Introduction to Abisko and Kebnekaise

Pedro Ojeda-May, Jerry Eriksson, and Birgitte Brydsö

HPC2N,
UmeåUniversity,

901 87, Sweden.

November 7, 2016

Table of contents

File Systems

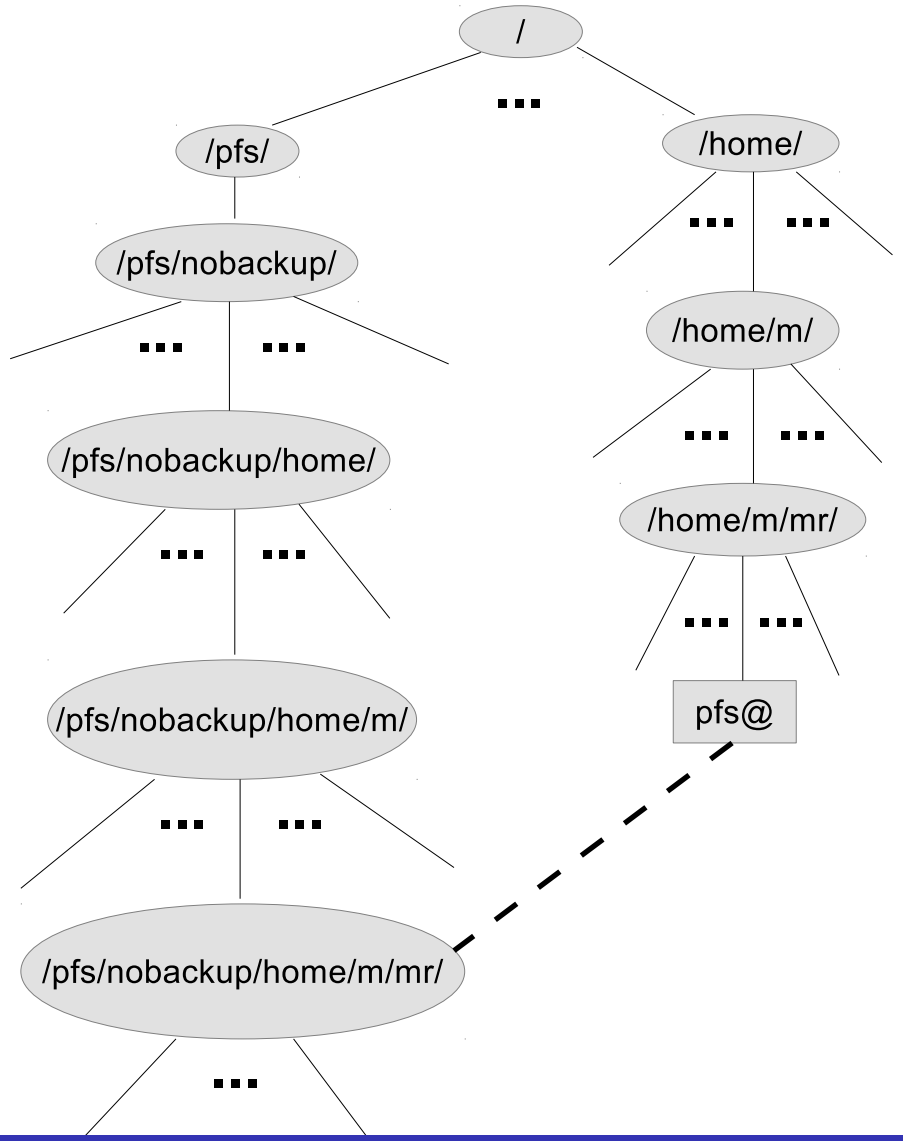
There are 2 file systems

AFS

- Your home directory (cd $HOME)
- Backed up regularly
- NOT accessable by the batch system

PFS

- Parallel file system
- NO BACKUP
- Accessable by the batch system



PFS

- Offers high performance when accessed from the nodes
- This is the correct place to run all your batch jobs
- To create soft link from your home directory to your corresponding home on the parallel file system
  `ln -s /pfs/nobackup$HOME $HOME/pfs`
- Then if you use
  `cd pfs`
  from your home directory you will end up in your "parallel" home directory

Batch System (SLURM)

