BATCH USAGE ON JSC
Introduction to Slurm

May 2024 | Chrysovalantis Paschoulas <c.paschoulas@fz-juelich.de> | HPCCDSS @ JSC

Member of Helmholtz Association

JÜLICH
Forschungszentrum

Batch System Concepts

- **Resource Manager** is the software responsible for managing the resources of a cluster, usually controlled by a scheduler.
  - It manages resources like tasks, nodes, CPUs, memory, network, etc.
  - It handles the execution of the jobs on the compute nodes.
  - It makes sure that jobs are not overlapping on the resources and does the cleaning.
- **Scheduler** is the software that controls user's jobs on a cluster according to policies. It receives and handles jobs from the users and controls the resource manager. It offers many features like:
  - Partitions, queues and QoS to control jobs according to policies/limits.
  - Scheduling mechanisms (backfill, fifo, etc).
  - Interfaces for defining workflows (jobscripts) or job dependencies and commands for managing the jobs (submit, cancel, etc).
- **Batch-System/Workload-Manager** is the combination of a scheduler and a resource manager. It combines all the features of these two parts in an efficient way.

JSC Batch Model

- Job **scheduling according to priorities**. Jobs with the highest priorities will be scheduled next.
- **Backfilling scheduling algorithm**. The scheduler checks the queue and may schedule jobs with lower priorities that can fit in the gap created by freeing resources for the next highest priority jobs.
- **No node-sharing**. The smallest allocation for jobs is one compute node. Running jobs do not disturb each other.
- For each **project** we create a **Unix-group**. Each real **person** has a unique **Unix-user account** that can be member of multiple projects/Unix-groups. A **project** can have one or more **CPU-Quotas** (contingent budgets which map to **Slurm Accounts** in SlurmDB).
- CPU-Quota modes: **monthly** and **fixed**. The projects are charged on a monthly base or get a fixed amount until it is completely used.
- Accounted CPU-Quotas/job = *Number-of-nodes x Walltime x cores/node (corehours)*
- Contingent/CPU-Quota states for the projects (for monthly mode): **normal**, **low-contingent**, **no-contingent**.
- Contingent priorities: **normal** > **lowcont** > **nocont**. Users without contingent get some penalties for the their jobs, but they are still allowed to submit and run jobs.

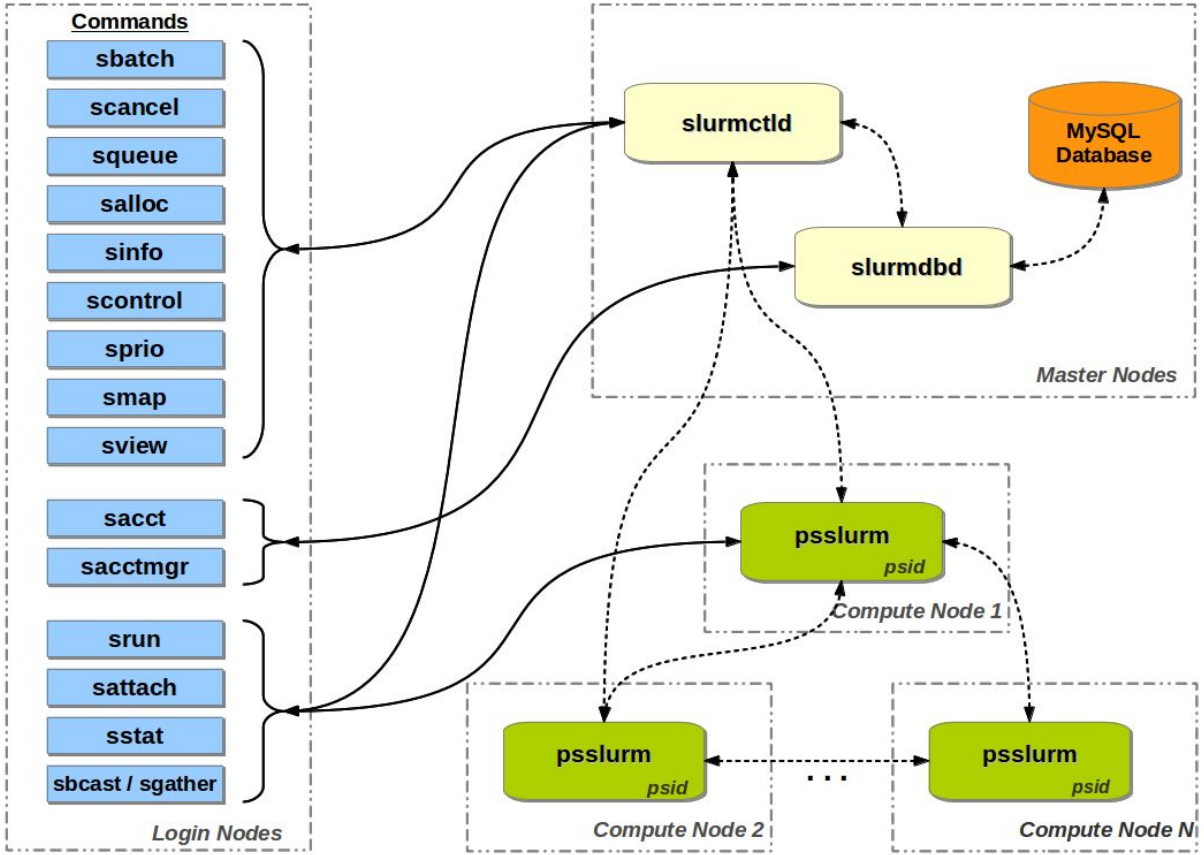
Slurm - Introduction (1)

