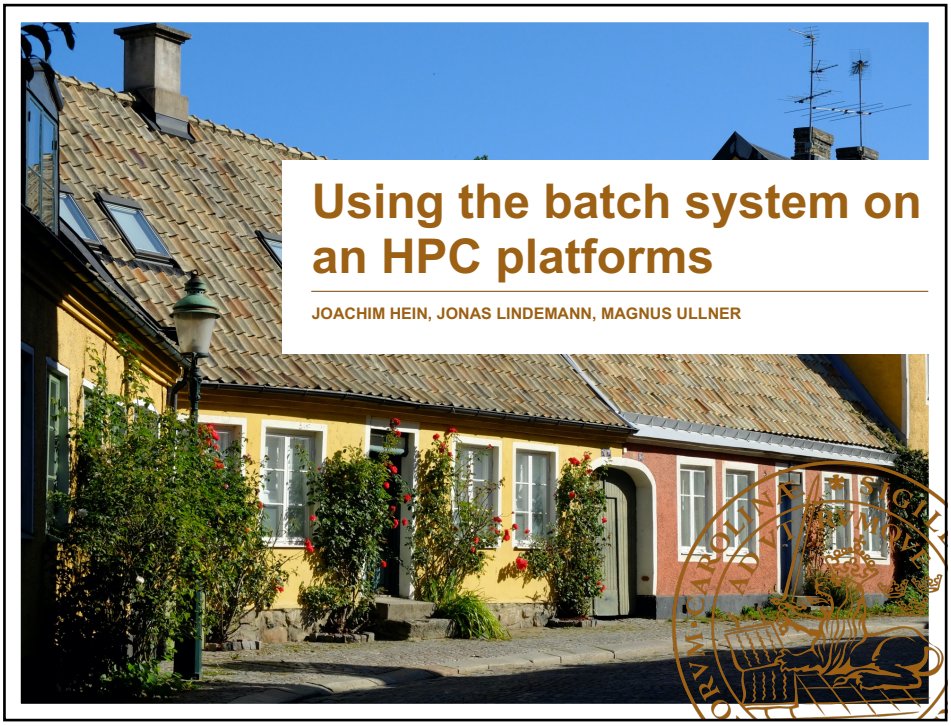


Using the batch system on an HPC platforms

JOACHIM HEIN, JONAS LINDEMANN, MAGNUS ULLNER

1

Outline

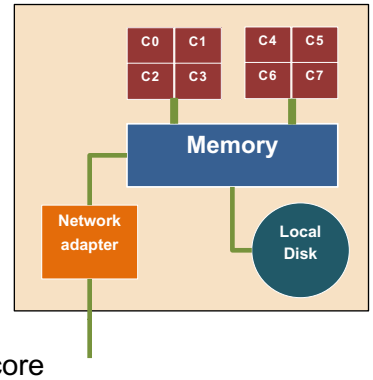
- Batch system concepts
- Using the batch systems deployed on an HPC system
  - SLURM (this lecture)
- Examples
  - Parallel jobs
  - Task farm

LUND UNIVERSITY

2

Schematic view: Compute node

- Multiprocessor
- Multicores
- Shared memory
- Node-local disk
  - Fast access
- Network adapter



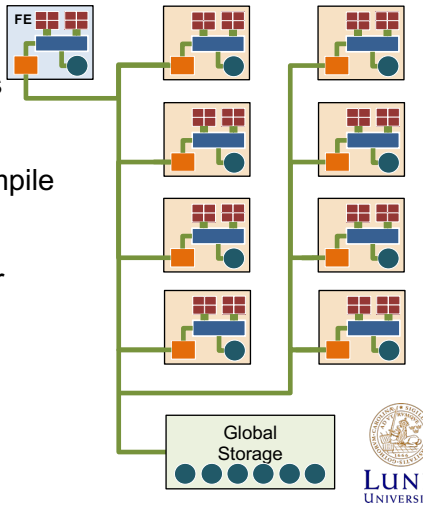
- Operation: Single task/core
  - Multiple serial jobs
  - Parallel multicore job(s)

