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, a suitably parameterized call to a parallel program binary (usually mpiexec) is used to start up that program, using the resources assigned by SLURM.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>salloc --partition=mpp2_inter export OMP_NUM_TH READS=28 srun ./myprog.exe exit</td>
<td>Use the srun command only for pure OpenMP Jobs. Example shows use of a CoolMUC-2 node.</td>
</tr>
<tr>
<td>salloc --ntasks=56 --partition=mpp2_inter mpiexec ./myprog.exe exit</td>
<td>Start an MPP mode Intel MPI program using two 28-way nodes on the CoolMUC-2 cluster</td>
</tr>
</tbody>
</table>
salloc --ntasks=8 --cpus-per-task=7 --partition=mpp2_inter
export OMP_NUM_TH READS=7
mpiexec -n 8 ./myprog.exe
exit

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).

Note that currently there are 28 cores per node available on CoolMUC-2, but no hyperthreading.

salloc --nodes=2 --tasks-per-node=3 --partition=mpp3_inter
export OMP_NUM_TH READS=8
mpiexec -n 6 ./myprog.exe
exit

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.

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).

by default, a SLURM shell generated via salloc will run for 15 minutes. This interval can be extended to the partition maximum by specifying a suitable --time=hh:mm:ss argument. Also, the --partition option can be used to explicitly specify a desired partition, but we advise against doing so because the different login nodes have different environments that are incompatible.

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.
- Use of SLURM's own srun command to start up parallel programs may not always work as desired.
- 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.

```
#!/bin/bash

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

#SBATCH -o ./%x.%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 %x encodes the job name into the output file. The %j encodes the job ID into the output file. The %N encodes the master node of the job and can be added if job IDs from different SLURM clusters might be the same. Here, the specified path is relative to the directory specified in the -D spec.

#SBATCH --D ./ directory used by script as starting point (working directory). The directory specified must exist. Here, the path is relative to the submission directory.

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

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

#SBATCH --nodes=8 Number of (shared-memory multi-core) nodes assigned to the job.

#SBATCH --ntasks-per-node=28 The number of MPI tasks to start on each node. Typically, the value used here should not be larger than the number of physical cores in a node. It may be chosen smaller for various reasons (memory needed for a task, hybrid programs, etc).
```
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.

### 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: mpp2
JOBID PARTITION   NAME   USER  ST TIME   NODES NODELIST(REASON)
65646 mpp2_batch job1  xyz1 R  24:19  2     ....
65647 mpp2_batch myj  xza2  R  0:09   1     ....
65648 mpp2_batch calc yaz7 PD 0:00   6     (Resources)
```

(assuming mpp2 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>queue`, 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.