Check status with `squeue --me`

```
b-an01 [~]$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          27774852  cpu_zen4 simple.s bbrydsoe  R       0:00      1 b-cn1701
```

Submit several jobs (here several instances of the same), check on the status

```
b-an01 [~]$ sbatch simple.sh
Submitted batch job 27774872
b-an01 [~]$ sbatch simple.sh
Submitted batch job 27774873
b-an01 [~]$ sbatch simple.sh
Submitted batch job 27774874
b-an01 [~]$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          27774873  cpu_zen4 simple.s bbrydsoe  R       0:02      1 b-cn1702
          27774874  cpu_zen4 simple.s bbrydsoe  R       0:02      1 b-cn1702
          27774872  cpu_zen4 simple.s bbrydsoe  CG      0:04      1 b-cn1702
```

The status “R” means it is running. “CG” means completing. When a job is pending it has the state “PD”.

In these examples the jobs all ended up on nodes in the partition cpu_zen4. We will soon talk more about different types of nodes.

Job scripts and output

The official name for batch scripts in Slurm is Job Submission Files, but here we will use both names interchangeably. If you search the internet, you will find several other names used, including Slurm submit file, batch submit file, batch script, job script.

A job submission file can contain any of the commands that you would otherwise issue yourself from the command line. It is, for example, possible to both compile and run a program and also to set any necessary environment values (though remember that Slurm exports the environment variables in your shell per default, so you can also just set them all there before submitting the job).

Note

The results from compiling or running your programs can generally be seen after the job has completed, though as Slurm will write to the output file during the run, some results will be available quicker.

Outputs and any errors will per default be placed in the directory you are running from, though this can be changed.

Note

This directory should preferrably be placed under your project storage, since your home directory only has 25 GB of space.

The output file from the job run will default be named `slurm-JOBID.out`. It will contain both output as well as any errors. You can look at the content with `vi`, `nano`, `emacs`, `cat`, `less`...

The exception is if your program creates its own output files, or if you name the output file(s) differently within your jobscript.

Note

You can use Slurm commands within your job script to split the error and output in separate files, and name them as you want. It is highly recommended to include the environment variable `%J` (the job ID) in the name, as that is an easy way to get a new name for each time you run the script and thus avoiding the previous output being overwritten.

Example, using the environment variable `%J`:

- Error file: `#SBATCH --error=job.%J.err`
- Output file: `#SBATCH --output=job.%J.out`

Job scripts

A job submission file can either be very simple, with most of the job attributes specified on the command line, or it may consist of several Slurm directives, comments and executable statements. A Slurm directive provides a way of specifying job attributes in addition to the command line options.

Naming: You can name your script anything, including the suffix. It does not matter. Just name it something that makes sense to you and helps you remember what the script is for. The standard is to name it with a suffix of `.sbatch` or `.sh`.

Simple, serial job script

```
#!/bin/bash
# The name of the account you are running in, mandatory.
#SBATCH -A hpc2nXXXX-YYY
# Request resources - here for a serial job
# tasks per core is 1 as default (can be changed with ``-c``)
#SBATCH -n 1
# Request runtime for the job (HHH:MM:SS) where 168 hours is the maximum. Here asking for 15 m
#SBATCH --time=00:15:00

# Clear the environment from any previously loaded modules
module purge > /dev/null 2>&1

# Load the module environment suitable for the job - here foss/2022b
module load foss/2022b

# And finally run the serial jobs
./my_serial_program
```

Note

- You have to always include `#!/bin/bash` at the beginning of the script, since bash is the only supported shell. Some things may work under other shells, but not everything.
- All Slurm directives start with `#SBATCH`.
- One (or more) `#` in front of a text line means it is a comment, with the exception of the string `#SBATCH`. In order to comment out the Slurm directives, you need to put one more `#` in front of the `#SBATCH`.
- It is important to use capital letters for `#SBATCH`. Otherwise the line will be considered a comment, and ignored.

Let us go through the most commonly used arguments:

- **-A PROJ-ID**: The project that should be accounted. It is a simple conversion from the SUPR project id. You can also find your project account with the command `projinfo`. The **PROJ-ID** argument is of the form
  - hpc2nXXXX-YYY (HPC2N local project)
- **-N**: number of nodes. If this is not given, enough will be allocated to fullfill the requirements of -n and/or -c. A range can be given. If you ask for, say, 1-1, then you will get 1 and only 1 node, no matter what you ask for otherwise. It will also assure that all the processors will be allocated on the same node.
- **-n**: number of tasks.
- **-c**: cores per task. Request that a specific number of cores be allocated to each task. This can be useful if the job is multi-threaded and requires more than one core per task for optimal performance. The default is one core per task.

