Running parallel jobs on the Linux-Cluster

On all HPC systems at LRZ, the SLURM scheduler is used to execute parallel jobs. This document describes usage, policies and resources available for submission and management of such jobs.

- Examples, Policies and Commands
- Detailed Instructions (for Beginners)
  - Interactive SLURM shell for parallel testing
  - Batch Jobs
    - Step 1: Edit a job script
    - Step 2: Submission procedure
    - Step 3: Checking the status of a job
    - Inspection and modification of jobs
    - Deleting jobs from the queue

Examples, Policies and Commands

<table>
<thead>
<tr>
<th>Examples</th>
<th>provides example job scripts which cover the most common usage patterns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policies</td>
<td>provides information about the policies, such as memory limits, run time limits etc.</td>
</tr>
<tr>
<td>SLURM Workload Manager</td>
<td>lists SLURM commands and options, and explains them, making appropriate recommendations where necessary. Provides hints for aborting jobs.</td>
</tr>
</tbody>
</table>

Detailed Instructions (for Beginners)

All parallel programs in the parallel segments of the cluster must be started up using either
- an interactive SLURM shell
- a SLURM batch script

In order to access the SLURM infrastructure described here, please first log in to a login node of the cluster as described in the introduction.

This document provides information on how to configure, submit and execute SLURM jobs, as well as information about batch processing policies. In particular, please be aware that misuse of the resources described here can result in the invalidation of the violating account. In particular, all parallel runs should always use either a salloc shell (for testing) or a scripted SLURM job.

Interactive SLURM shell for parallel testing

For performing program testing and short runs the following sequence of commands can be used: First, salloc is invoked to reserve the needed resources. Then, mpiexec can be used to start up a program on these resources.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>salloc srn ./myprog.exe exit</td>
<td>Please invoke this command from lxlogin5,6,7.</td>
</tr>
<tr>
<td>salloc --ntasks=56 mpiexec ./myprog.exe exit</td>
<td>Start an MPP mode Intel MPI program using two 28-way nodes on the CooLMUC-2 cluster. Please invoke this command from lxlogin5,6,7.</td>
</tr>
<tr>
<td>salloc --ntasks=8 --cpus-per-task=7 export OMP_NUM_THREADS=7 mpiexec -n 6 ./myprog.exe exit</td>
<td>Start a hybrid mode Intel MPI program on the CooLMUC-2 cluster using 8 MPI tasks, with 7 OpenMP threads per task (2 nodes will be needed). Please invoke this command from lxlogin5,6,7. Note that currently there are 28 cores per node available on CooLMUC-2, but no hyperthreading.</td>
</tr>
</tbody>
</table>
Start a hybrid mode Intel MPI program on the CoolMUC-3 cluster using 16 MPI tasks, with 8 OpenMP threads per task, distributed across 2 nodes.

Please invoke this command from lxlogin8. Note that there are 64 physical cores per node available on CoolMUC-3, and due to hyperthreading the logical core count on each node is 256. In this example, the hyperthreads are not used, but you could increase the value of OMP_NUM_THREADS to make use of them. A non-hybrid program will need to use OMP_NUM_THREADS=1, and --tasks-per-node can have any value up to 64 (larger values may result in failure to start).

Notes and Warnings:

- Only application/commands which are started with mpiexec are executed on the allocated nodes. All other commands will still be executed on the login node. This might block the login node for other users. A workaround would be to start memory or time consuming commands with "mpiexec -n 1", even if they are serial, optionally packing them into a script and starting it with mpiexec. Try "mpiexec -n 2 hostname" and compare the output with that of just typing "hostname".
- SLURM also has its own srun command which you can try to use.
- Once the allocation expires, the program will be signalled and killed; further programs can not be started. Please issue the exit command and start a new allocation.

Batch Jobs

This type of execution method should be used for all production runs. A step-by-step recipe for the simplest type of parallel job is given, illustrating the use of the SLURM commands for users of the bash shell. See the documentation section at the end for pointers to more complex setups.