LUND UNIVERSITY

3

Schematic view: HPC Cluster

- Many compute nodes
  - Resource intensive jobs
- Frontend node(s)
  - Login, Submission, Compile
- Global storage disks
  - Shared for entire cluster
  - Visible from all nodes
- Node-local disks
  - Best performance
  - Visible only inside node



LUND UNIVERSITY

4

Concept: Scheduling jobs

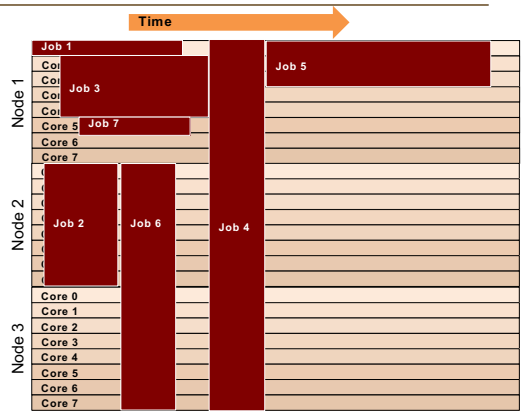
- Job 1: 1 core, 8 h
- Job 2: 8 cores, 4 h
- Job 3: 4 cores, 8 h
- Job 4: 24 cores, 3 h

  wait for Job 3
- Job 5: 3 cores, 12 h

  not enough time before Job 4
- Job 6: 16 cores, 2 h

  run after 2, before 4
- Job 7: 1 core, 6 h

  run directly



LUND UNIVERSITY

5

Remarks on start times

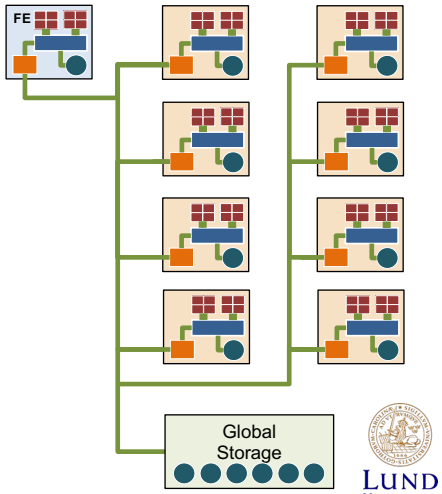
- Animation is a bit simplistic
- Scheduling typically not static
- Job start might move forward or backward in time
- You are typically not there when the job starts
- Provide system with a job description: **job script**

LUND UNIVERSITY

6

Basic purpose of a jobscript

```
#!/bin/bash

#resource: time, cores, …

prepare the job (e.g. input)

run the program

post process (e.g. copy results)
```



LUND UNIVERSITY

7

SLURM

- LUNARC: COSMOS
- HPC2N: Kebnekaise
- NSC: Tetralith
- PDC: Dardel

SLURM

- Widely used on HPC clusters world wide
- Always check **local documentation**

LUND UNIVERSITY

8

10

A simple job script: Three parts! SLURM example

```
#!/bin/bash
#SBATCH -t 00:05:00
#SBATCH –A account

echo "hello"
```

1. specify UNIX shell
2. resource statements

3. UNIX script

Write jobscript into a file
Submit to the job queue

LUND UNIVERSITY

11

Comment on walltime

- ***The*** most important resource statement
- Points to consider
  - Job **terminated** after specified time
  - Charging based on consumed time
  - Runtime variations for successive runs
- Over specify your requirements within reason
  - You want hanging jobs terminated
  - Shorter jobs can start quicker (backfill)

LUND UNIVERSITY

12

LUND UNIVERSITY

Interacting with the queue

SUBMISSION, QUEUE MONITORING AND MODIFICATION

13

SLURM: Submission with `sbatch`

- Use sbatch to submit your job script to the job-queue
- Example:

```
[fred@aurora Timetest]$ sbatch runjob.sh
Submitted batch job 7197
```

- Submit script "`runjob.sh`"
- Successful submission returns a job-id number
- **Best practice**: load modules inside script