- **Slurm** is the chosen Batch System (Workload Manager) that is used on our clusters. Slurm is an open-source project developed by SchedMD. For our clusters `psslurm`, which is a plug-in of `psid` daemon and part of the Parastation Cluster tools, is replacing slurmd on the compute nodes. `psslurm` is under development by ParTec and JSC in the context of our collaboration.
- Slurm's configuration on our clusters:
  - High-availability for the main daemons slurmctld and slurmdbd.
  - Backfilling scheduling algorithm.
  - No node-sharing.
  - Job scheduling according to priorities.
  - Accounting mechanism: slurmdbd with MySQL/MariaDB database.
  - User and job limits configured by QoS and Partitions.
  - No preemption configured. Running jobs cannot be preempted.
  - Prologue and Epilogue, with pshealthcheck from Parastation.
  - Generic resources (GRES) for different types of resources on the nodes.

Slurm - Introduction (2)

- Slurm groups the compute nodes into **Partitions**. Some limits and policies can be configured for each Partition:
  - allowed users, groups or accounts
  - max. nodes and max. wall-time limit per job
- Other limits are enforced also by the Quality-of-Services (**QoS**), according to the contingent of user's group, e.g. max. wall-time limit, max number or queued or running jobs per user, etc...
- Default limits/settings are used when not given by the users, like: number of nodes, number of tasks per node, wall-time limit, etc.
- According to project's contingent status jobs are given certain QoS:
  - **normal**: group has contingent, high job priorities.
  - **lowcont**: this months contingent was used.
    - penalty: *lower* job priorities, max. wall-time limit and max. running jobs
  - **nocont**: all contingent of the 3 months time-frame was used.
    - penalty: *lowest* job priorities, max. wall-time limit and max. running jobs
  - **suspended**: the group's project has ended; user cannot submit jobs

Slurm - Architecture



Slurm - TRES and GRES

- Slurm supports trackable resources (**TRES**) for accounting purposes. There are default TRES like: **nodes**, **cpus**, **memory**, **billing**, etc. But we can extend the resources that are tracked by defining extra generic resources (**GRES**) and configuring Slurm to track them too. TRES info is stored in Slurm database and can be queried with sacct command.

- On our clusters we have set 2 types of GRES:
  - **Memory**: `mem96, mem128, mem256, mem512, mem1024, ...`
  - **GPUs**: `gpu:[0-N]`

- GRES are used for:
  - specifying desired resources during submission
  - accounting and statistics

Slurm - User Commands (1)

- `salloc` is used to request interactive jobs/allocations.
- `sattach` is used to attach standard input, output, and error plus signal capabilities to a currently running job or job step.
- `sbatch` is used to submit a batch script (which can be a bash, Perl or Python script).
- `scancel` is used to cancel a pending or running job or job step.
- `sbcast` is used to transfer a file to all nodes allocated for a job.
- `sgather` is used to transfer a file from all allocated nodes to the currently active job. This command can be used only inside a job script.
- `scontrol` provides also some functionality for the users to manage jobs or query and get some information about the system configuration.
- `sinfo` is used to retrieve information about the partitions, reservations and node states.

