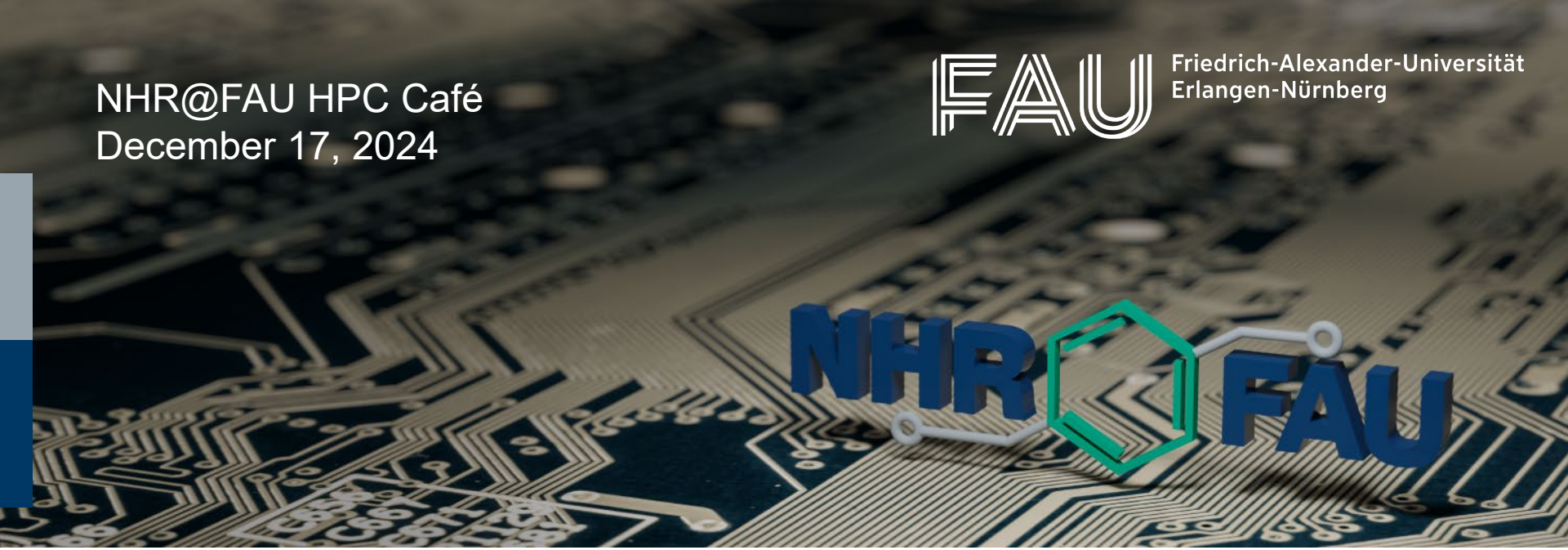
NHR@FAU HPC Café
December 17, 2024

Friedrich-Alexander-Universität
Erlangen-Nürnberg



Slurm – Best Practices and Advanced Use

Thomas Gruber, Georg Hager

Erlangen National High Performance Computing Center (NHR@FAU)

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Slurm Basics

Slurm documentation

- NHR@FAU
  - General: https://doc.nhr.fau.de/batch-processing/batch_system_slurm/
  - Cluster-specific: https://doc.nhr.fau.de/clusters/overview/

- Official Slurm documentation
  - Separate documentation for every command and the available options: https://slurm.schedmd.com/man_index.html
  - Slurm commands and their counterparts in different batch systems: https://slurm.schedmd.com/rosetta.pdf
  - Slurm tutorials: https://slurm.schedmd.com/tutorials.html

Terminology

- **Job**: allocation of resources assigned to a user for a specified amount of time
- **Partition**: set of nodes grouped by specific property (e.g. hardware); can have constraints on job size, time limit, permitted users, etc. → queues
- **Task**: how many instances of your command are executed; normally corresponds to number of MPI processes
- **Jobstep**: set of tasks within a job; a job can contain multiple job steps executing sequentially or in parallel
- **QoS** (Quality-of-Service): limits set on a per-group-basis (walltime, #GPUs, running jobs per group,…)
- **GRES**: generic resources, here: GPUs
- **CPU**: equivalent to hyperthread if configured; otherwise equivalent to core