14

SLURM: Monitoring the queue with `squeue`

- Use **squeue** to monitor the job queue

```
JOBID  PARTITION    NAME   USER  ST   TIME  NODES NODELIST(REASON)
  7303   snic   hybrid_n   fred  PD   0:00     32 (Priority)
  7302   snic   hybrid_n   fred  PD   0:00     32 (Priority)
  7301   snic   hybrid_n   fred  PD   0:00     32 (Resources)
  7304   snic   preproce   karl  PD   0:00      6 (Priority)
  7300   snic   hybrid_n   fred   R   0:24     32 au[001-032]
  7305   snic   preproce   karl   R   0:37      6 au[081-086]
  7306   snic   hybrid_n   fred   R   0:37      6 au[081-086]
  7307   snic   testsimu   sven   R   0:07      1 au081
```

- Typically lots of output – use options of `squeue` to filter

15

SLURM: Options of `squeue`

- Showing jobs for a specific user

```
squeue -u fred
```

will show the jobs of user "`fred`" only

- Option `--start` gives the estimated job start time
  - Estimate can shift in either direction

16

SLURM: Deleting jobs with `scancel`

- You can cancel a queued or running job
- Determine job-id, e.g. with `squeue`
- Use `scancel`

```
scancel 7103
```

  - terminates job 7103, if running
  - removes from the queue

17

SLURM: Dependencies

- Workflows often require subsequent jobs
  - E.g. serial preprocessing, followed by parallel simulation and serial post processing
- Submit first job:

```
[fred@aurora Simcode]$ sbatch run_mesh.sh
Submitted batch job 8042
```

- Use returned job-id in submission of second job

```
[fred@aurora Simcode]$ sbatch -d afterok:8042 run_sim.sh
Submitted batch job 8043
```

- Continue with next job, consider scripting (e.g. python)

LUND UNIVERSITY

18

19

Example resource header: Basic job script - SLURM

```
#!/bin/bash
#
# you can comment
#SBATCH -t 04:30:00
#
#SBATCH -J data_process
#
#SBATCH -o process_%j.out
#SBATCH -e process_%j.err
```

- Use as many "#SBATCH" as needed
- Use "blank" line to structure
- Use comments
- `-t` time, here 4h and 30 min
- `-J` job-name
- `-o` output file, `%j` gives job-id
- `-e` error file, `%j` gives job-id
- **Followed by the script part**

20

SLURM Email about your job

- Specify mail address

```
#SBATCH --mail-user=name@institute.uni.se
```

- Specify mail occassion

```
#SBATCH --mail-type=END
```

- Valid types include (multiple possible – comma separate):
  - BEGIN
  - END
  - FAIL
  - TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80

21

Get your jobscript into the output

- It is often useful to get your jobscript into the job output
  - E.g. when developing scripts
- Add the following as the first line of the script portion (after the last **`#SBATCH`** line)

```
cat $0
```

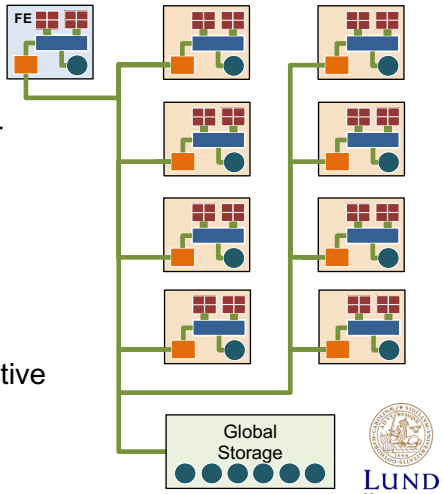
LUND UNIVERSITY

22

23

Reminder: Schematic view: HPC Cluster

- Many compute nodes
- Global storage disks
  - Shared on entire cluster
  - Visible from all nodes
- Node-local disks
  - Best performance
  - Visible only inside node
  - Available while job is active



LUND UNIVERSITY

24

Node local disks

- Often nodes (e.g. Tetralith, COSMOS) have a local disk
  - Best performance (access time, bandwidth)
  - Shared only between the cores of the node
  - Shared between all users of the node
  - Multi-node jobs: each node has **different** local disk
- Need to:
  - Copy input files onto local disk(s) before program start
  - Start your program from local disk
  - Copy results from local disk after program finish
- Use job script to do so

