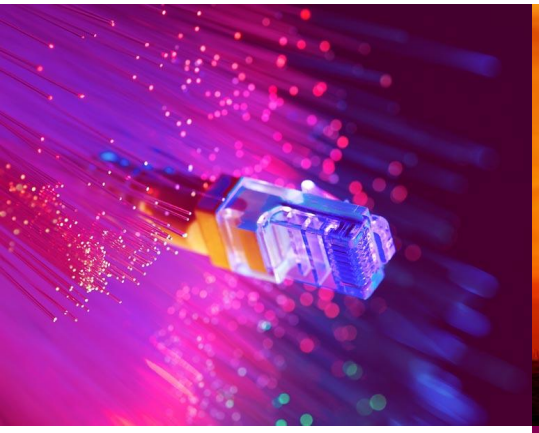
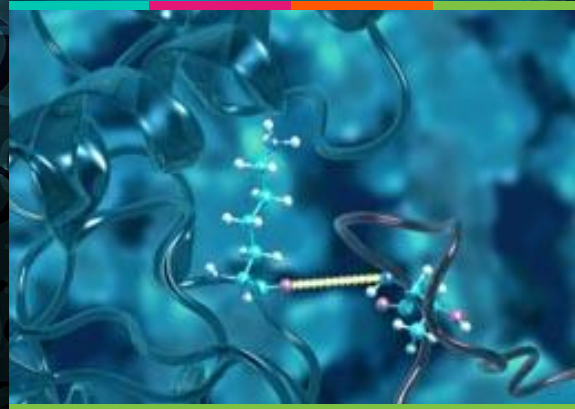


Bioweek: Using Modern HPC Environment Efficiently

2020-02-03

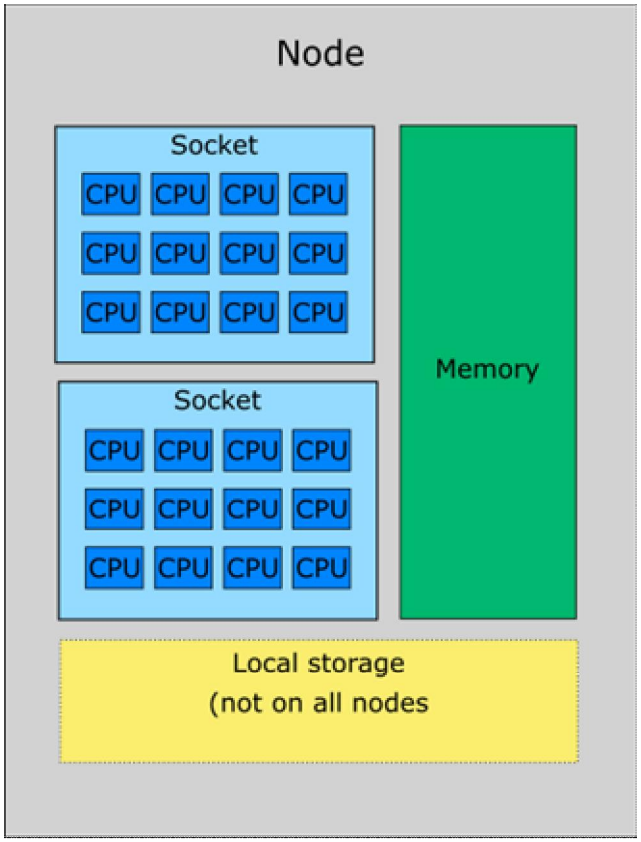

CSC – Suomalainen tutkimuksen, koulutuksen, kulttuurin ja julkishallinnon ICT-osaamiskeskus