Job script – general structure

```
#!/bin/bash -l
#
#SBATCH --nodes=2
#SBATCH --ntasks=20
#SBATCH --time=01:00:00
#SBATCH --output=myoutput_%x_%j.out
#SBATCH --cpu-freq=high-high:performance
#SBATCH --job-name=myJob
#SBATCH --export=NONE

unset SLURM_EXPORT_ENV

module load <modules>

srun ./application [options]
```

Script is interpreted as a bash script; `-l` is necessary for correct module initalization!

Job script – general structure

```
#!/bin/bash -l
#
#SBATCH --nodes=2
#SBATCH --ntasks=20
#SBATCH --time=01:00:00
#SBATCH --output=myoutput_%x_%j.out
#SBATCH --cpu-freq=high-high:performance
#SBATCH --job-name=myJob
#SBATCH --export=NONE

unset SLURM_EXPORT_ENV

module load <modules>

srun ./application [options]
```

Do not export environment from submitting shell

Enable export of environment from this script to `srun`; equivalent to `export SLURM_EXPORT_ENV=ALL`

Job script – general structure

```
#!/bin/bash -l
#
#SBATCH --nodes=2
#SBATCH --ntasks=20
#SBATCH --time=01:00:00
#SBATCH --output=myoutput_%x_%j.out
#SBATCH --cpu-freq=high-high:performance
#SBATCH --job-name=myJob
#SBATCH --export=NONE

unset SLURM_EXPORT_ENV

module load <modules>

srun ./application [options]
```

`--output` and `--error` are not recommended but if required, use SLURM's filename patterns

`%j`: job ID

`%x`: job name

`%a` array ID

`%N` hostname

… and more

Job script – general structure

```
#!/bin/bash -l
#
#SBATCH --nodes=2
#SBATCH --ntasks=20
#SBATCH --time=01:00:00
#SBATCH --output=myoutput_%x_%j.out
#SBATCH --cpu-freq=high-high:performance
#SBATCH --job-name=myJob
#SBATCH --export=NONE

unset SLURM_EXPORT_ENV

module load <modules>

srun ./application [options]
```

Configure job to use highest CPU frequency. Only in combination with `srun`!

Affinity control

- For hybrid jobs use the `--cpus-per-task=X` option
- General recommendation for your batch scripts:

```
export SLURM_CPU_BIND=cores
export SRUN_CPUS_PER_TASK=${SLURM_CPUS_PER_TASK:-1}
export OMP_NUM_THREADS=${SRUN_CPUS_PER_TASK}
export OMP_PROC_BIND=true
export OMP_PLACES=cores
```

- Do no use `mpirun` or `mpiexec`
  - If really required, use `--cpu-bind=none` and vendor specific options
  - Hybrid with IntelMPI: `I_MPI_PIN_DOMAIN=omp, I_MPI_PIN_ORDER=compact`
  - Hybrid with OpenMPI: `--bind-to core --map-by node:PE=$OMP_NUM_THREADS`

GPU jobs

- Previously discussed resource specifications are also applicable for GPU jobs
- Amount of host resources is determined by requested number of GPUs
- Share of host resources per GPU cannot be exceeded
- **`--ntasks`**/**`--cpus-per-task`** still have to be requested! Per default ntasks=1
- **`sinfo`** prints available partitions

- How to request GPUs?
  - **`--gres=gpu:<count>`** type is not important (only on clusters with **`work/any`** partition)
  - **`--gres=gpu:<type>:<count>`** request specific type
  - **`--gres=gpu:a100:<count> -C a100_80`** for A100 with 80 GB RAM (Alex only)

- TinyGPU: partition must be specified along with (matching) GPU type:
  - E.g., **`--gres=gpu:v100:1 -p v100`** request one V100

GPU jobs

```
#!/bin/bash -l
#
#SBATCH --ntasks=16                    #share for one GPU on Alex
#SBATCH --time=06:00:00
#SBATCH --gres=gpu:a40:1
#SBATCH --export=NONE

unset SLURM_EXPORT_ENV

module load <modules>

srun ./mpi_cuda_application
```