Slurm - User Commands (2)

- `sprio` can be used to query job priorities.
- `squeue` allows to query the list of pending and running jobs.
- `srun` is used to initiate job-steps mainly within a job or start an interactive jobs. A job can contain multiple job steps executing sequentially or in parallel on independent or shared nodes within the job's node allocation.
- `sshare` is used to retrieve fair-share information for each user.
- `sstat` allows to query status information about a running job.
- `sview` is a graphical user interface to get state information for jobs, partitions, and nodes.
- `sacct` is used to retrieve accounting information about jobs and job steps in Slurm's database.
- `sacctmgr` allows also the users to query some information about their accounts and other accounting information in Slurm's database.

** For more detailed info please check the online documentation and the man pages.*

Slurm - Job Submission

- There are 2 commands for job allocation: sbatch is used for batch jobs and salloc is used to allocate resource for interactive jobs. The format of these commands:
  - `sbatch [options] <jobscript>`
  - `salloc [options] [<command> [args...]]`
- List of the most important submission/allocation options:

```
-A|--account          Charge CPU-Quota to specified account (budget ID)
-c|--cpus-per-task    Number of logical CPUs (hardware threads) per task
-e|--error            Path to the job's standard error
-i|--input            Connect the jobscript’s standard input directly to a file
-J|--job-name         Set the name of the job
--mail-user           Define the mail address for notifications
--mail-type           When to send mail notifications. Options: BEGIN,END,FAIL,ALL,..
-N|--nodes            Number of compute nodes used by the job
-n|--ntasks           Number of tasks (MPI processes)
--ntasks-per-node     Number of tasks per compute node
-o|--output           Path to the job's standard output
-p|--partition        Partition to be used from the job
-t|--time             Maximum wall-clock time of the job!!! Please estimate properly !!!
--gres                Request nodes with specific Generic Resources
--gpus-per-node       Number of gpus per node, similar to --gres=gpu:N
--disable-turbomode   Disable CPU turbo mode
--globres             Custom global resources, like filesystem licenses
```

Slurm - Submission filter

- Slurm is using a submission filter with the following functionality:
  - Deny jobs requesting multiple partitions, we allow only one.
  - Disable the `--requeue` option. We do not allow users to re-queue their jobs.
  - Deny submission if budget account was not defined (with `--account` or `-A`).
  - By default add the memory GRES when missing, users can always specify the memory GRES if they want.
  - Add the gpu GRES when missing for jobs on gpu partitions. By default the max number of available GPUs per node is being used, users can specify different number manually.
  - Deny jobs with wrong memory GRES, e.g. job submitted to `mem512` partition with GRES `mem128`.
- Examples:
  - Submit a job in the `gpus` partition requesting 4 GPUs per node:

```
sbatch -N 2 -p gpus -A <budgetID> --gres=gpu:4 <job-script>
```

  - Submit a job in the `mem512` partition:

```
sbatch -N 4 -p mem512 -A <budgetID> --gres=mem512 <job-script>
```

Slurm - Spawning command

- With **`srun`** the users can spawn any kind of application, process or task inside a job allocation. srun should be used either:
  - Inside a job script submitted by sbatch (starts a job-step).
  - After calling salloc (execute programs interactively).
- Command format:
  - `srun [options..] <executable> [args..]`
- `srun` accepts almost all allocation options of `sbatch` and `salloc`. There are however some other unique options:

```
--forward-x          Enable X11 forwarding only for interactive jobs.
--pty                Execute a task in pseudo terminal mode.
--multiprog <file>   Run different programs with different arguments per task specified in a text file.
--exact              Allow a step access to only the resources requested for the step.
--overlap            Allow steps to overlap each other on the CPUs.
```

Note: In order to spawn MPI applications the users *should* always use `srun` and not `mpiexec`.

Jobscript - Serial job

- Instead of passing options to `sbatch` from the command-line, it is better to specify these options using the `#SBATCH` directives inside the job scripts which must be positioned in the very beginning of the jobscript!
- Here is a simple example where some system commands are executed inside the jobscript. This job will have the name “TestJob”. One compute node will be allocated for 30 minutes. Output will be written in the defined files. The job will run in the default partition (depends on cluster).

```
#!/bin/bash

#SBATCH -J TestJob
#SBATCH -N 1
#SBATCH -o TestJob-%j.out
#SBATCH -e TestJob-%j.err
#SBATCH --time=30
#SBATCH -A <budgetID>

sleep 5

hostname
```