Brief introduction to HPC Environments



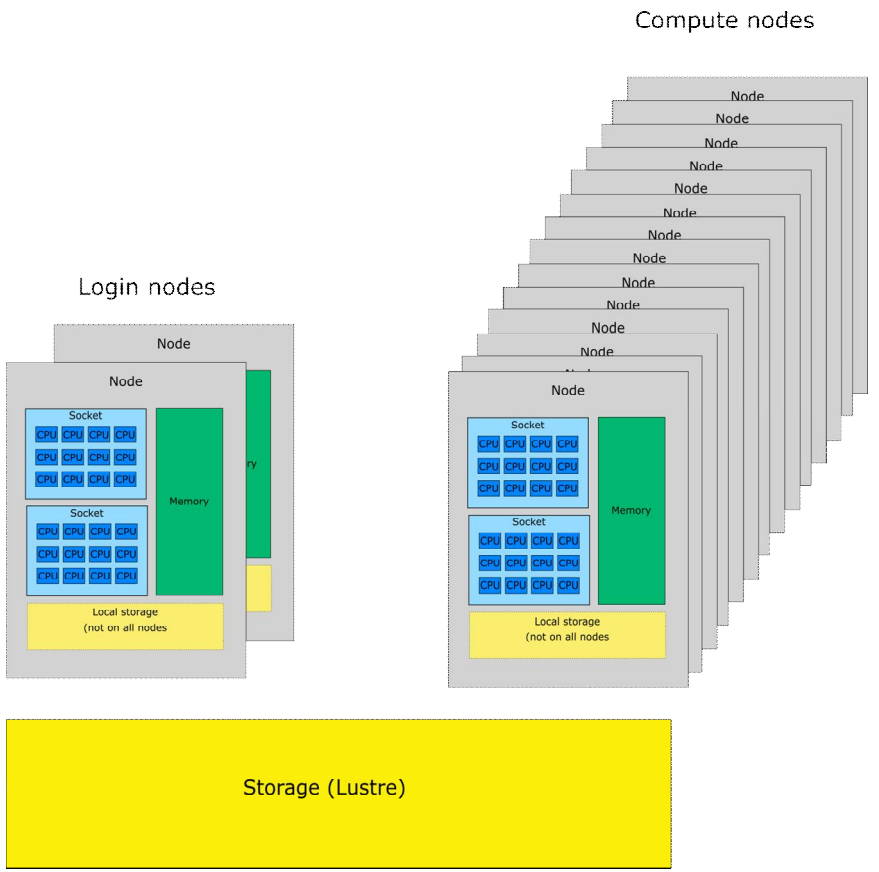
Some notes on vocabulary

computer ~= node
processor ~= socket
core ~= CPU



Cluster systems

- Login nodes are used to set up the jobs
- Jobs are run in the compute nodes
- A batch job system (aka scheluder) is used to run and manage the jobs
  - On this course we use Slurm
  - Other common systems include SGE and Torque/PBS
  - Syntax is different, but basic operation is similar



- To be able to plan your jobs efficiently, you need to familiarize yourself with the available resources
- Each system is different, so check the documentation
- Things to check
  - What batch job system is used
  - What kind of nodes are available?
    - Number of cores
    - Size of memory
    - Extra hardware, e.g GPU, fast local storage
  - What partitions (queues) are available
    - Job sizes, max length, etcc
  - Provisioning policy
    - Per core/per node/other

Puhti nodes

	Type	CPU	CPU cores	Memory	Number of nodes
Puhti CPU parition	M	Xeon Gold 6230	2 x 20 cores @ 2.1 GHz	192 GB	532
	L	Xeon Gold 6230	2 x 20 cores @ 2.1 GHz	384 GB	92
	IO	Xeon Gold 6230	2 x 20 cores @ 2.1 GHz	384 GB + 3.2 TB NVMe	40
	XL	Xeon Gold 6230	2 x 20 cores @ 2.1 GHz	768 GB	12
	BM	Xeon Gold 6230	2 x 20 cores @ 2.1 GHz	1.5TB	6
Puhti-AI GPU partition	GPU	Xeon Gold 6230 4 x V100 32 GB	2 x 20 cores @ 2.1 GHz	384 GB (Host) 128 GB (GPUs) 3.2 TB NVMe	80

Puhti CPU partitions

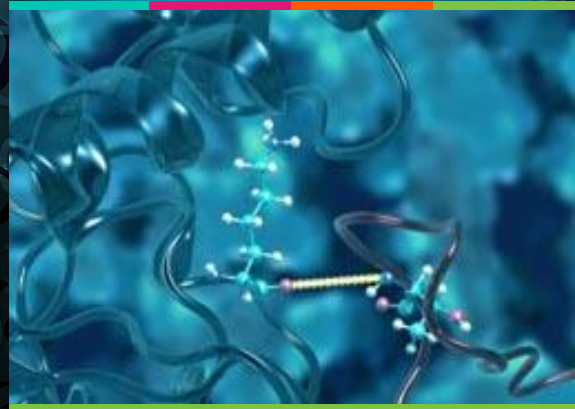
Partition	Time limit	Max tasks	Max nodes	Node types	Max memory	Max local storage
test	15 min	80	2	M	382 GB	
small	3 days	40	1	M, L, IO	382 GB	3600 GB
large	3 days	4000	100	M, L, IO	382 GB	3600 GB
longrun	14 days	40	1	M, L, IO	382 GB	3600 GB
hugemem	3 days	160	4	XL, BM	1534 GB	
hugemem_longrun	14 days	40	1	XL, BM	1534 GB	

Puhti GPU partitions

Partition	Time limit	Max GPUs	Max nodes	Node types	Max memory	Max local storage
gputest	15 min	8	2	GPU	382 GB	3600 GB
gpu	3 days	80	20	GPU	382 GB	3600 GB

Note that for each GPU, you should reserve at most 10 cores/task.