LUND UNIVERSITY

25

Examples for sizes of node local disk

System	Local disk
COSMOS standard node (LUNARC)	1.6 TB SSD
Tetralith standard node (NSC)	240 GB SSD
Kebnekaise standard node (HPC2N)	170 GB SSD

26

Useful UNIX variables to access local disks

Variable name	Addressed Volume
`SNIC_TMP`	**node-local disk** copy your input data here and start your program from here
`SLURM_SUBMIT_DIR`	submission directory in **SLURM** Directory where you ran `sbatch`

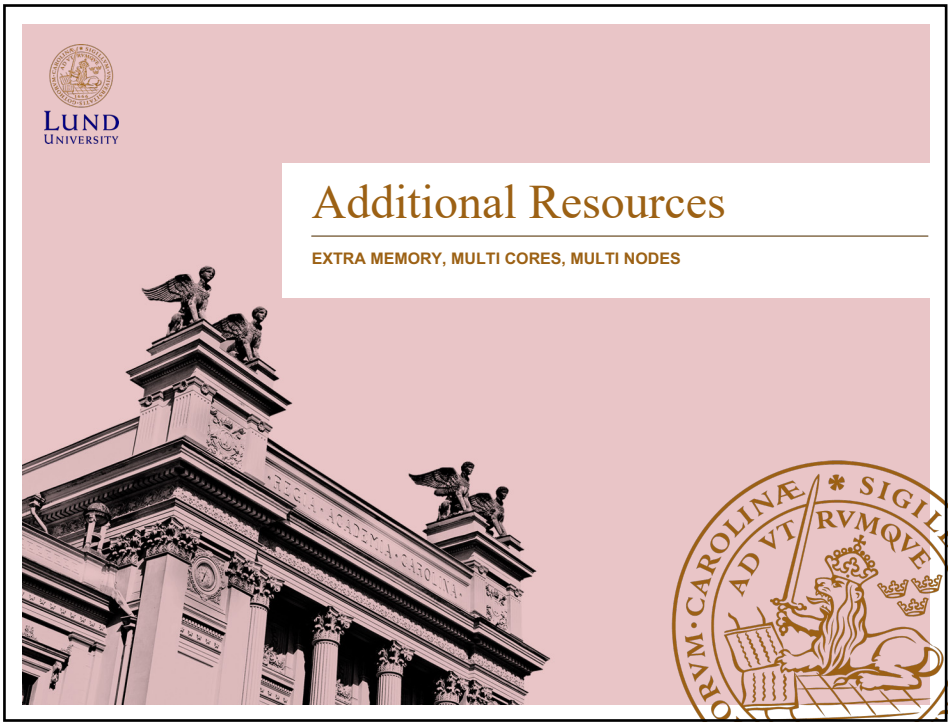
- On some systems $TMPDIR points to the node-local disk as well
  - Check your system if you need this
- Check documentation of ”your” system

27

UNIX script part: Basic job script using local disk

```
# copy the input data and program to local disk
cp -p input.dat  $SNIC_TMP
cp -p my_program $SNIC_TMP

# change to the execution directory
cd $SNIC_TMP

# run the program
./my_program

# rescue the results to the submission directory
cp -p result.dat $SLURM_SUBMIT_DIR
```

28

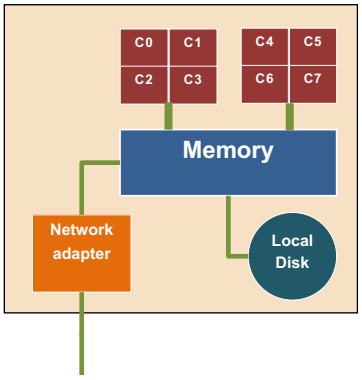


29

Memory: A shared resource

COSMOS:

- 48 cores/node
- 256 GB Memory
  - 254000 MB available to users
  - Default: 5300 MB/core
- 1.6 TB local SSD disk space



LUND UNIVERSITY

30

Memory requirements on COSMOS

- Nodes have 256 GB and 48 cores
- Default memory request: **5300 MB per core**
- If you need more memory
  - Specify your requirements

```
#SBATCH --mem-per-cpu=10600
```