Testing of batch scripts

- Do not run the batch scripts on the frontends!
- Get a short interactive job and execute script by sourcing

```
# get sbatch options from script.sh
frontend $ grep -E "^#SBATCH" script.sh | cut -d' ' -f 2- | xargs
# get interactive job with reduced runtime
# (not all sbatch options are supported like --export)
frontend $ salloc <opts_from_above> -t 00:30:00
# Execute script by sourcing
computenode $ source script.sh
```

Misc topics – Configuration for performance tools

- We monitor the cluster nodes while jobs are running (ClusterCockpit)
- If you want to do own measurements with PAPI, Vtune or LIKWID, use `-C hwperf` with salloc or sbatch
- This disables some of the metrics in ClusterCockpit

- For clusters with shared nodes, `-C hwperf` only works in node-exclusive jobs: `sbatch --exclusive …`

Misc topics – My workflow takes longer than 24h!

- All of our systems have a maximum job runtime of 24h!

- In case of issues, contact hpc-support@fau.de
  - We might be able to give advice how to fix the issue
    - Checkpoint & Restart
    - Chain jobs
    - Code optimization/parallelization
  - Might require some work on your side (code and/or script changes)

SLURM job states and fair share

- If a job does not start, `squeue` prints the reason:
  - `Priority`: One or more higher priority jobs are queued
  - `Dependency`: This job is waiting for a dependent job to complete
  - `Resources`: The job is waiting for resources to become available
  - `ReqNodeNotAvail`: A node specifically required by the job is not currently available
  - `AssociationGroup<Resources>Limit`: All resources assigned to your association/group are currently in use
  - `QOSGrp<Resource>Limit`: All resources assigned to the specified QoS are currently in use
  - `Partition<Resource>Limit`: All resources assigned to the specified partition are currently in use
- Priority value depends on parameters like waiting time, partition, user group, and recently used CPU time (a.k.a. fairshare)

Monitoring your jobs

You can connect to nodes when a job is running to check it interactively:

```
$ srun --jobid=<jobID> --overlap --pty /bin/bash -l
```

- CPU jobs: use `top`/`htop`/`perf top` etc.
- GPU jobs: `nvidia-smi`

- ClusterCockpit: https://monitoring.nhr.fau.de/ (NHR users) or use the button in the HPC portal (Tier-3)
  - See HPC Café January 10, 2023: https://www.fau.tv/clip/id/46327

What we left out

- If you are a Tier-3 user but need more power, you can request access to NHR@FAU resources:
  - https://hpc.fau.de/tier3-access-to-alex/
  - https://hpc.fau.de/tier3-access-to-fritz/

- Multi-node GPU jobs on Alex are only allowed upon request

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Data staging

https://doc.nhr.fau.de/data/staging/

Previous HPC Cafes about this and similar topics:

- Jan 18, 2022: https://www.fau.tv/clip/id/40199
- Feb 6, 2024: https://hpc.fau.de/2024/01/29/monthly-hpc-cafe-efficient-data-handling-and-data-formats-february-6-hybrid-event/
- Oct 8, 2024: https://youtu.be/jRDd3zUQZE0

Problem statement

- Your jobs read and/or write a lot of data
  - The data is stored on a shared file system (`$WORK`, `$FASTTMP`)
  - Best data access: Large files, read/write sequentially
- Some access patterns are bad for performance
  - … of your own jobs
  - … of others' jobs working on the same file system
- Frequent metadata accesses slow down file system operations
  - Open/close in rapid succession
  - Parallel file systems (`$FASTTMP`) are especially prone to slowdowns (but all of them are)