Software environment



Module system

- Different software packages have different, possibly conflicting, requirements.
- CSC uses LMOD module system to manage software and programming environments
- `module load biokit`
  - sets up most of the bioinformatics tools (but not all the tools)
- See software web pages for details
  - https://docs.csc.fi/apps/

Most commonly used module commands:

`module help`	Shows available options
`module load` *`modulename`*	Loads the default version of the module
`module load` *`modulename/version`*	*Loads specific version of the module*
`module list`	List the loaded modules
`module avail`	List modules that are available to be loaded (i.e. compatible with your current environment)
`module spider`	List all existing modules
`module spider` *`name`*	Searches the entire list of existing modules
`module spider` *`modulename/version`*	Gives information on how to load the module (prerequisites etc)
`module swap` *`module1 module2`*	Replaces a module with a another module and tries to re-load compatible versions of other loaded modules
`module unload` *`modulename`*	Unloads the given environment module
`module purge`	Unloads all modules

Advanced module commands:

- These can be helpful with your own software installations

`module save filename`	Saves current module set
`module restore filename`	Loads saved module set

- It is also possible to write your own modulefiles. For example, if you add the module files to $HOME/modulefiles, you can access them after you add the path to the modules search path using command:

```
module use $HOME/modulefiles
```

Setup scripts

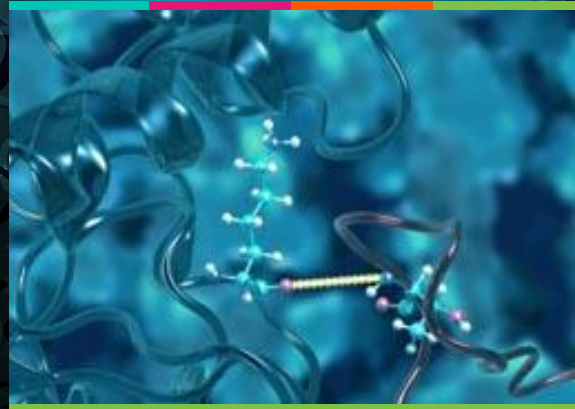
- Instead of personal module files, setup scripts can be handy
  - Easy and fast to write
  - Changes to .bashrc not recommended: Permanent changes to $PATH, $LD_LIBRARY_PATH, $PYTHONPATH, $PERL5LIB etc can lead to incompatibility issues later on
- Example:

```
module load module1
module load module2
export PATH=/my/install/bin:${PATH}
export LD_LIBRARY_PATH=/my/install/lib:${LD_LIBRARY_PATH}
```

- Run with:

```
source my_prog.setup
```

Writing batch job scripts



Types of jobs

- Serial jobs
  - o Use only one core
  - o Many bioinformatics tools

- Embarrassingly parallel tasks:
  - o Job can be split to numerous sub jobs
  - o Array jobs
  - o Some tools do it automatically, e.g. pb blast, cluster_interproscan, trinity, miso

Types of jobs

- Threads/ shared memory/OpenMP based parallelization
  - All the parallel processes must see the same memory -> all processes must run within one node
  - Usually best to match number of threads to number of cores
  - See software documentation for details
  - Many bioinformatics tools use this approach. Bowtie2, BWA, Tophat...

Types of jobs

- MPI parallelization.
  - Each task has own memory -> job can utilize several nodes
  - Check scaling before launching big jobs
  - Not all analyses can be split this way, and only few bioscience applications use MPI

Parallel jobs

- Only applicable if your program supports parallel running
  - If the software documentation makes no mention of number of cores/processors/processes/threads to use, it's probably a serial code
- Check application documentation on number of cores to use
  - Speed-up is often not linear
  - Maximum effective number of cores can be limited by the algorithms and data
  - Using too many cores can actually make your job run slower
- Remember to set software parameters to match your batch job script
  - Some software will use only one core by default, even if more allocated
  - Some software will try to use all the cores in the node by default, even if less cores allocated

Interactive vs. Batch jobs

- Typical interactive jobs
  - Short jobs
  - Serial jobs (or small shared memory parallel jobs)
  - Software with GUI

- Typical batch jobs
  - Long jobs
  - Parallel jobs
  - Jobs that need specific resources (e.g. hugemem nodes, GPU nodes etc.)

Interactive jobs in Puhti

- Only very small tasks should be done on the login nodes
  - You can run GUI:s to set up jobs, to manage program options etc, but actual jobs should be run through the batch job system
- Currently no interactive nodes in Puhti (like Taito-shell)
  - They are planned, but timetable is still open
- You can request resources for an interactive job through the batch job system

```
srun --account <project> --gres=nvme:100 --time 01:00:00 --mem=16G --pty $SHELL
```

Steps for running a Batch job

- Write a batch job script
  - Script format depends on server, check the user guides, e.g:
    - https://docs.csc.fi/#computing/running/creating-job-scripts
- Make sure you have all the necessary input files where the program can find them
  - Usually best to use scratch
  - $HOME and projappl have limited space
- Submit your job
  - `sbatch myscript`

Batch jobs

- User has to specify necessary resources
  - Can be added to the batch job script or given as command line options for `sbatch` (or a combination of script and command line options)
- Resources need to be adequate for the job
  - Too small memory reservation will cause the job to fail
  - When the time reservation ends, the job will be terminated, whether finished or not
- But: Requested resources can affect the time the job spends in the queue
  - Especially core number and memory reservation
- So: Realistic resource request give best results
  - Not always easy to know beforehand
  - Usually best to try with smaller tasks first and check the used resources