Jobscript - Tasks in parallel

- Here is a simple example of a jobscript where we allocate 4 compute nodes for 1 hour. Inside the jobscript with the srun command we request to execute on 4 nodes with 2 processes per node the system command hostname, requesting a walltime of 10 minutes. In order to start a parallel job, users have to use the `srun` command that will spawn processes on the allocated compute nodes of the job.

```
#!/bin/bash

#SBATCH -J TestJob
#SBATCH -N 4
#SBATCH -o TestJob-%j.out
#SBATCH -e TestJob-%j.err
#SBATCH --time=10
#SBATCH -A <budgetID>

srun --ntasks-per-node=2 hostname
```

Jobscript - OpenMP job

- In this example the job will execute an **OpenMP** application named `omp-prog`. The allocation is for 1 node and by default, since there is no node-sharing, all CPUs of the node are available for the application. The output filenames are also defined and a walltime of 2 hours is requested. Note: It is important to define and export the variable **`OMP_NUM_THREADS`** that will be used by the executable.

```
#!/bin/bash

#SBATCH -J TestOMP
#SBATCH -N 1
#SBATCH -o TestOMP-%j.out
#SBATCH -e TestOMP-%j.err
#SBATCH --time=02:00:00
#SBATCH -A <budgetID>

export OMP_NUM_THREADS=48

/home/user/test/omp-prog
```

Jobscript - MPI job

- In the following example, an **MPI** application will start 144 tasks on 3 nodes running 48 tasks per node requesting a wall-time limit of 45 minutes in `batch` partition. Each MPI task will run on a separate core of the CPU. Users can change the modules also inside the jobscript.

```
#!/bin/bash

#SBATCH --nodes=3
#SBATCH --ntasks=144
#SBATCH --output=mpi-out.%j
#SBATCH --error=mpi-err.%j
#SBATCH --time=00:45:00
#SBATCH --partition=batch
#SBATCH -A <budgetID>

module purge
module load Intel ParaStationMPI

srun ./mpi-prog   # implied --ntasks-per-node=48
```

Jobscript - Hybrid job

- In this example, a hybrid job which uses both **MPI** and **OpenMP** is presented. This job will allocate 5 compute nodes for 2 hours. The job will have 30 MPI tasks in total, 6 tasks per node and 4 OpenMP threads per task. **Note**: It is important to define the environment variable `OMP_NUM_THREADS` and this must match with the value of the option `--cpus-per-task|-c`.

```
#!/bin/bash

#SBATCH -J TestJob
#SBATCH -N 5
#SBATCH -o TestJob-%j.out
#SBATCH -e TestJob-%j.err
#SBATCH --time= 02:00:00
#SBATCH --partition=large
#SBATCH -A <budgetID>

export OMP_NUM_THREADS=4

srun -N 5 --ntasks-per-node=6 --cpus-per-task=4 ./hybrid-prog
```

Multiple jobsteps

- Slurm introduces the concept of **jobsteps**. A jobstep can be viewed as a smaller job or allocation inside the current allocation. Jobsteps can be started only with the `srun` command.
- The following example shows the usage of jobsteps. With sbatch we allocate 32 compute nodes for 6 hours. Then we spawn 3 jobsteps. The first step will run on 16 compute nodes for 50 minutes, the second step on 2 nodes for 10 minutes and the third step will use all 32 allocated nodes for 5 hours.

```
#!/bin/bash
#SBATCH -N 32 -t 06:00:00 -p batch -A <budgetID>

srun -N 16 -n 32 -t 00:50:00 ./mpi-prog1
srun -N 2 -n 4 -t 00:10:00 ./mpi-prog2
srun -N 32 --ntasks-per-node=2 -t 05:00:00 ./mpi-prog3
```

- To run multiple jobsteps in parallel you will have to run the `srun` commands in background with "`&`" and then call the `wait` command afterwards. With the `--exact` option each jobstep will use only the specified resources and will not block resources from the other ones:

```
#!/bin/bash
#SBATCH -N 10 -t 05:00:00 -p batch -A <budgetID>

srun --exact -N 2 -ntasks-per-node=24 -c 1 ./mpi-prog1 &
srun --exact -N 8 --ntasks-per-node=2 -c 1 ./mpi-prog2 &
wait
```