    This asks for 10600 MB per core
  - Results into some cores without memory
    - » Your account gets charged for those
    - » Consider using shared memory parallelism to get (some) use of these idle cores

LUND UNIVERSITY

31

Memory requirement on Kebnekaise

- Physical memory per core:
  - 4450 MB Broadwell node
  - 6750 MB Skylake node
  - 41666 MB Large memory node
- Default setting 4450 MB in standard CPU partition
- If you ask more cores (e.g. -c 2) you get twice the default

```
#SBATCH -c 2
```

- If you project has access to the large memory nodes:

```
#SBATCH -p largemem
```

33

Memory requirement on Tetralith

- Physical memory per node:
  - 96 GiB per 32 core standard node
  - 96 GiB per GPU node
  - 384 GiB Large memory node
- Default setting 2904 MB per core for standard node
- To use a large memory node add (11616 MB/core)

```
#SBATCH -C fat
```

34

SLURM
Number of cores and number of nodes

- Typically the first two suffice (our recommendation):
- Number of nodes

```
#SBATCH -N 4
```

- Number of tasks per node (max value depends)

```
#SBATCH --tasks-per-node=48
```

- This example will give (and charge) you for 192 cores
  - Use 28 cores per node on standard Kebnekaise node
  - Use 32 cores per node on standard Tetralith nodes
- Aim to use complete nodes

LUND UNIVERSITY

35

COSMOS, Tetralith:
Interactive Jobs

- No **CPU** or **memory intensive** jobs on frontend (login machine)
- Run interactive jobs on compute nodes via SLURM

```
interactive --tasks-per-node=4 -N 1 -t 60
```

  Example asks 4 cores on 1 node for 60 minutes
- Not a SLURM feature – many Swedish centres offer this
- Job waits in the queue for resources
- Shell starts in the invocation directory
  - **Important**: purge your modules and reload
- Primarily intended for:

  tests, debug, interactive analyses and heavy compilation
- For production runs, use job scripts (batch jobs)

LUND UNIVERSITY

37

43

Things to consider when utilising shared memory

- Application can be parallelised using shared memory techniques (e.g. Posix, OpenMP, TBB, …)
- Linking a threaded library results in your code being (partly) parallel, e.g. FFTW, threaded MKL, OpenBLAS
- Start the code like a serial program
- Control number of cores/threads via SLURM
  - On COSMOS you can use up to 48 cores/threads
  - On Kebnekaise you can use up to 28 cores/threads
  - On Tetralith you can use up to 32 cores/thread
- In a hybrid situation (MPI + threads) you need to control the thread count using environment variable OMP_NUM_THREADS

LUND UNIVERSITY

46

Things to consider for MPI jobs

- Today most MPI programs are executed as single program multiple data (SPMD)
- MPI programs need a job launcher
  - This takes the actual executable as argument
  - Connects to all the cores
  - Starts a copy of the executable on every core

49

Examples for job launchers

- Job launcher on COSMOS:
  - Use `srun` when using the Intel MPI library
  - Use `mpirun` or `mpiexec` when using OpenMPI
- Job launcher on Tetralith
  - Use `mpprun`
- Job launcher on Kebnekaise
  - Use `srun`

50

SLURM standard MPI script: Resource header portion (COSMOS)

```
#!/bin/bash
# requesting the 4 nodes 48 cores each (192 cores total)
#SBATCH -N 4
#SBATCH --tasks-per-node=48
#
#SBATCH -t 0:30:00
#
#SBATCH -J simula_n192
#
#SBATCH -o simula_n192_%j.out
#SBATCH -e simula_n192_%j.err
```

LUND UNIVERSITY

52

SLURM standard MPI script (cont.): UNIX body (*OpenMPI, COSMOS*)

```
cat $0

module purge
module load foss/2022b


# starting the executable
# foss/2022b deploys OpenMPI – start executable with mpirun
# --bind-to core is recommended for standard MPI jobs

mpirun --bind-to core simula_mpi
```

UNIVERSITY

53

SLURM standard MPI script (cont.): UNIX body (*Intel MPI, COSMOS*)

```
module purge
module load intel/2022a


# starting the executable
# intel/2022a deploys Intel MPI – start executable with srun
# --cpu_bind=cores is recommended for standard MPI jobs

srun --cpu_bind=cores simula_mpi
```