Example serial batch job script on Puhti (Slurm):

```
#!/bin/bash
#SBATCH --job-name=bowtie2
#SBATCH --account=<project>
#SBATCH --output=output_%j.txt
#SBATCH --error=errors_%j.txt
#SBATCH --time=02:00:00
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6G
#SBATCH --partition=small
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK chr_18 reads.fq > out.sam
```

`#!/bin/bash`

- Tells the computer this is a script that should be run using bash shell
- Everything starting with ”#SBATCH” is passed on to the batch job system
- Everything starting with ”# ” is considered a comment
- Everything else is executed as a command

`#SBATCH --job-name=bowtie2`

- Sets the name of the job
- Job names can be used to manage jobs, but unlike *jobids* they are not necessarily unique, so care should be taken
  - E.g

```
scancel –n bowtie2
```

- By default `squeue`, only shows 8 first characters of job names
  - Can be controlled with `--format` option, *e.g:*

```
squeue --format="%.18i %.9P %.30j %.8u %.2t %.10M %.6D %R"
```

#SBATCH --account=<project>

System specific
Required for CSC systems

- Billing project needs to be specifed in the batch job script
- If omitted, job submission will fail
- Use project name. Project names typically look something like ”project_12345”
- You can check the projects you are a member of with command: `id`

```
uid=10004231(training027) gid=10004155(4training)
groups=10004155(4training),2000745(project_2000745)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

- In CSC systems you can also use command `csc-workspace`
  - Or check web portal: https://my.csc.fi/myProjects

```
#SBATCH --output=output_%j.txt
#SBATCH --error=errors_%j.txt
```

Optional
–but highly recommended

- When running a command interactively, text output from the command to the shell is delivered via the stdout (standard out) stream. Error messages from the command are sent through the stderr (standard error) stream.
- When running a batch, job these have to be saved to a file. In puhti both stdout and stderr are directed to file ”slurm-<jobid>.out” by default.
- You can choose to save them separately by specifying options `--output` and `–error`
  - `%j` is replaced with the job id number in the actual file name
- What gets written to stdout and stderr depends on the program. If you are unfamiliar with the program, it’s always safest to capture both

```
#SBATCH --time=02:00:00
```

- Time reserved for the job in hh:mm:ss
- When the time runs out the job will be terminated!
- With longer reservations the job might spend longer in the queue
- Maximum time determined by the queue

```
#SBATCH –-ntasks=1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
```

- In this case we are running a shared memory program. It must run inside one node, so we specify:
  - 1 task (`--ntasks`)
  - 1 node (`--nodes`)
  - 6 cores (`--cpus-per-task`)
- For a MPI program we would not need to run inside one node, so we might specify simply something like:
  - `#SBATCH --ntasks=36`
- Check software documentation
  - Many bioinformatics software can not utilize more than one core
  - Some can use threads and run as a shared memory job
  - Only very few utilize MPI

`#SBATCH --mem=6G`

- The amount of memory reserved for the job
- For MPI jobs use `--mem-per-cpu`
- For shared memory (OpenMP) jobs --mem is easiest
  - `--mem-per-cpu` can be used, but rember to adjust if changing core number
- Keep in mind the specifications for the nodes. Jobs with impossible requests are rejected
- If you reserve too little memory the job will fail
- If you reserve too much memory, your job will spend much longer in queue

#SBATCH --partition=small

- The queue (partition) the job should be submitted to
- You can check the available queues with command

```
sinfo [-p <partition_name>]
```

or

```
scontrol show partition [<partition_name>]
```

Additional resources

System specific
These examples for Puhti

- Local storage:

  `#SBATCH --gres=nvme:<local_storage_space_per_node>`

  - Environment variable $LOCAL_SCRATCH
  - Make sure to copy the results at then end of the jobs. The local storage is emptied after the job has finished
  - Can be helpful when running heavily I/O bound applications
  - Most nodes have no local storage
  - Available in `small`, `large` and `longrun` partitions

- GPUs

  `#SBATCH --gres=gpu:v100:<number_of_gpus_per_node>`

  - The --gres reservation is on a per node basis. There are 4 GPUs per GPU node
  - Available in gpu and gputest partitions

```
module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK chr_18 reads.fq > out.sam
```

- Remember to load modules if necessary
- By default the working directory is the directory where you submitted the job
  - If you include a `cd` command, make sure it points to correct directory
- Command syntax depends on the software
  - It's not enough to reserve the cores: Also remember to tell the program to use them!
  - See application documentation for correct syntax
  - You can use system variable $SLURM_CPUS_PER_TASK
- For MPI jobs, remember to use srun