- Large/parallel programs, run through the batch system
- Keeps track of available system resources
- Takes care of scheduling jobs of multiple users, running tasks simultaneously
- Enforces local system resource usage and job scheduling policies
- Users submit to a queue (running, idle, blocked)
- Same batch system on Abisko and Kebnekeise. The differences are that there are GPUs and KNLs which can be allocated on Kebnekeise.
- Guides at: http://www.hpc2n.umu.se/support/

Batch System (SLURM)

Useful commands:

- squeue -u <username>
- srun <commands for your job/program>
- salloc
- scontrol show job <jobid>
- scancel <jobid>

Interactive jobs

```
$ salloc -N 1 -n 8 --time=1:30:00
```

It asks to allocate 1 node/8 processors. Once job has been allocated we can run our programs

```
srun -n 2 my_program
```

plus additional environment variables if the case.

Job script

```
#!/bin/bash
#SBATCH -A SNICYYYY-XX-NN
#SBATCH -n 48
#SBATCH --time=01:00:00

module add openmpi/gcc
srun ./parallel_prog args
```

- Submitting:
  `sbatch <jobscript>`
- Show the job queue:
  `squeue [-u username]`
- Delete a job:
  `scancel <jobid>`

Job script

```
#!/bin/bash
#SBATCH -A SNICYYYY-XX-NN
#SBATCH -n 48
#SBATCH --time=01:00:00

module add openmpi/gcc
srun ./parallel_prog args
```

Your account (-A)

- The account is your project id
- Low priority if not set
- You can find your project id by running: `projinfo`

Job script

```
#!/bin/bash
#SBATCH -A SNICYYYY-XX-NN
#SBATCH -n 48
#SBATCH --time=01:00:00

module add openmpi/gcc
srun ./parallel_prog args
```

Number of tasks (-n)

- The number of tasks is for the most cases the number of processes you want to start.
- The default value is one
- e.g. number of MPI tasks
- e.g. number of serial programs

Job script

```
#!/bin/bash
#SBATCH -A SNICYYYY-XX-NN
#SBATCH -n 48
#SBATCH --time=01:00:00

module add openmpi/gcc
srun ./parallel_prog args
```

Number of cores per task (-c)

- For multi threaded applications (OpenMP/pthreads/...)
- indicates the number of cores each task can use
- The default value is one
- Maximum is 48

Job script

```
#!/bin/bash
#SBATCH -A SNICYYYY-XX-NN
#SBATCH -n 48
#SBATCH --time=01:00:00

module add openmpi/gcc
srun ./parallel_prog args
```

The run/wallclock time
`D-HH:MM:SS`

- Runtime (wall clock time) of your job
- Try to estimate correctly
  - Hard limit
  - Shorter jobs are more likely to fit into slots of unused space faster.

Job script

```
#!/bin/bash
#SBATCH -A SNICYYYY-XX-NN
#SBATCH -n 48
#SBATCH --time=01:00:00

module add openmpi/gcc
srun ./parallel_prog args
```

Load modules needed or other things. (This is for your program that is compiled with the GCC compiler and the OpenMPI library.)

Job script

```
#!/bin/bash
#SBATCH -A SNICYYYY-XX-NN
#SBATCH -n 48
#SBATCH --time=01:00:00

module add openmpi/gcc
srun ./parallel_prog args
```

Run your MPI application using `srun`

- Starts the required number of processes
- Note! If your program is serial it will start many instances

Job script

```
#!/bin/bash
#SBATCH -A SNICYYYY-XX-NN
#SBATCH -n 48
#SBATCH --time=01:00:00

module add openmpi/gcc
srun ./parallel_prog args
```

Run your multi threaded application on 36 cores.
Change the marked lines with:

```
#SBATCH -c 36

export OMP_NUM_THREADS=36

./my_OpenMP_program args
```

Job script

```
#!/bin/bash
#SBATCH -A SNICYYYY-XX-NN
#SBATCH -n 48
#SBATCH --time=01:00:00

module add openmpi/gcc
srun ./parallel_prog args
```

