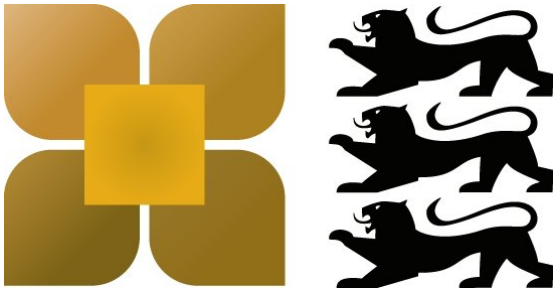
Resource management on bwUniCluster



bwHPC support team Hohenheim

How to get resources on bwUniCluster

- bwUniCluster uses the resource manager SLURM
- You request resources for a fixed amount of time (at most 3 days).
- There are three ways to request resources (=computation cores and nodes including memory) and run commands
  - Via executing a bash script by **sbatch**
    - SLURM runs your commands from the script, starts as soon as resources are granted
  - Via interactive mode, initialised by **salloc**
    - Good for experimenting, debugging
    - Resources start when they are granted (time runs whether you do sth. Or not)
  - (Via running a single executable with SLURM command **srun**)

A SLURM workflow

(1) User creates a **job script** and submits it to Slurm via the "**sbatch**" command

```
#!/bin/bash
#SBATCH -p dev_cpuonly
#SBATCH -N 1 -n 1
#SBATCH -t 00:01:00
#SBATCH -mem-per-cpu=50

./your_simulation
```

(2) Slurm parses the job script:
→ where & when to run job

Slurm
(central management daemon)

(3) Job execution:
delegated to resource manager on the node

compute resources:
Slurm compute node daemon

(4) The resource manager executes the job and communicates status information to nodes

from R. Häfner's slides, KIT

The setup of bwUniCluster - nodes and queues

- Login nodes: This is where the command line operates after you log in
  - Don't use these for computations
  - To create workspaces, request resources,...
- Most other nodes: computation nodes, the resources that SLURM can distribute among the users
- Computation nodes are organized in queues (see next slides for some queues)
- Queues differ in purpose and maximum resources (time, # cores, # nodes, memory, # GPUs, ...) and underlying hardware
- When you request resources, you specify your queue
- 3 specific development queues dev_single, dev_multiple, dev_gpu_4

Queues on bwUniCluster (more details on bwHPC-wiki)

Queue	What for?	Properties
dev_single	Development -single node	max 30 min, 40 cores
dev_multiple	Development - multiple nodes	max 30 min, 4 nodes with 40 cores
dev_gpu_4	Development - GPU single node	max 30 min, 40 nodes
single	Single node	max 72 h, 40 cores, max 180000mb memory
multiple	Multiple nodes - many cores	max 72 h, 128 nodes w. 40 cores
fat	Single node - high memory	max 72 h, 80 cores, min 180001mb memory
gpu_4	With 4 GPUs	max 48 h, 14 nodes w. 40 cores
gpu_8	With 8 GPUs and higher memory	max 48 h, 10 nodes w. 40 cores

How quickly can you get resources?

- Many users want resources, resources are allocated via a fairshare scheme and dependent on
  - Your requested resources & the queue
  - Your request history (more past use, less priority)
  - The share of your university (money spend to buy/maintain resources, currently 3.2 % of all resources for U. Hohenheim)
  - Total usage of the system

You can check available nodes for each queue via bash command **sinfo_t_idle**

```
[ho_ffreund@uc2n995 ~]$ sinfo_t_idle
Partition dev_single     :      5 nodes idle
Partition single         :     39 nodes idle
Partition dev_multiple   :      4 nodes idle
Partition multiple       :     22 nodes idle
Partition fat            :      0 nodes idle
Partition dev_multiple_e:      3 nodes idle
Partition multiple_e     :      2 nodes idle
Partition dev_special    :      2 nodes idle
Partition special        :      1 nodes idle
Partition gpu_4          :      0 nodes idle
Partition dev_gpu_4      :      1 nodes idle
Partition gpu_8          :      0 nodes idle
```

SBATCH scripts: bash scripts with resource requests

```
#!/bin/sh
########## Begin Slurm header ##########
#
# Give job a reasonable name
#SBATCH --job-name=echotest
#
# Request number of nodes for job
#SBATCH --ntasks=1
#
#
# Maximum run time of job (hh:mm:ss)
#SBATCH --time=00:01:00
#
#SBATCH -p dev_single
########### End Slurm header ##########

echo "Test"

#Show some more useful environment variables

echo "Working Directory:                $PWD"
echo "Running on host                   `hostname`"
echo "Job id:                           $SLURM_JOB_ID"
echo "Job name:                         $SLURM_JOB_NAME"
echo "Number of nodes allocated to job: $SLURM_NNODES"
echo "Number of cores allocated to job: $SLURM_NTASKS"
```

always the shebang

Each line starting with #SBATCH ... states a resource or option request for SLURM