```
srun mympiprog <options>
```

Array jobs

- Best suited for running the same analysis for large number of files
- Defined by adding `--array` option to batch job script
  - Can be defined as a range or a list. For ranges step size can be defined
- When run, variable $SLURM_ARRAY_TASK_ID will be replaced with the current array index
- Note that the batch job script is executed for each iteration, so things that should be done only once should not be included in the script
- All resource definitions are for a single sub-task
- Job can be controlled as whole or at sub-task level

`scancel 12345`	cancels the whole array job
`scancel 12345_1`	cancels just one sub-task

Defining the array

- As a range:

  --array=1-100

- Range with step size:

  --array=1-100:20

- A list:

  --array=1,34,77

  o Can be useful when some subjobs fail

- It is also possible to limit simultaneously running subjobs:

  --array=1-100%5

  o Can be useful e.g if program uses limited licenses

Array size

- There are two limits to maximum array job size: Maximum number of jobs allowed to be submitted by the user (MaxSubmit) and maximum array index number (MaxArraySize -1)
- These are system/partition/account/user specific and may change
- To see current Slurm MaxJobs (maximum running jobs) and MaxSubmit (maximum submitted jobs) limits for user:

```
sacctmgr list user $USER withassoc
```

- To see current MaxArrayJob:

```
scontrol show config | grep MaxArraySize
```

Simple array job example

```
#!/bin/bash
#SBATCH --job-name=array_job
#SBATCH --account=<project>
#SBATCH --output=array_job_out_%A_%a.txt
#SBATCH --error=array_job_err_%A_%a.txt
#SBATCH --ntasks=1
#SBATCH --mem=4G
#SBATCH --time=02:00:00
#SBATCH --partition=small
#SBATCH --array=1-50

# run the analysis command
my_prog data_${SLURM_ARRAY_TASK_ID}.inp data_${SLURM_ARRAY_TASK_ID}.out
```

In this example the actual command run at each iteration will be:

```
myprog data_1.inp data_1.out
myprog data_2.inp data_2.out
..
myprog data_50.inp data_50.out
```

Using a list of file names in an array job

- Often it is easiest to use a list of input filenames
- You can use a combination of sed and the $SLURM_ARRAY_TASK_ID variable
- To create a list of filenames

```
ls data_*.inp > namelist
```

- To print a single row in a file by row number:

```
sed –n "row_number"p inputfile
```

- Example commands in batch job script

```
name=$(sed -n ${SLURM_ARRAY_TASK_ID}p namelist)
my_prog ${name} ${name}.out
```

Example batch job script using a list of file names in an array job

```
#!/bin/bash
#SBATCH --job-name=array_job
#SBATCH --account=<project>
#SBATCH --output=array_job_out_%A_%a.txt
#SBATCH --error=array_job_err_%A_%a.txt
#SBATCH --ntasks=1
#SBATCH --mem=4G
#SBATCH --time=02:00:00
#SBATCH --partition=small
#SBATCH --array=1-50

# set input file to be processed
name=$(sed -n ${SLURM_ARRAY_TASK_ID}p namelist)
# run the analysis command
my_prog $name $name.out
```

Array job using sbatch_commandlist

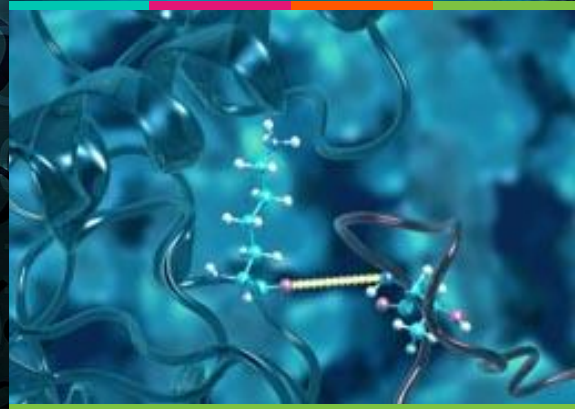
- sbatch_commandlist executes given command list as independent sub-tasks of an array job
- Command launches the array job and monitors it until all the sub jobs have finished
- Syntax:

```
sbatch_commandlist –project <project> -commands list_of_independet_command_lines_to_execute
```

Optional options:

`-t`	SLURM run time reservation. Default: 12:00:00
`-mem`	SLURM memory reservation. Default 8000
`-jobname`	SLURM job name. Default: array_job
`-threads`	SLURM num-cpus. Default: 1
`-project`	SLURM Accounting project. Default: the one set up up with csc-workspaces set

Managing batch jobs



Submitting and cancelling jobs

- The script file is submitted with command

```
sbatch batch_job.file
```

- sbatch options are usually listed in the batch job script, but they can also be specified on command line, e.g.

```
sbatch --job-name=test2 --time=00:05:00 batch_job_file.sh
```

- Job can be deleted with command

```
scancel <jobid>
```

Queues

- The job can be followed with command squeue:

`squeue`	(shows all jobs in all queues)
`squeue –p <partition>`	(shows all jobs in single queue (partition))
`squeue –u <username>`	(shows all jobs for a single user)
`squeue –j <jobid>`	(shows status of a single job)

- To estimate the start time of a job in queue

```
scontrol show job <jobid>
```

  - row "StartTime=..." gives an estimate on the job start-up time, e.g.