UNIVERSITY

54

SLURM standard MPI script: (Tetralith)

```
#!/bin/bash
# requesting the 4 nodes 32 cores each (128 cores total)
#SBATCH -N 4
#SBATCH --tasks-per-node=32
#
#SBATCH -t 0:30:00
#
#SBATCH -J simula_n128
#
#SBATCH -o simula_n128_%j.out
#SBATCH -e simula_n128_%j.err

cat $0
mpprun simula_mpi
```

UNIVERSITY

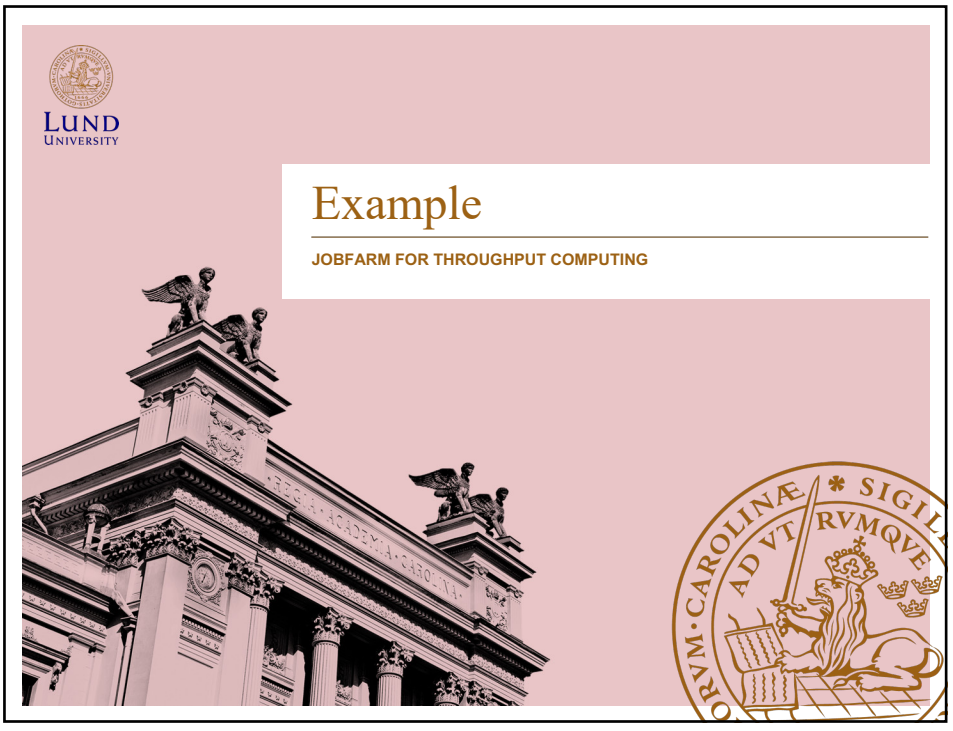
55

SLURM standard OpenMP script shared mem, non-IO intensive, COSMOS

```
#!/bin/bash
#SBATCH --tasks-per-node=48      # 28 or 32 on K-kaise/Tetralith
#SBATCH -N 1                             # this is crucial
#SBATCH -t 08:00:00
#SBATCH -J data_process
#SBATCH -o process_omp_%j.out
#SBATCH -e process_omp_%j.err


export OMP_PROC_BIND=true                # use binding
./processor_omp                          # run the program
```

LUND UNIVERSITY

57



59

Jobfarms: queue inside job

- Standard problem: running a large number of serial jobs
  - often same program but different input files
  - execution time can vary depending on the input file
- Idea: Jobfarm
  - Take a number of cores
  - Take an even larger number of jobs (up to about 1000)
  - Script register the jobs with SLURM as a “mini queue”
  - SLURM runs them 1 by 1 until “all done”
    - » Each core has 1 job at a time
    - » Get new job once done
  - Gives natural load balance
- Not subject to job limit restrictions (e.g. XXX jobs /user)
- Check LUNARC documentation for details