Step 1: Edit a job script

The following script is assumed to be stored in the file myjob.cmd.

```bash
#!/bin/bash

#SBATCH -o /home/hpc/<group>/<user>/myjob.%j.%N.out (Placeholder) standard output and error go there. Note that the directory where the output file is placed must exist before the job starts, and the full path name must be specified (no environment variable!). The %j encodes the job ID into the output file. The %N encodes the master node of the job and should be added since job IDs from different SLURM clusters may be the same.

#SBATCH -D /home/hpc/<group>/<user>/mydir directory used by script as starting point (working directory)

#SBATCH -J <job_name> (Placeholder) name of job (not more than 10 characters please)

#SBATCH --clusters=mpp2 The name "mpp2" specifies the cluster to be used - here the CoolMUC-2 MPP cluster.

#SBATCH --get-user-env Set user environment properly

#SBATCH --ntasks=64 Number of MPI tasks assigned to job. By default, SLURM will start as many MPI tasks per node as there are physical cores in the node.

#SBATCH --mail-type=end Send an e-mail at job completion

#SBATCH --mail-user=<email_address>@<domain> (Placeholder) e-mail address (don't forget, and please enter a valid address!)

#SBATCH --export=NONE Do not export the environment of the submitting shell into the job; while SLURM allows to also use ALL here, this is strongly discouraged, because the submission environment is very likely to be inconsistent with the environment required for execution of the job.

#SBATCH --time=08:00:00 maximum run time is 8 hours 0 minutes 0 seconds; this may be increased up to the queue limit

source /etc/profile.d/modules.sh initialize module system
module load gsl # ... etc

load any required environment modules (may be needed if program is linked against shared libraries). "gsl" of course is only a placeholder.

mpiexec ./my_mpi_prog.exe

start MPI executable. The MPI variant used depends on the loaded module set; non-MPI programs may fail to start up - please consult the example jobs or the software-specific documentation for other startup mechanisms.

This script essentially looks like a bash script. However, there are specially marked comment lines ("control sequences"), which have a special meaning in the SLURM context explained on the right hand of the above table. The entries marked "Placeholder" must be suitably modified to have valid user-specific values.

For this script, the environment of the submitting shell will not be exported to the job's environment; the job will start 64 MPI tasks on as many cores.

**Step 2: Submission procedure**

The job script is submitted to the queue via the command

```
sbatch myjob.cmd
```

At submission time the control sequences are evaluated and stored in the queuing database, and the script is copied into an internal directory for later execution. If the command was executed successfully, the Job ID will be returned as follows:

```
Submitted batch job 65648.
```

It is a good idea to note down your Job ID's, for example to provide to LRZ HPC support as information if anything goes wrong. The submission command can also contain control sequences. For example,

```
sbatch --time=12:00:00 myjob.cmd
```

would override the setting inside the script, forcing it to run 12 instead of 8 hours.

**Step 3: Checking the status of a job**

Once submitted, the job will be queued for some time, depending on how many jobs are presently submitted. Eventually, more or less after previously submitted jobs have completed, the job will be started on one or more of the systems determined by its resource requirements. The status of the job can be queried with the squeue --clusters=[all | cluster_name] command, which will give an output like

```
CLUSTER: mpp1
JOBD PARTITION  NAME  USER  ST  TIME   NODES NODELIST(REASON)
65646 mpp1_batch job1  xyz1  R  24:19  2     lxa[7-8]
65647 mpp1_batch myj  xza2  R  0:09   1     lxa14
65648 mpp1_batch calc yaz7  PD 0:00   6     (Resources)
```

(assuming mpp1 is specified as the clusters argument) indicating that the job is queued. Once the job is running, the output would indicate the state to be "R" (=running), and would also list the host(s) it was running on. For jobs that have not yet started, the --start option, applied to kbd>squeue, will provide an estimate (!) for the starting time. The sinfo --clusters=[all | cluster_name] command prints out an overview of the status of all clusters or a particular clusters in the SLURM configuration.

**Inspection and modification of jobs**

Queued jobs can be inspected for their characteristics via the command

```
scontrol --clusters=<cluster_name> show jobid=<job ID>
```

which will print out a list of "Keyword=Value" pairs which characterize the job. As long as a job is waiting in the queue, it is possible to modify at least some of these; for example, the command

```
scontrol --clusters=<cluster_name> update jobid=65648 TimeLimit=04:00:00
```

would change the run time limit of the above-mentioned example job from 8 hours to 4 hours.

**Deleting jobs from the queue**

To forcibly remove a job from SLURM, the command

```
scancel --clusters=<cluster_name> <JOB_ID>
```

can be used. Please do not forget to specify the cluster! The scancel (1) man page provides further information on the use of this command.