```
StartTime=2019-11-12T19:46:44 EndTime=Unknown
```

Available queues

- You can check available queues on each machine with command:

```
sinfo -l
```

```
PARTITION       AVAIL  TIMELIMIT   JOB_SIZE ROOT OVERSUBS     GROUPS  NODES       STATE NODELIST
small*             up 3-00:00:00          1   no       NO        all     41     drained r17c[01-24],r18c[13-28,32]
small*             up 3-00:00:00          1   no       NO        all     50       mixed
r01c[07,18],r02c[14,21,40],r03c[24,34,39,47],r04c[05,18,21,23,28,39,47],r05c[10-
11,30,32,43,47],r06c[02,06,09,19,23,28,30,34,37,46,50,58],r07c[09,24,30,33],r13c[15,19,31,37],r14c[10-
11],r15c[19,44],r16c[08,25],r18c[33-34]
small*             up 3-00:00:00          1   no       NO        all    567   allocated r01c[01-06,08-17,19-48],r02c[01-13,15-
20,22-39,41-48],r03c[01-23,25-33,35-38,40-46,48],r04c[01-04,06-17,19-20,22,24-27,29-38,40-46,48],r05c[01-09,12-29,31,33-42,44-
46,48-64],r06c[01,03-05,07-08,10-18,20-22,24-27,29,31-33,35-36,38-45,47-49,51-57,59-64],r07c[07-08,10-23,25-29,31-32,34-
56],r13c[01-14,16-18,20-30,32-36,38-48],r14c[01-09,12-48],r15c[01-18,20-43,45-48],r16c[01-07,09-24,26-48],r17c[25-48],r18c[01-
12,29-31,35-48]
..
..
```

Available nodes

- You can check available nodes in each queue with command:

```
sjstat -c
```

```
Scheduling pool data:
-------------------------------------------------------------
Pool        Memory  Cpus  Total Usable   Free  Other Traits
-------------------------------------------------------------
small*     382000Mb    40     92     92      0  type_l
small*     190000Mb    40    526    485      0  type_m
small*     382000Mb    40     40     40      0  type_io
large      382000Mb    40     92     92      0  type_l
large      190000Mb    40    526    485      0  type_m
large      382000Mb    40     40     40      0  type_io
test       190000Mb    40      6      6      4  type_m
longrun    382000Mb    40     92     92      0  type_l
longrun    190000Mb    40    526    485      0  type_m
longrun    382000Mb    40     40     40      0  type_io
fmi        190000Mb    40    238    236    188  type_m
fmitest    190000Mb    40      2      2      2  type_m
hugemem    764000Mb    40     12     12      0  type_xl
hugemem   1532000Mb    40      6      6      4  type_bigmem
hugemem_l  764000Mb    40     12     12      0  type_xl
hugemem_l 1532000Mb    40      6      6      4  type_bigmem
gputest    382000Mb    40      2      2      1  type_gpu
gpu        382000Mb    40     78     74      1  type_gpu
```

Most frequently used SLURM commands

Command	Description
`srun`	Run a parallel job.
`salloc`	Allocate resources for interactive use.
`sbatch`	Submit a job script to a queue.
`scancel`	Cancel jobs or job steps.
`sinfo`	View information about SLURM nodes and partitions.
`squeue`	View information about jobs located in the SLURM scheduling queue
`smap`	Graphically view information about SLURM jobs, partitions, and set configuration parameters
`sjstat`	display statistics of jobs under control of SLURM (combines data from sinfo, squeue and scontrol)
`scontrol`	View SLURM configuration and state.
`sacct`	Displays accounting data for batch jobs.

Monitoring resource usage

seff

- Command `seff` will print a summary of requested and used resources for both running and finished batch jobs

```
seff  <jobid>
```

Example 1: Core utilization bad

```
>seff 123456

Job ID: 123456
Cluster: puhti
User/Group: user/group
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 8
CPU Utilized: 0:51:01
CPU Efficiency: 12.48% of 06:56:08 core-walltime
Memory Utilized: 5.98 GB
Memory Efficiency: 75.89% of 7.88 GB
Job consumed X.XX CSC billing units based on cpu reservation
   multiplier
```

Example 2: Memory utilization bad

```
>seff 123456

Job ID: 123456
Cluster: puhti
User/Group: user/group
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 8
CPU Utilized: 05:49:01
CPU Efficiency: 83.89% of 06:56:08 core-walltime
Memory Utilized: 5.98 GB
Memory Efficiency: 6.25% of 92.59 GB
Job consumed X.XX CSC billing units based on cpu reservation
   multiplier
```

Example 3: Job failed due to time reservation running out

```
>seff 123456

Job ID: 1234566
Cluster: puhti
User/Group: user/csc
State: TIMEOUT (exit code 1)
Nodes: 1
Cores per node: 12
CPU Utilized: 02:06:41
CPU Efficiency: 70.30% of 03:00:12 core-walltime
Memory Utilized: 24.70 GB
Memory Efficiency: 72.27% of 34.18 GB