Simple MPI program

```
#!/bin/bash
# The name of the account you are running in, mandatory.
#SBATCH -A hpc2nXXXX-YYY
# Request resources - here for eight MPI tasks
#SBATCH -n 8
# Request runtime for the job (HHH:MM:SS) where 168 hours is the maximum. Here asking for 15 m
#SBATCH --time=00:15:00

# Clear the environment from any previously loaded modules
module purge > /dev/null 2>&1

# Load the module environment suitable for the job - here foss/2022b
module load foss/2022b

# And finally run the job - use srun for MPI jobs, but not for serial jobs
srun ./my_mpi_program
```

Prepare the exercise environment

Note

If you have not already done so, clone the material from the website https://github.com/hpc2n/intro-course:

1. Change to the storage area you created under `/proj/nobackup/kebnekaise-intro/`.
2. Clone the material:

```
git clone https://github.com/hpc2n/intro-course.git
```

3. Change to the subdirectory with the exercises:

```
cd intro-course/exercises/simple
```

You will now find several small programs and batch scripts which are used in this section and the next, "Simple examples".

In this section, we are just going to try submitting a few jobs, checking their status, cancelling a job, and looking at the output.

Preparations

1. Load the module `foss/2022b` (`ml foss/2022b`) on the regular login node. This module is available on all nodes.
2. Compile the following programs: `hello.c`, `mpi_hello.c`, `mpi_greeting.c`, and `mpi_hi.c`

```
gcc -o hello hello.c
mpicc -o mpi_hello mpi_hello.c
mpicc -o mpi_greeting mpi_greeting.c
mpicc -o mpi_hi mpi_hi.c
```

3. If you compiled and named the executables as above, you should be able to submit the following batch scripts directly: `simple.sh`, `mpi_greeting.sh`, `mpi_hello.sh`, `mpi_hi.sh`, `multiple-parallel-sequential.sh`, `multiple-parallel.sh`, or `multiple-parallel-simultaneous.sh`.

Exercises

Exercise: sbatch and squeue

Submit (`sbatch`) one of the batch scripts listed in 3. under preparations.

Check with `squeue --me` if it is running, pending, or completing.

Exercise: sbatch and scontrol show job

Submit a few instances of `multiple-parallel.sh` and `multiple-parallel-sequential.sh` (so they do not finish running before you have time to check on them).

Do `scontrol show job JOBID` on one or more of the job IDs. You should be able to see node assigned (unless the job has not yet had one allocated), expected runtime, etc. If the job is running, you can see how long it has run. You will also get paths to submit directory etc.

Exercise: sbatch and scancel

Submit a few instances of `multiple-parallel.sh` and `multiple-parallel-sequential.sh` (so they do not finish running before you have time to check on them).

Do `squeue --me` and see the jobs listed. Pick one and do `scancel JOBID` on it. Do `squeue --me` again to see it is no longer there.

Exercise: check output, change output files

1. Use `nano` to open one of the output files `slurm-JOBID.out` and looks at the content.
2. Try adding `#SBATCH --error=job.%J.err` and `#SBATCH --output=job.%J.out` to one of the batch scripts (you can edit it with `nano`). Submit the batch script again. See that the expected files get created.

Using the different parts of Kebnekaise

As mentioned under the introduction, Kebnekaise is a very heterogeneous system, comprised of several different types of CPUs and GPUs. The batch system reflects these several different types of resources.

At the top we have partitions, which are similar to queues. Each partition is made up of a specific set of nodes. At HPC2N we have three classes of partitions, one for CPU-only nodes, one for GPU nodes and one for large memory nodes. Each node type also has a set of features that can be used to select (constrain) which node(s) the job should run on.

Note

The three types of nodes also have corresponding resources one must apply for in SUPR to be able to use them.

While Kebnekaise has multiple partitions, one for each major type of resource, there is only a single partition, `batch`, that users can submit jobs to. The system then figures out which partition(s) the job should be sent to, based on the requested features (constraints).

Node overview

The “Type” can be used if you need a specific type of node. More about that later.

CPU-only nodes

CPU	Memory/core	number nodes	Type
2 x 14 core Intel skylake	6785 MB	52	skylake (intel_cpu)
2 x 64 core AMD zen3	8020 MB	1	zen3 (amd_cpu)
2 x 128 core AMD zen4	2516 MB	8	zen4 (amd_cpu)

GPU enabled nodes

CPU	Memory/core	GPU card	number nodes	Type
2 x 14 core Intel skylake	6785 MB	2 x Nvidia V100	10	v100
2 x 24 core AMD zen3	10600 MB	2 x Nvidia A100	2	a100
2 x 24 core AMD zen3	10600 MB	2 x AMD MI100	1	mi100
2 x 24 core AMD zen4	6630 MB	2 x Nvidia A6000	1	a6000
2 x 24 core AMD zen4	6630 MB	2 x Nvidia L40s	10	l40s
2 x 48 core AMD zen4	6630 MB	4 x Nvidia H100 SXM5	2	h100
2 x 32 core AMD zen4	11968 MB	6 x Nvidia L40s	2	l40s
2 x 32 core AMD zen4	11968 MB	8 x Nvidia A40	2	a40