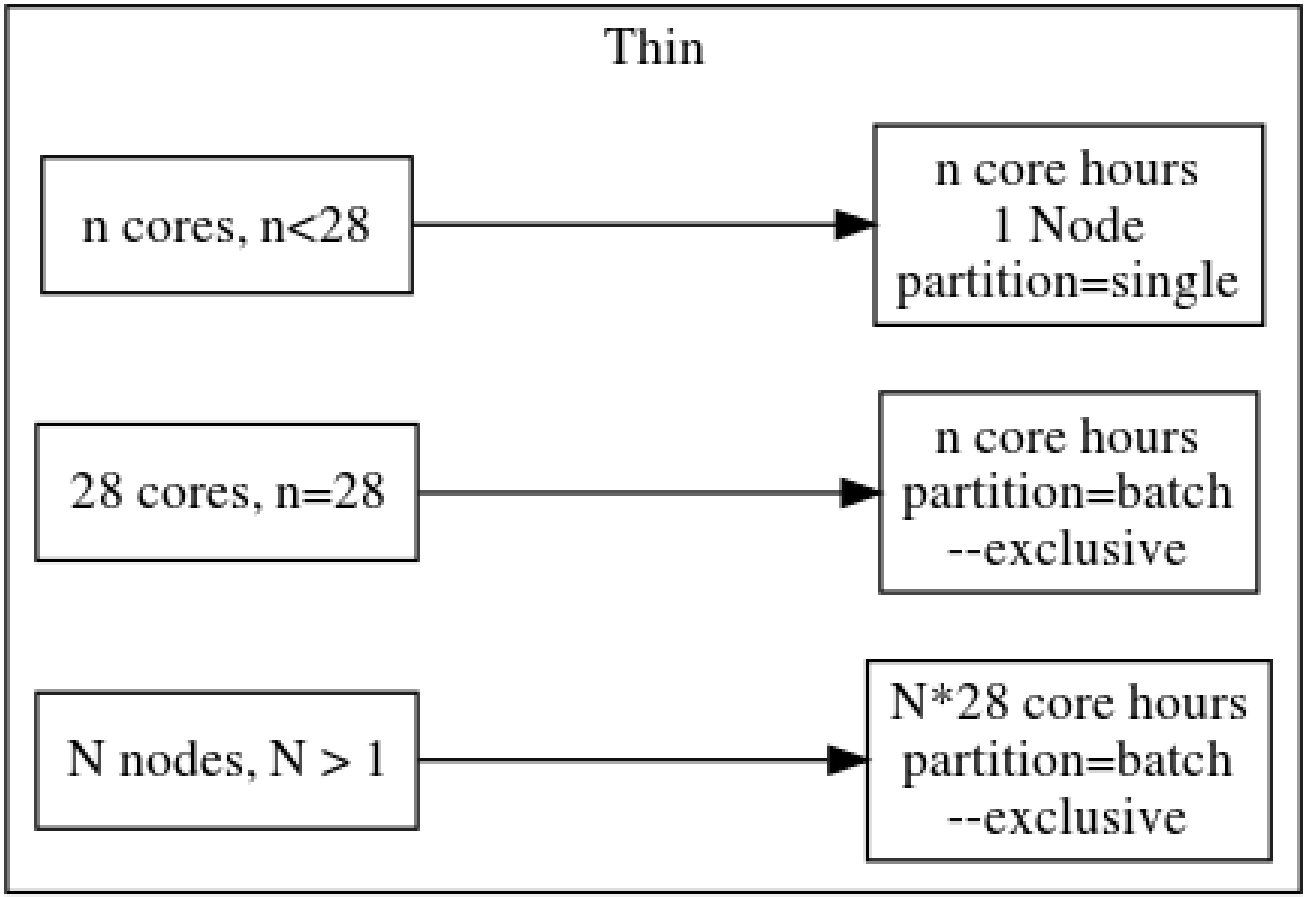
Output

- Put stdout into the file <*jobid*>.out
  `#SBATCH --output=%J.out`
- Put stderr into the file <*jobid*>.err
  `#SBATCH --error=%J.err`
- By default both to slurm-<*jobid*>.out