Job consumed 6.01 CSC billing units based on cpu reservation multiplie
```

Example 4: Job failed probably due to memory reservation running out

```
>seff 123456

Job ID: 1234566
Cluster: puhti
User/Group: user/csc
State: FAILED (exit code 1)
Nodes: 1
Cores per node: 12
CPU Utilized: 02:06:41
CPU Efficiency: 70.30% of 03:00:12 core-walltime
Memory Utilized: 35.70 GB
Memory Efficiency: 101.3% of 34.18 GB

Job consumed 6.01 CSC billing units based on cpu reservation multiplie
```

sacct

- Command `sacct` can be used to study past jobs
- Usefull when deciding proper resource requests

`sacct`	Short format listing of jobs starting from midnight today
`sacct –j <jobid>`	information on single job
`sacct –S YYYY-MM-DD`	listing start date
`sacct –l`	long format output
`sacct –o`	list only named data fields, e.g.

```
sacct -o jobid,jobname,reqmem,maxrss,averss,state,elapsed -j <jobid>
```

sacct

Some useful data fields:

`jobid`	Job id number
`jobname`	Job name
`reqcpus`	Cores requested from SLURM
`reqmem`	Memory requested from SLURM
`maxrss`	Maximum used memory
`averss`	Average used memory per process/core
`state`	Exit status of job
`elapsed`	Execution time

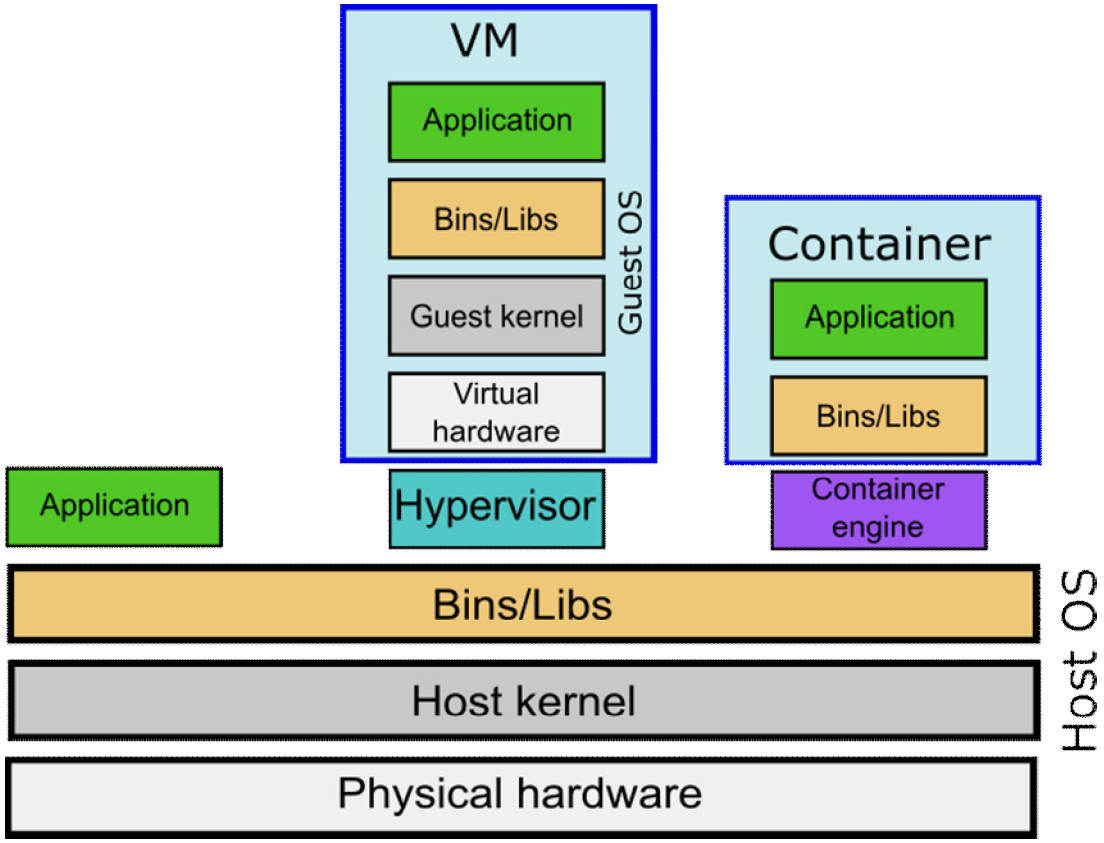
```
sacct -o jobid,jobname,ntasks,reqnodes,allocnodes,reqcpus,alloccpus,reqmem,maxrss,averss,timelimit,elapsed,state –j 17317981
```