Large memory nodes

CPU	Memory/core	number nodes	Type
4 x 18 core Intel broadwell	41666 MB	8	largemem

Requesting features

To make it possible to target nodes in more detail there are a couple of features defined on each group of nodes. To select a feature one can use the `-C` option to `sbatch` or `salloc`. This sets *constraints* on the job.

There are several reasons why one might want to do that, including for benchmarks, to be able to replicate results (in some cases), because specific modules are only available for certain architectures, etc.

To constrain a job to a certain feature, use

```
#SBATCH -C Type
```

Note

Features can be combined using “and” (`&`) or “or” (`|`). They should be wrapped in `'`’s.

Example:

```
#SBATCH -C 'zen3|zen4'
```

List of constraints:

For selecting type of CPU

Type is:

- intel_cpu
- broadwell
- skylake
- amd_cpu
- zen3
- zen4

For selecting type of GPU

Type is:

- v100
- a40
- a6000
- a100
- l40s
- h100
- mi100

For GPUs, the above GPU list of constraints can be used either as a specifier to `--gpus=type:number` or as a constraint together with an unspecified gpu request `--gpus=number` or `gpus-per-node=number`.

Note

For some MPI jobs, `mpirun` may fail on some GPU nodes if you specify GPUs with `--gpus=type:number` instead of using `--gpus-per-node=number` and a constraint for type of GPU.

The problem should not appear if you use `srun` instead of `mpirun`

For selecting GPUs with certain features

Type is:

- nvidia_gpu (Any Nvidia GPU)
- amd_gpu (Any AMD GPU)
- GPU_SP (GPU with single precision capability)
- GPU_DP (GPU with double precision capability)
- GPU_AI (GPU with AI features, like half precisions and lower)
- GPU_ML (GPU with ML features, like half precisions and lower)

For selecting large memory nodes

Type is:

- largemem

Examples, constraints

Only nodes with Zen4

```
#SBATCH -C zen4
```

Nodes with a combination of features: a Zen4 CPU and a GPU with AI features

```
#SBATCH -C 'zen4&GPU_AI'
```

Nodes with either a Zen3 CPU or a Zen4 CPU

```
#SBATCH -C 'zen3|zen4'
```

Examples, requesting GPUs

To use GPU resources one has to explicitly ask for one or more GPUs. Requests for GPUs can be done either in total for the job or per node of the job.

Ask for one GPU of any kind

```
#SBATCH --gpus=1
```

Another way to ask for one GPU of any kind

```
#SBATCH --gpus-per-node=1
```

Asking for a specific type of GPU

As mentioned before, for GPUs, constraints can be used either as a specifier to

`--gpus=type:number`

or as a constraint together with an unspecified gpu request

`--gpus=number`

or

`--gpus-per-node=number`

If doing one of the latter two, you need to add the constraint

`#SBATCH -C type`

In the batch job you would write something like this:

```
#SBATCH --gpus=number
#SBATCH -C type
```

where type is, as mentioned:

- v100
- a40
- a6000
- a100
- l40s
- h100
- mi100

Simple GPU Job - V100

```
#!/bin/bash
#SBATCH -A hpc2nXXXX-YYY
# Expected time for job to complete
#SBATCH --time=00:10:00
# Number of GPU cards needed. Here asking for 2 V100 cards
#SBATCH --gpus=2
#SBATCH -C v100

# Clear the environment from any previously loaded modules
module purge > /dev/null 2>&1
# Load modules needed for your program - here fosscuda/2020b
ml fosscuda/2020b

./my-gpu-program
```

Important

- The course project has the following project ID: hpc2n2025-014
- In order to use it in a batch job, add this to the batch script: `#SBATCH -A hpc2n2025-014`
- We have a storage project linked to the compute project: **kebnekaise-intro**.
  - You find it in `/proj/nobackup/kebnekaise-intro`.
  - Remember to create your own directory under it.

Keypoints

- To submit a job, you first need to create a batch submit script, which you then submit with `sbatch SUBMIT-SCRIPT`.
- You can get a list of your running and pending jobs with `squeue --me`.
- Kebnekaise has many different nodes, both CPU and GPU. It is possible to constrain the the job to run only on specific types of nodes.
- If your job is an MPI job, you need to use `srun` (or `mpirun`) in front of your executable in the batch script (unless you use software which handles the parallelization itself).