60

Jobfarms: SLURM array job

- Problem: running a large number of serial/parallel jobs
- Idea: SLURM array job
  - Easier to set up (less UNIX scripting expertise)
  - Submit all jobs in single array script
    - » Submits large number of almost independent jobs
  - Work (e.g. input file) is controlled by an ***array index***
  - Specify the values the index can take
    - » Range
    - » Range with increment
    - » List

61

Discussion

- Queue inside job
  - Test whether or not it works on multiple nodes
  - Hides many serial jobs inside a single SLURM job
    - » Can process 1000 ser. jobs of modest runtime
  - Requires more UNIX/SLURM expertise to develop
- Array job
  - Easy to set up
  - Easy to use with any other SLURM feature
    - » Parallel, Queue inside job, ...
  - Subject to queueing limits (e.g. max job-counts)
- Both give job all the memory of the core

62

Preparation for array job

- Assume we want to run 200 jobs
- Prepare 200 directories: `job_0, job_1, …, job_199`
  - Contains: program executable(s), input files, …
- Maximum number of jobs restricted by number of jobs a user can have

63

Specifying array jobs

- Specify array job and index as a range from 12 to 45

```
#SBATCH --array=12-45
```

- Specify array job and index from 12 to 24 in steps of 4

```
#SBATCH --array=12-24:4
```

- Specify array job as a list

```
#SBATCH --array=3,6,7,15
```

- Adapt name of output/error file, e.g.

```
#SBATCH -o process_%A_%a.out
```

- Query the array index with the UNIX variable

```
$SLURM_ARRAY_TASK_ID
```

64

Header of array job

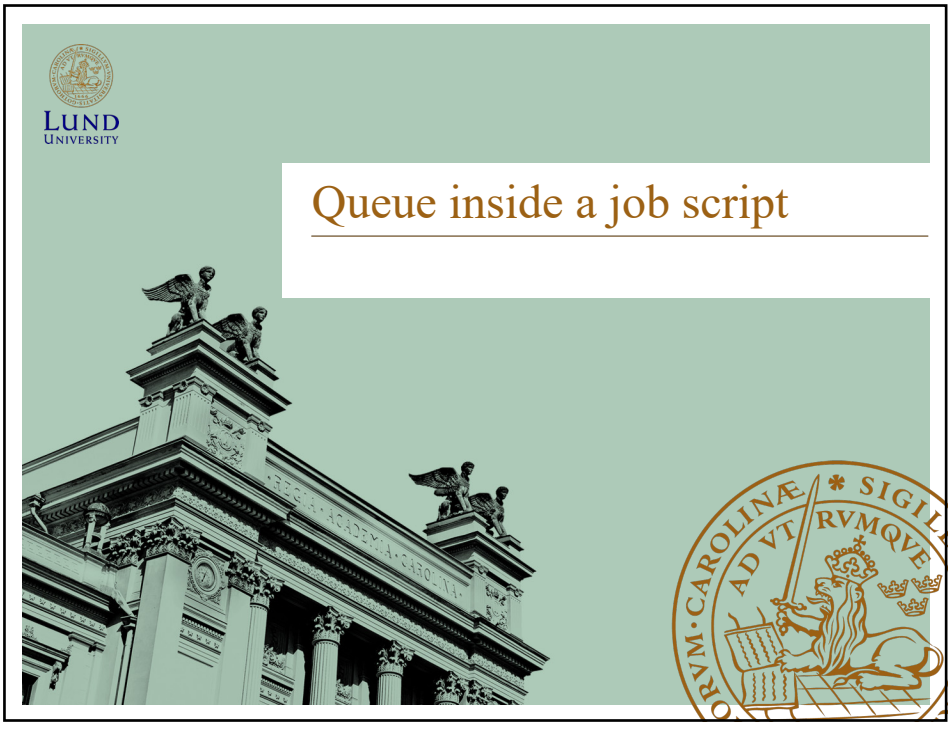
```
#!/bin/bash
#SBATCH -t 04:00:00
#SBATCH -A lu-test
#SBATCH -J jobFarm
#SBATCH --array=0-199                # Array index
#SBATCH -o res_array_Farm_%A_%a.out
#SBATCH -e res_array_Farm_%A_%a.out
```