```
JobID           JobName   NTasks ReqNodes AllocNodes  ReqCPUS  AllocCPUS    ReqMem     MaxRSS     AveRSS  Timelimit    Elapsed     State
------------ ---------- -------- -------- ---------- -------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
17317981        cretest              1          1        8          8      504Mc                          00:01:00   00:00:01  COMPLETED
17317981.ba+    batch        1        1          1        8          8      504Mc      1580K      1580K              00:00:01  COMPLETED
```

Exercises 1-3

Singularity containers

Virtual machines vs containers



- Containers are a way to package software with its dependencies (libraries,etc)
- Popular container engines include Docker, Singularity, Shifter
- Container benefits:
  - Ease of installation: even complex software packages can be installed with single command (provided a container is available)
  - Environment isolation: possibility to run software that is incompatible with host system
  - Enviroment reproducibility: analysis environment can be saved as a whole

Docker

- Widely used
- Many software packages available as Docker containers

However:

- Not suitable for HPC systems
  - Requires root access to run, etc

Singularity

- Designed for HPC environments
- Containers can be run with user level rights
  - Building new containers requires root access
- Minimal performance overhead
- Supports MPI
- Can import and run Docker containers
- Command syntax very similar to Docker
  - Docker instructions can often be utilized almos as-is

Running Singularity containers

- Basic syntax

```
singularity run [run options...] <container>
```

Runs script called “singularity” in the root directory of the container

```
singularity exec [exec options...] <container> <command>
```

Executes a command in the container

```
singularity shell [shell options...] <container>
```

Opens a shell in the container

File system

- Containers have their own, internal file system
- To use host file system for input/output, host directories need to mapped to container directories

  `--bind /data:/mnt`

  host directory `/data` is mapped to directory `/mnt` inside the container
- Target directory inside the container does not need to exist. It is created as necessary
- More than one directory can be mapped if necessary

- Mounting container directories with the same path as host directories allows you to use same command line as you would without a container – but can be confusing when troubleshooting

```
singularity exec --bind /scratch/project_12345/data:/scratch/project_12345/data myimage.sif \
myprog --input /scratch/project_12345/data/myfile

singularity exec --bind $PWD:$PWD myimage.sif myprog --input myfile
```

- Using different name space for the container may be clearer, but you need to remember use it in commands

```
singularity exec --bind /scratch/project_12345/data:/container/data myimage.sif \

myprog --input /container/data/myfile
```

Matter of taste: take you pick

Using Docker containers with Singularity

- You can build a Singularity container from a Docker container with normal user rights:

```
singularity build <image> docker://<address>
```

For example:

```
singularity build pytorch_19.10-py3.sif \
docker://nvcr.io/nvidia/pytorch:19.10-py3
```

- Instructions for Puhti:

https://docs.csc.fi/computing/containers/run-existing/

Building a new Singularity container

Requires root access:
Can not be done directly
In e.g. Puhti

- Typical steps

1. Build a basic container in sandbox mode (`--sandbox`)
    1. Uses a folder structure instead of an image file
    2. Requires root access!
2. Open a shell in the container and install software
    1. Depending on base image system, package managers can be used to install libraries etc (`apt install`, `yum install` etc)
    2. Installation as per software developer instructions
3. Build a production image from the sandbox
4. (optional) Make a definition file and build a production image from it

Singularity in Puhti

- Singularity installed only in compute nodes
  - Singularity jobs need to run as batch jobs
  - No need to load a module
- Users can run their own containers
- Some CSC software installations provided as containers
  - See software pages for details

- Documentation (under construction):
  - https://docs.csc.fi/computing/containers/run-existing/

Exercise 4

Note on Exercise 4

Installation type	Size on disc	Number of files	Time to install	Number of unprintable words
Native	1,9 MB	47	~5h (ok, not fair)	~100
Conda	1,1 GB	27464	10 min	0
Singularity	339 MB	1	10 min	0

- Containers are not the solution for everything, but they do have their uses…

Troubleshooting

Getting familiar with a new program

- Read the manual
- It may be helpful to first try run the program interactively to find the correct command line options
  - Good chance to use `top` to get rough estimate on memory use etc
- If developers provide some test or example data, run it first
  - Make sure the results are as expected
- You can use test queue to check your batch job script
  - Limits : 15 min, 2 nodes
  - Job turnaround usually very fast
  - Can be useful to spot typos, missing files etc before submitting a job that will spend long in the queue
- Before very large runs, it's a good idea do a smaller trial run
  - Check that results are as expected
  - Check resource usage after test run and adjust accordingly
  - Try different core numbers and see how the software scales

Troubleshooting checklist

Start with these if you job fails:

1. Did the job run out of time?
2. Did the job run out of memory?
3. Did the job actually use resources you specified?
    1. Problems in batch job script can cause parameters to be ignored and default values are used instead
4. Did it fail immediately or did it run for some time?
    1. Jobs failing immediately are often due to something simple like typos in command line, missing inputs, bad parameters etc
5. Check the error file captured by batch job script
6. Check any other error files and logs the program may have produced
7. Error messaged can sometimes be long, cryptic and a bit intimidating, but try skimming through them and see if you can spot something ”human readable” instead of ”nerd readable”
    1. Often you can spot the actual problem easily if you go through the whole message. Something like ”required input file so-and-so missing” or ”parameter X out of range” etc.

And always: please don’t hesitate to contact us at servicedesk@csc.fi