Input

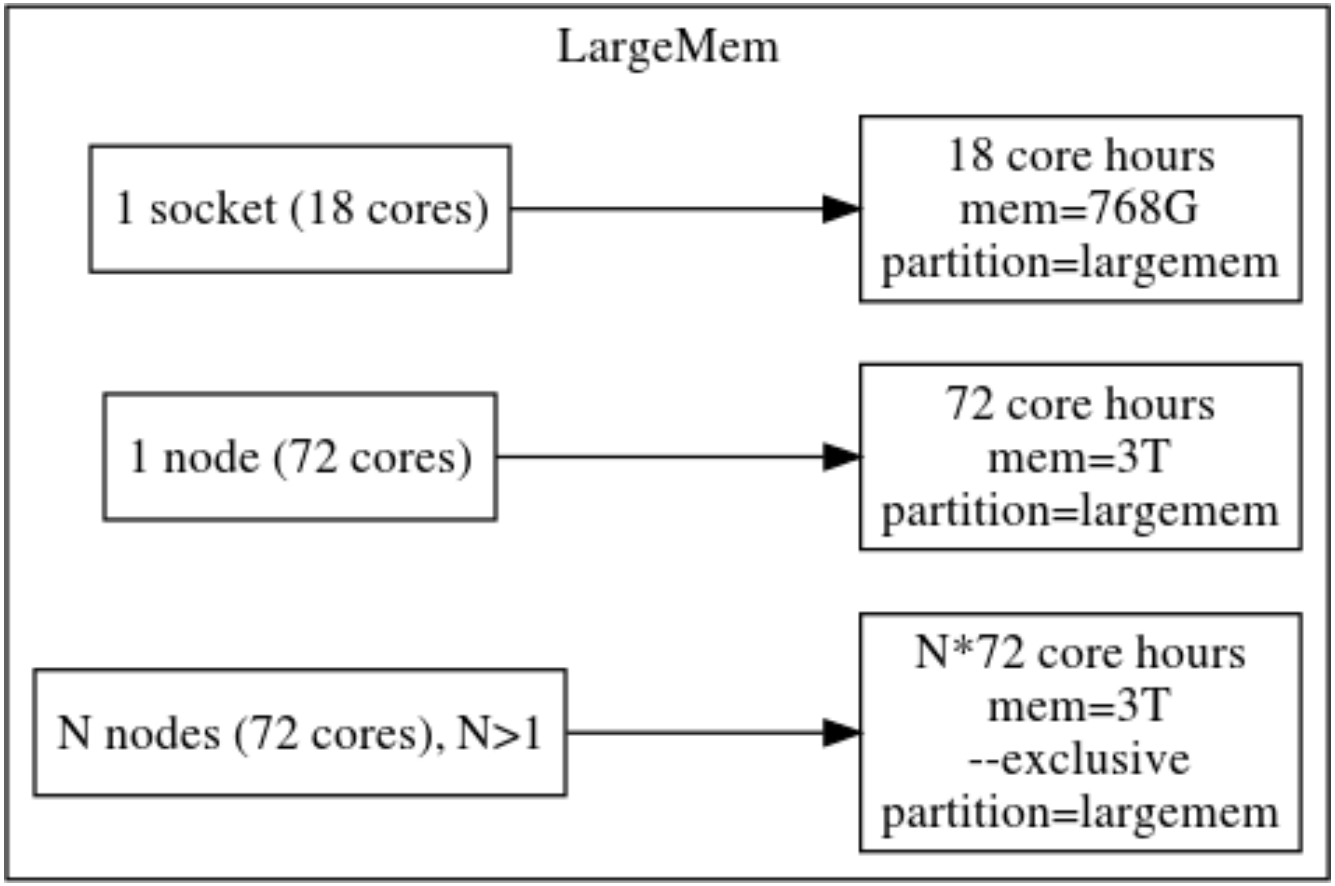
- Use `file.txt` as stdin
  `#SBATCH --input=file.txt`

How resources are charged?