65

UNIX body of array job

```
export WRK_NB=$SLURM_ARRAY_TASK_ID       # worker id from task id

export WRK_DIR=$SNIC_TMP/WRK_${WRK_NB}
mkdir  $WRK_DIR                          # private working dir

                                         # copy inputs
export COM_DIR=$SLURM_SUBMIT_DIR/CommonFiles
export JOB_DIR=$SLURM_SUBMIT_DIR/job_${WRK_NB}

cp -p $COM_DIR/commonInput.dat $WRK_DIR
cp -p $JOB_DIR/input.dat $JOB_DIR/processor $WRK_DIR

cd $WRK_DIR
time ./processor                          # run the program

cp -p result.dat ${JOB_DIR}
cd $SNIC_TMP
rm -rf WRK_${WRK_NB}
```

66



67

Master script

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --tasks-per-node=48
#SBATCH -t 20:00:00
#SBATCH -J jobFarm
#SBATCH -o res_jobFarm_%j.out
#SBATCH -e res_jobFarm_%j.out

export NB_of_jobs=192    # set the number of jobs - change
for ((i=0; i<$NB_of_jobs; i++))
do
 srun -Q --exclusive -n 1 -N 1 \
   workScript.sh $i &> worker_${SLURM_JOB_ID}_${i} &
 sleep 1  # this is crucial for stability
done
wait  # keep the wait statement, it is important!
```

68

Worker script: Outline

- Modification of a basic serial job script
- Job-private sub-directory on `$SNIC_TMP` to avoid conflicts
- Input file(s) expected in `job_0, job_1, …`
  - Subdirectories of `$SLURM_SUBMIT_DIR`
- Example assumes single input and output file
  - Modify for multiple files
- Current set up allows for different executable for each job
  - Modify/simplify if same executable for all jobs
- After saving of output remove the private dir from `$SNIC_TMP`

69

Example for a worker script put in file: `workScript.sh`

```
#!/bin/sh
export WRK_NB=$1                  # receive my worker number

export WRK_DIR=$SNIC_TMP/WRK_${WRK_NB}
mkdir $WRK_DIR                    # private working directory

export JOB_DIR=$SLURM_SUBMIT_DIR/job_${WRK_NB}
cd $JOB_DIR
cp -p input.dat processor $WRK_DIR

cd $WRK_DIR
./processor                        # run the program

cp -p result.dat ${JOB_DIR}
cd $SNIC_TMP
rm -rf WRK_${WRK_NB}
```

70

Modification for common files

If you have input files and/or executables common to all jobs:

- Place in a common subdirectory on `$SNIC_TMP`
- Multinode jobs: Each node needs the common sub-dirs
  - Use `srun` to copy onto the nodes

71

Monitoring your task farm

- Option `-s` of `squeue` allows monitoring of your taskfarm
- Specify the job number with the `-j` option

```
[fred@aurora MultiSerialTest]$ squeue -j 8070 -s
STEPID         NAME PARTITION     USER     TIME NODELIST
8070.130   small_ex      snic     fred     2:09 an074
8070.133   small_ex      snic     fred     2:02 an073
8070.135   small_ex      snic     fred     1:55 an074
8070.136   small_ex      snic     fred     1:41 an073
8070.139   small_ex      snic     fred     1:41 an073
8070.140   small_ex      snic     fred     1:41 an073
8070.143   small_ex      snic     fred     1:41 an073
8070.144   small_ex      snic     fred     1:41 an074
```

LUND UNIVERSITY

72

Documentation and advanced options

- SLURM on COSMOS
  - https://lunarc-documentation.readthedocs.io/en/latest/manual/manual_intro/
- SLURM on Tetralith
  - https://www.nsc.liu.se/support/batch-jobs/introduction/
- SLURM on Kebnekaise
  - https://www.hpc2n.umu.se/documentation/guides/beginner-guide
  - https://www.hpc2n.umu.se/documentation/guides/using_kebnekaise

73

Summary

- Job schedulers are required for HPC installations
- They balance the needs of various users
- Specify the resources you need
- Specify the work that needs doing
- They offer a great deal of flexibility

75

LUND
UNIVERSITY

76