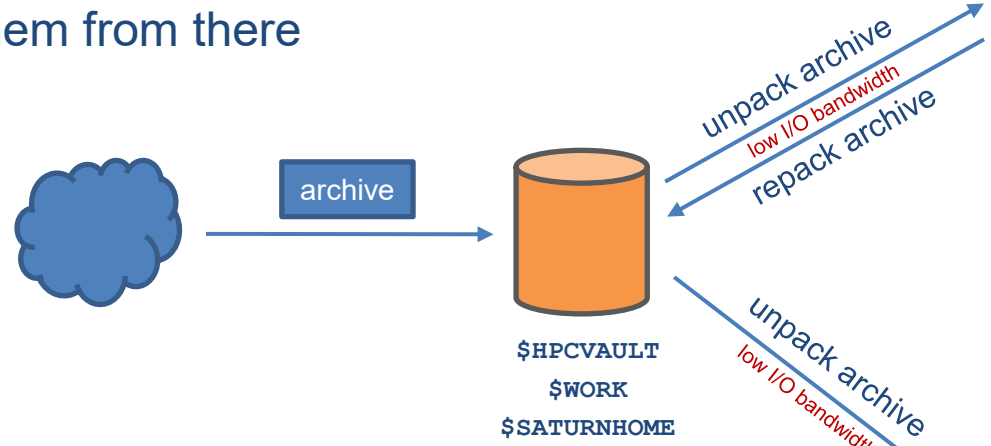
Not like the SSD in your laptop!

Archives and node-local disk ($TMPDIR)

Do not unpack archive to:

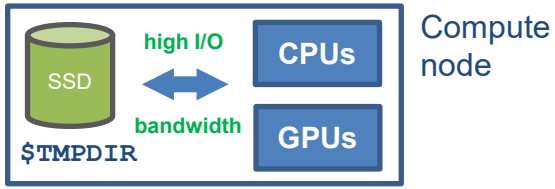
`$HOME/$HPCVAULT/$WORK`

Unpack files to node-local SSDs only and use them from there



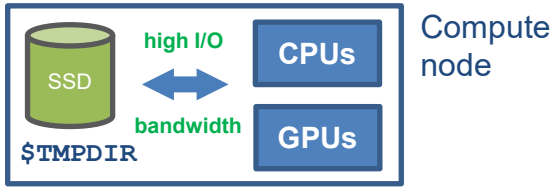
Optionally: if original archive must be altered

- unpack it to node local SSD (interactive job)
- optionally change files
- repack files and copy back to NFS



For simulation, training, …

- unpack archive to node local SSD
- perform simulation/training



Staging many small files

- Put files into ZIP/tar archive (better copy performance)
- Unpack to node-local temp directory and work from there
- Cleanup may be automatic

Job script

```
WORK_DIR=`mktemp -d -p $TMPDIR`
cd $WORK_DIR
unzip $WOODYHOME/foo.zip
# ... Now work with data in $WORK_DIR
# Clean up at the end:
cd
rm -rf $WORK_DIR
```

Other options

- Archive file formats
  - e.g., HDF5
  - Packs everything into a large file an uses special functions to access the data
  - Advantage: much lower metadata load
  - Disadvantage: Code support required
- Data streaming
  - Set up server for direct data download within job
  - Advantage: No shared file system space needed
  - Disadvantage: may be limited by network bandwidth
  - See https://hpc.fau.de/files/2024/02/HPC-cafe.pdf (HPC Café Feb 6, 2024)
- Workspaces
  - High-performance temporary networked storage, only available on Alex
  - https://doc.nhr.fau.de/data/workspaces/

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Workflows: job arrays and dependencies

Grouping work together: array jobs

- Many jobs that only differ by some index → Array jobs
  - Jobs are differentiable by **`$SLURM_ARRAY_TASK_ID`**
  - Submit with **`#SBATCH --array=1-10`**
  - Default job ID is then **`$SLURM_JOBID_$SLURM_ARRAY_TASK_ID`**

```
#!/bin/bash -l
#SBATCH --job-name=array_f
#SBATCH --nodes=1
#SBATCH --time=1:00:00
#SBATCH --array=1-5
#SBATCH --export=NONE
unset SLURM_EXPORT_ENV
echo "I am job $SLURM_JOBID, index $SLURM_ARRAY_TASK_ID"
# use task ID in program arguments
./a.out 500 300 $SLURM_ARRAY_TASK_ID
```

```
$ squeue
     JOBID PARTITION NAME                 USER     ST       TI
 1714146_2 singlenod array_f              unrz55    R       0:
 1714146_3 singlenod array_f              unrz55    R       0:
 1714146_4 singlenod array_f              unrz55    R       0:
 1714146_5 singlenod array_f              unrz55    R       0:
 1714146_1 singlenod array_f              unrz55    R       0:
```