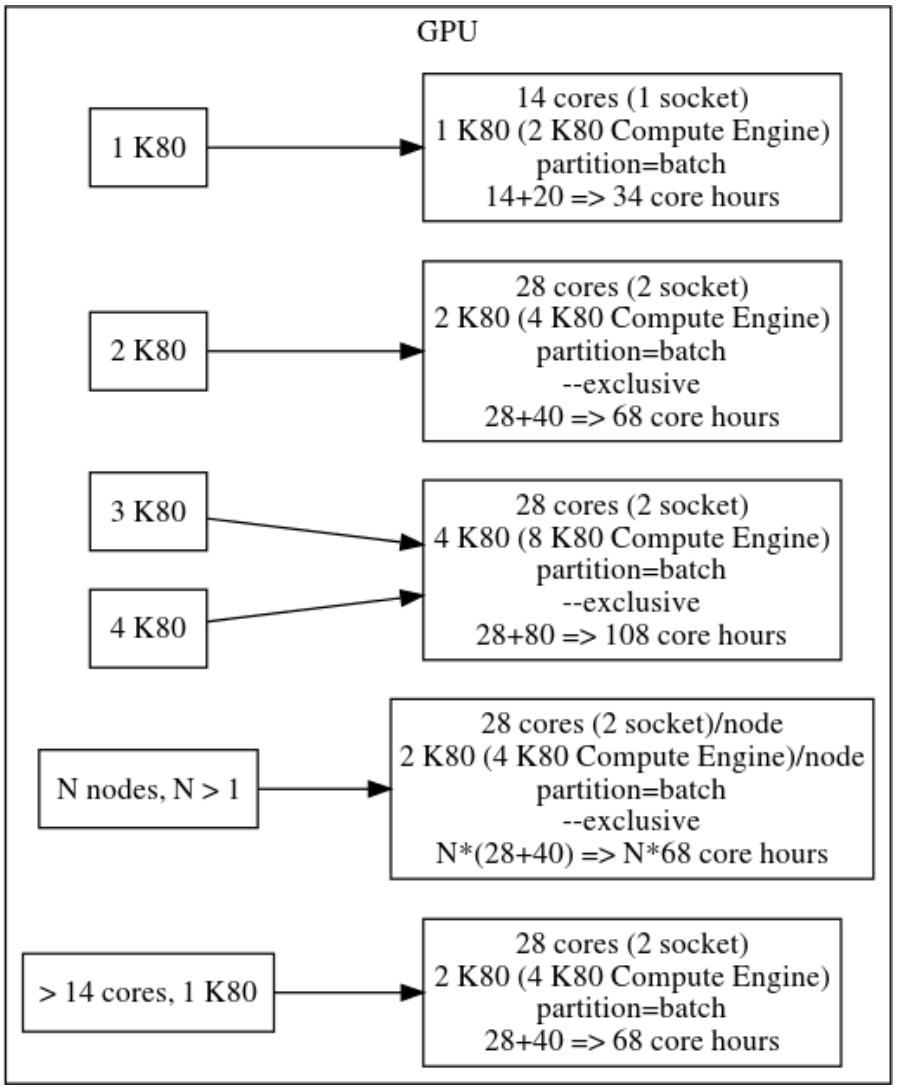
Thin node

How resources are charged?



Fat node

How resources are charged?



Connecting from a Windows System

You need an ssh client to connect.

- PuTTY
- Cygwin

If you want to open graphical displays, you need an X11 server

- Xming
- Cygwin

Transferring files (sftp or scp)

- WinSCP
- FileZilla (only sftp)
- PSCP/PSFTP

Connecting from a UNIX/Linux System

- Login with ssh:
  ```
  local> ssh username@abisko.hpc2n.umu.se
  ```
- If you want to open graphical displays, you need to enable X11 Forwarding:
  ```
  local> ssh -X username@abisko.hpc2n.umu.se
  ```
- Use scp for file transfer:
  ```
  local> scp username@abisko.hpc2n.umu.se:file /tmp
  local> scp file username@abisko.hpc2n.umu.se:file
  ```

GROMACS

```
$module spider GROMACS/2016-hybrid
----------------------------------------------------------
  GROMACS: GROMACS/2016-hybrid
----------------------------------------------------------
 Description:
   GROMACS is a versatile package to ...
   Homepage: http://www.gromacs.org

   You will need to load all module(s) on any one of
   the lines below before the "GROMACS/2016-hybrid" ...

   GCC/5.4.0-2.26  CUDA/8.0.44  OpenMPI/2.0.1
   GCC/6.2.0-2.27  OpenMPI/2.0.1
```

GROMACS batch script

```
#!/bin/bash
#SBATCH -A ~SNICYYYY-XX-NN
#SBATCH -J G2016-gpu
#SBATCH -t 01:00:00
#SBATCH -n 12
#SBATCH -c 7
#SBATCH --gres=gpu:k80:2
#SBATCH -p batch
module add CUDA/8.0.44   GCC/5.4.0-2.26
module add OpenMPI/2.0.1 GROMACS/2016-hybrid
mdargs="-ntomp $SLURM_CPUS_PER_TASK"
mpirun -np $SLURM_NTASKS gmx_mpi mdrun $mdargs \
-dlb yes  -v -deffnm npt
```

GROMACS output

```
Running on 3 nodes with total 84 cores,
84 logical cores, 12 compatible GPUs
  Cores per node:            28
  Logical cores per node:    28
  Compatible GPUs per node:  4
  All nodes have identical type(s) of GPUs
```

GROMACS performance 100K atoms