(other lines are comments which are NOT executed)

bash commands to run - the “actual” script

SBATCH scripts: Most important SBATCH parameters

#SBATCH	What it does
#SBATCH --time=*hh:mm:ss*	Maximum time the resources are available
#SBATCH --nodes=*count*	Number of nodes
#SBATCH --ntasks=*count*	Number of tasks
#SBATCH --ntasks-per-node=*count*	Number of tasks per node (2 tasks/CPU)
#SBATCH --mem-per-cpu=*value*	Number of memory per CPU in MB (has default)
#SBATCH --partition=*queuename*	Which queue to use (MANDATORY)
#SBATCH --job-name=*name*	Name of job
#SBATCH --mail-type=*which*	Should SLURM send emails when job starts, ends, fails? NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-user=*address*	Your email address (for --mail-type)
#SBATCH --output=*name*	Change name of SLURM output
#SBATCH --reservation=*name*	Use reserved resources within a queue

Bigger list in bwHPC-wiki (here, also check queue settings).
For GPU use see here , request GPUs via #SBATCH --gres=gpu:<# GPUs>

Submitting a job

- After you have prepared the SBATCH script, you can request that SLURM executes it via **sbatch <sbatch script name>**

```
Submitted batch job 20234802
```

- If all your resource requests are possible within the requested queue: SLURM queues your job. Your job is done!

- Check your job status with **squeue --long**

```
Wed Dec 01 11:45:31 2021
             JOBID PARTITION     NAME     USER    STATE       TIME TIME_LIMI  NODES NODELIST(REASON)
          20234714 dev_singl echotest ho_ffreu  PENDING       0:00     10:00      1 (None)
```

- Get estimate of starting time of jobs via **squeue --start**

Exercise

- Copy the file /pfs/work7/workspace/scratch/ho_ffreund-bwhpcintro/job1.sh to your work space
- Try to submit it via sbatch - it fails! Why?
- Fix it (solution script: job1f.sh in the directory), submit it with **sbatch**

Postprocessing a job

- By default, all command line output and some further information is written to **slurm-<job-d>.out** . This is the output of my run of job1f.sh (in nano):

```
  GNU nano 2.9.8                        slurm-20234802.out

Test
Working Directory:                      /pfs/work7/workspace/scratch/ho_ffreund-bwhpcintro
Running on host                         uc2n361.localdomain
Job id:                                 20234802
Job name:                               echotest
Number of nodes allocated to job:       1
Number of cores allocated to job:       1

============================== JOB FEEDBACK ==============================

NodeName=uc2n361
Job ID: 20234802
Cluster: uc2
User/Group: ho_ffreund/ho_kim
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 2
CPU Utilized: 00:00:00
CPU Efficiency: 0.00% of 00:00:50 core-walltime
Job Wall-clock time: 00:00:25
Memory Utilized: 2.26 MB
Memory Efficiency: 0.10% of 2.20 GB
```

Output of your commands

Info added by SLURM on how the job ran

Running jobs

- Further information on running jobs via **scontrol show job <job id>**
- You can also check the slurm-... output file while your job runs
- Cancel your job: $ **scancel <job id>** (check your running job id's with **squeue**)

Finished jobs

- **sacct** shows accounting data for job and job steps
  - Short info: $ **sacct -j <job id>**
  - Much more info: $ **sacct -l -j <job id>**
  - List all recent job ids: $ **sacct -X** (since 0:00, see **man sacct** how to change it)

Exercise: Check your job's accounting data

Interactive jobs

- You can also request resources to be granted directly on the command line via **salloc**
- You can specify the SBATCH options as salloc options,
- e.g. for single node resources
  $ **salloc -p <queue> -n <how many tasks?> -t <time in min>**
- To end the interactive session (brings you back to the login node)
  $ **exit**
- See bwHPC-wiki for more information

Some good practice rules

- Test your programs before submitting the “real” job via sbatch
  - Run it on toy examples, subsets of your data
  - If it helps: run in interactive mode
  - Run prototype on local machine

- Estimate running times, think about/check resource consumption (especially #threads/cores/nodes)

- Use your resources efficiently (if possible), e.g. try to prevent idle cores for long times (advanced users: code efficiently).

- Cancel jobs that you don’t need/know that they will fail.

This is not only nice for the other users, but saves a meaningful amount of your time by improvement of your code and decreased waiting times for bwUniCluster resources. Also keep in mind: It may save energy!

Soon: More information on good practice/energy saving on bwhpc.de or on the Wiki

About parallelisation

- Parallel computing: Specific parts (instances) of your computation are computed **at the same time** on different threads/cores/nodes. Compare it to painting different rooms in a flat - given enough people (=threads/core/nodes) can be done in parallel.

- If you don't use parallel computing, you essentially only run on a desktop computer with MUCH more memory -> usually not what you want

- Many programs have built-in parallelisation, programming languages allow for it (e.g. in R)

- The most powerful approaches are MP/MPI, but there are also easy-to-learn all-purpose approaches. There are resources and recurrent bwHPC courses on these topics (e.g., for MP/MPI, GNU parallel/srun).

- You can also run an array job or use the parbatch software module (see next section)