Job dependencies & job-chains

- Slurm supports dependency chains which are collections of batch jobs with defined dependencies. Job dependencies can be defined using the `--dependency|-d` option of `sbatch` command. The format is:
  - `sbatch -d <type>:<jobID> <jobscript>`
  - Available dependency types: `afterany, afternotok, afterok, ..`
- Below is an example of a bash script for starting a chain of jobs. The script submits a chain of `$NO_OF_JOBS`. Each job will start only after successful completion of its predecessor.

```
#!/bin/bash

NO_OF_JOBS=<no-of-jobs>
JOB_SCRIPT=<jobscript-name>

JOBID=$(sbatch ${JOB_SCRIPT} 2>&1 | awk '{print $(NF)}')

I=0
while [ ${I} -le ${NO_OF_JOBS} ]; do
  JOBID=$(sbatch -d afterok:${JOBID} ${JOB_SCRIPT} 2>&1 | awk '{print $(NF)}')
  let I=${I}+1
done
```

Job arrays

- Slurm supports job-arrays which can be defined using the option `--array|-a` of `sbatch` command. To address a job-array, Slurm provides a base array ID and an array index for each job. The syntax for specifying an array-job is: `--array=<range of indices>`. Their jobids have format like: `${BASE_JOB_ID}_${TASK_ID}`.
- Slurm exports also 2 env. variables that can be used in the job scripts:
  - `SLURM_ARRAY_JOB_ID` : base array jobid, replaces `%A` in jobscript
  - `SLURM_ARRAY_TASK_ID`: array index, replaces `%a` in jobscript

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --output=prog-%A_%a.out
#SBATCH --error=prog-%A_%a.err
#SBATCH --time=02:00:00
#SBATCH --array=1-20
#SBATCH -A <budgetID>

srun -N 1 --ntasks-per-node=1  ./prog input${SLURM_ARRAY_TASK_ID}.txt
```

Heterogeneous jobs

- Submission commands:
  - Multiple independent job specifications identified in command line separated by "`:`".
    - `salloc -N 2 -p gpus : -N 16 -p batch`
  - The job specifications are sent to slurmctld daemon as a list in a single RPC.
  - The entire request is validated and accepted or rejected.
- Job scripts:
  - Use "`#SBATCH hetjob`" between the other `#SBATCH` options in jobscript to separate job packs and their groups of resources.
  - With `srun` and the "`:`" format you can spawn jobsteps using heterogeneous resources. With the `srun`'s option `--het-group` you can define which het-group of resources will be used for the jobsteps. An example of a jobscript:

```
#!/bin/bash
#SBATCH -N 32 --ntasks-per-node=8 -p batch
#SBATCH hetjob
#SBATCH -N 1 --ntasks-per-node=1 -p batch
#SBATCH hetjob
#SBATCH -N 16 --gres=gpu:4--ntasks-per-node=4 -p gpus

srun --het-group=1 exec1 : --het-group=0 exec1 : --het-group=2 exec2
```

Interactive jobs

- Interactive sessions can be allocated using the `salloc` command. The following command will allocate 8 nodes for 30 minutes:

```
@login$ salloc -A <budget-ID> -p <partition> --nodes=8 -t 00:30:00
```

- After a successful allocation, salloc will start a **new shell** on the login node where the submission happened. After the allocation the users can execute `srun` in order to spawn interactively their applications on the compute nodes. The interactive session is terminated by exiting the shell.

```
@login$ srun -N 4 --ntasks-per-node=2 -t 00:10:00 ./mpi-prog
```

- It is possible to obtain a remote shell on the a node, after salloc, by running srun with the following arguments:

```
@login$ srun -N 1 -n 1 --interactive -t 20 --pty /bin/bash
@compute$ srun -N 8 --ntasks-per-node=4 app-exec
```

With the `--interactive` option the jobstep will not block any resources from the allocated nodes, all available resources will be free to be consumed by the jobsteps that can be spawned afterwards.

Further Information

- Updated status of the systems:
  - Read “Message of the day” on login nodes.
  - JSC status website: https://status.jsc.fz-juelich.de
- Check the online documentation of each system.
- User support at FZJ:
  - **Email**: sc@fz-juelich.de
  - **Phone**: +49 2461 61-2828
- Highlights:
  - Please try to estimate and give **proper walltime limit** for your jobs! This will help Slurm to do better scheduling! Avoid giving max. walltime (e.g. 24 hours) to all jobs..
- Current state & future plans:
  - On our clusters current Slurm version is 22.05, we will move to Slurm 23.02 soon.
  - When we move to 23.02 there will be some pinning changes and we will disable by default SMT.

JÜLICH
Forschungszentrum

Questions?