Job Scheduling with SLURM

In addition to the shared disc space there is also shared computation power in the form of a computer cluster, meaning the connection of many computers to allow parallel and distributed processing. Simply, this allows for the speedup of calculation by spreading the compute load to multiple computers.

The use of the cluster is simple and it is controlled by a queuing system, in our case SLURM. A user simply writes a short script specifying the programs, parameters and files together with a control script, and passes this information to the queuing system, which in turn will distribute the task on the cluster, if there are enough free slots. If too many people are using the cluster at the same time, some tasks will remain in the queue until the next slot becomes available. In the following you will find a quick glance on how to use SLURM. The following video clips (sorry for the poor quality) may help:

- How2Slurm1: Overview of the ApplBio Computer system
- How2Slurm2: Introducing SLURM and the concept of control scripts
- How2Slurm3: Control scripts in detail

Architecture

Slurm has a centralized manager, slurmctld, to monitor resources and work. There may also be a backup manager to assume those responsibilities in the event of failure. Each compute server (node) has a slurmd daemon, which can be compared to a remote shell: it waits for work, executes that work, returns status, and waits for more work. The slurmd daemons provide fault-tolerant hierarchical communications. There is an optional slurmdbd (Slurm DataBase Daemon) which can be used to record accounting information for multiple Slurm-managed clusters in a single database.

User tools include *sbatch* to initiate jobs, *scancel* to terminate queued or running jobs, *sinfo* to report system status, *squeue* to report the status of jobs, and *sacct* to get information about jobs and job steps that are running or have completed. The *smap* and *sview* commands graphically reports system and job status including network topology. There is an administrative tool *scontrol* available to monitor and/or modify configuration and state information on the cluster. The administrative tool used to manage the database is sacctmgr. It can be used to identify the clusters, valid users, valid bank accounts, etc. APIs are available for all functions.

Creating a script file

The script file tells SLURM what should be done. The example below serves as a general guideline. Please do not simply copy it, but instead try to understand what information is provided to SLURM, and then modify the information to match your requirements. This applies, in particular, to the options –`cpu-per-task` and –`mem-per-cpu`. If you set the values for these options too high, your job will either remain pending until the requested resources are free on the cluster, or alternatively, your job will run and will then block the requested resources although they will not be used.

Example

This example uses a simple seed file that provides input for the program calls, e.g. file names, parameter values and the like

You can use any text editor like nano to create a file, you can then paste the contents of the example file below into that file. Exchange the example command with the actual command you want to use and don't forget to modify the CPU and memory requests.

```
#!/bin/bash
#SBATCH --partition=pool                # specifies the queue you are
submitting to
#SBATCH --account=intern                # specifies your role in the system
#SBATCH --cpus-per-task=1               # how many cpus will you require per
task (be reasonable here)
#SBATCH --mem-per-cpu=1mb               # how much RAM will you need (be
reasonable here)
#SBATCH --job-name="Test"               # self-explanatory

echo "Hello World"                      # your command goes here
```

Job submission

Use the command *sbatch* to submit the job to SLURM. You simply use the path to the script you just created for this

```
sbatch /path/to/<slurm_script>.sh
```

After your script was send to the cluster, SLURM will automatically create a file in the directory from which you issued the `sbatch` command. This file will contain all information that would be printed to the terminal when calling your command in the local terminal. This directory will also be used as the current directory for all relative paths in your slurm script.

Your script will run using your current (Anaconda) environment meaning that you can use all programs which are in the environment from which you issue the `sbatch` command.

Job status

Check Job and Task Status using the command squeue

```
squeue -u your_username
```

Note, if you omit the option `-u your_username` then you will get information about all jobs currently managed via SLURM.

See this page, for example, for an overview of the job status codes.

Cancel jobs

Use the command scancel to stop pending and running jobs

```
scancel -j jobid
```

Make sure that you double check that you cancel the correct job id

Information about slurm settings on our system

Use the command sinfo to see the available queues/partitions and the assigned computers

```
sinfo
```

More info of what is going on

To open a window with a whole array of informations about the cluster like current running jobs/partitions/reservations/available resources and more run

```
 sview
```

Common mistakes

This is a non-exhaustive list of issues and mistakes in the context of SLURM

- you specify a non-existing queue
- the number of cpus that you reserve per task is higher or lower than the number of cpus the task is actually using. It is a bad idea to reserve more cpus than needed, because you block resources that are needed by other people in the group!
- the amount of memory you reserve per task is higher or lower than the amount needed by your task. It is a bad idea to reserve more memory than needed, because you block resources that are needed by other people in the group!
- a program is not running because it is not installed on the remote server. Solution: Report to the tutor or the systems administrator
- your job is in pending state forever → it might be that non of the computers in the queue has the capacity[1] that you specified

- computerenvironment

[1]
e.g. number of cpus, amount of memory