Job dependencies

- Can be useful for long-running sequences of jobs.
- Jobs will be set on hold until specified dependencies are satisfied.

```
#SBATCH -d <type>:<jobID>[:<jobID>]
```

Available types:

- **`after`**: job can begin execution after the specified jobs have begun execution.
- **`afterany`**: job can begin execution after the specified jobs have terminated.
- **`afternotok`**: job can begin execution after the specified jobs have terminated in some failed state (non-zero exit code, node failure, timed out, etc).
- **`afterok`**: job can begin execution after the specified jobs have successfully finished (zero exit code).
- **`singleton`**: job can begin execution after any previously launched jobs sharing the same job name and user have terminated.

Job dependencies: example

Script-generated dependency chain:

```
$ ID=`sbatch test_dep.sh | grep "^Submitted" | cut -f 4 -d ' '`
$ for i in `seq 1 6`; do \
    ID=`sbatch -d afterok:$ID test_dep.sh | grep "^Submitted" | cut -f 4 -d ' '`; \
  done
$ squeue
     JOBID PARTITION […SNIP…] TIME  TIME_LIMIT  NODES CPUS NODELIST(REASON)
   1715105 singlenod […SNIP…] 0:00     1:00:00      1    1 (Dependency)
   1715104 singlenod […SNIP…] 0:00     1:00:00      1    1 (Dependency)
   1715103 singlenod […SNIP…] 0:00     1:00:00      1    1 (Dependency)
   1715102 singlenod […SNIP…] 0:00     1:00:00      1    1 (Dependency)
   1715101 singlenod […SNIP…] 0:00     1:00:00      1    1 (Dependency)
   1715100 singlenod […SNIP…] 0:00     1:00:00      1    1 (Dependency)
   1715099 singlenod […SNIP…] 0:05     1:00:00      1   72 f0458
```

Chain jobs

Auto-submit next job from the job script:

job_script.sh

```
#!/bin/bash -l

# ... do the work here ...
./a.out

# submit next job in chain
if [ $SECONDS -gt 3600 \
  -a ! -e ${SLURM_SUBMIT_DIR}/STOP_CHAIN ]; then
  cd $SLURM_SUBMIT_DIR
  sbatch job_script.sh
fi
```

Make sure that current shell has been running reasonably long

"touch STOP_CHAIN" breaks the chain

Chain jobs with checkpointing

If the program writes a checkpoint, resubmit if checkpoint exists

job_script.sh

```
#!/bin/bash -l

CKPT=$FASTTMP/ckpt.dat
# ... do the work here, ckpt in $FASTTMP ...
./a.out --checkpoint $CKPT

# submit next job in chain
if [ $SECONDS -gt 3600 \
  -a -s $CKPT_DIR/ckpt.dat \
  -a ! -e ${SLURM_SUBMIT_DIR}/STOP_CHAIN ]; then
  cd ${SLURM_SUBMIT_DIR}
  sbatch job_script.sh
fi
```

File exists and has nonzero size

A Slurm-managed serial job queue

Use case: I have a bag of serial tasks and I want to run them on a number of nodes; new tasks should start as soon as core/memory become available

```
#!/bin/bash -l
#SBATCH --nodes=2
#SBATCH --cpus-per-task=1
#SBATCH --export=NONE

unset SLURM_EXPORT_ENV

for i in `seq 1 2000`; do
  srun -N 1 -n 1 --mem-per-cpu=2G --exact ./a.out &
done

wait
```

Default: full node memory

Do not oversubscribe memory per node

Put them all into a big queue

Wait for all background job steps to complete

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

THANK YOU.

NHR@FAU

https://hpc.fau.de

Official docs: **https://doc.nhr.fau.de**

NHR@FAU video channel on FAU.tv:

https://www.fau.tv/course/id/1146