Introduction to HPC2N

Birgitte Brydsø, Mirko Myllykoski, and Pedro Ojeda-May

HPC2N, Umeå University

22 January 2020

Overview

- Projects - compute and storage
- Using our systems
- The File System
- The Module System
    - Overview
    - Compiler Tool Chains
    - Examples
- Compiling/linking with libraries
- The Batch System (SLURM)
    - Overview
    - Simple example

Projects - compute and storage

- Apply for a **compute project** in SUPR (need SUPR account) `https://supr.snic.se/round/compute/`
  - Small ($\leq$ 5000 core-h/month, at least PhD student to apply)
  - Medium
  - Large
- You can now apply for a HPC2N account if you don't have one
- PFS quota is only 25 GB so you will need to apply for a **storage project** `https://supr.snic.se/round/storage/`
  - Small (up to 2 TB, at least PhD student to apply)
  - Medium

Using our systems

1. Connect to:

   `kebnekaise.hpc2n.umu.se`

2. Transfer your files and data (optionally)
3. Compile own code, install software, or run pre-installed software
4. Create batch script, submit batch job
5. Download data/results

Using our systems
Connecting to HPC2N's systems

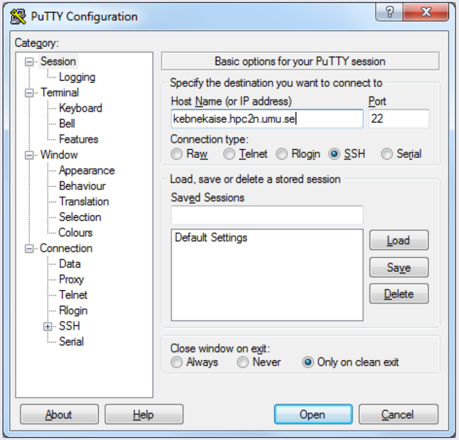
- **Linux, OS X:**
  - `ssh username@kebnekaise.hpc2n.umu.se`
  - Use `ssh -Y ....` if you want to open graphical displays.
- **Windows:**
  - Get an SSH client (PuTTY, Cygwin, MobaXterm ...)
  - Get an X11 server if you need graphical displays (Xming ...)
  - Start the client and login to

    `kebnekaise.hpc2n.umu.se`
  - More information here:

    https://www.hpc2n.umu.se/documentation/guides/windows-connection
- **Mac/OSX:** Guide here:

  https://www.hpc2n.umu.se/documentation/guides/mac-connection

Get the Zip file (http://www.putty.org/) with both PuTTY, PSCP, and PSFTP. Unzip, run putty.exe


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): kebnekaise.hpc2n.umu.se

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

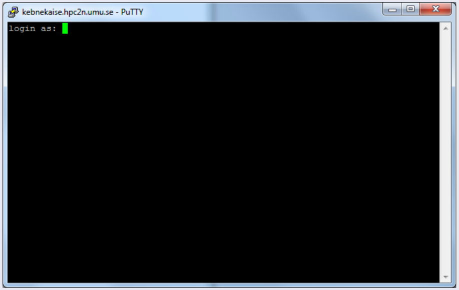
Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

Using our systems
Connecting from a Windows System with PuTTY

Enter your username and then your password.



```
login as: 
```

Using our systems
Transfer your files and data

- **Linux, OS X:**
  - Use scp for file transfer:

```
local> scp username@kebnekaise.hpc2n.umu.se:file .
local> scp file username@kebnekaise.hpc2n.umu.se:file
```

- **Windows:**
  - Download client: WinSCP, FileZilla (sftp), PSCP/PSFTP, ...
  - Transfer with sftp or scp
- https://www.hpc2n.umu.se/documentation/filesystems/filetransfer
- **Mac/OSX:**
  - Transfer with sftp or scp (as for Linux) using Terminal
  - Or download client: Cyberduck, Fetch, ...
- More info in guides (see previous slide) and here:
  https://www.hpc2n.umu.se/documentation/filesystems/filetransfer

Using our systems
Editors

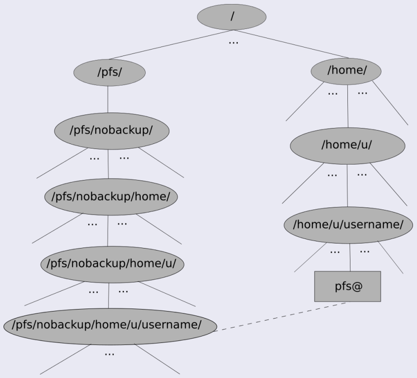
Editing your files

- Various editors: vi, vim, nano, emacs ...
- Example, nano:
  - `nano <filename>`
  - Save and exit nano: `Ctrl-x`
- Example, Emacs:
  - Start with: emacs
  - Open (or create) file: Ctrl-x Ctrl-f
  - Save: Ctrl-x Ctrl-s
  - Exit Emacs: Ctrl-x Ctrl-c
  - (If you want to run in an a separate emacs window, and with full functionality, you need to login with ssh -Y or similar, for X11 forwarding):

The File System

There are 2 file systems
More info here: http://www.hpc2n.umu.se/filesystems/overview

- **AFS**
  - This is where your home directory is located (cd $HOME)
  - Regularly backed up
  - NOT accesseable by the batch system (except the folder Public with the right settings)
- **PFS**
  - Parallel File System
  - NO BACKUP
  - Accessible by the batch system



The File System

AFS

- Your home directory is located in /home/u/username and can also be accessed with the environment variable $HOME
- It is located on the AFS (Andrew File System) file system
- Important! The batch system cannot access AFS since ticket-forwarding to batch jobs do not work
- AFS does secure authentification using Kerberos tickets

The File System
PFS

- The 'parallel' file system, where your 'parallel' home directory is located in /pfs/nobackup/home/u/username (/pfs/nobackup/$HOME)
- Offers high performance when accessed from the nodes
- The correct place to run all your batch jobs
- NOT backed up, so you should not leave files there that cannot easily be recreated
- For easier access, create a symbolic link from your home on AFS to your home on PFS:

  ```
  ln -s /pfs/nobackup/$HOME $HOME/pfs
  ```

  You can now access your pfs with `cd pfs` from your home directory on AFS

The Module System (Lmod)

Most programs are accessed by first loading them as a 'module'

Modules are

- used to set up your environment (paths to executables, libraries, etc.) for using a particular (set of) software package(s)
- a tool to help users manage their Unix/Linux shell environment, allowing groups of related environment-variable settings to be made or removed dynamically
- allows having multiple versions of a program or package available by just loading the proper module
- are installed in a hierarchial layout. This means that some modules are only available after loading a specific compiler and/or MPI version.

The Module System (Lmod)

Useful commands (Lmod)

- See which modules exists:
  `module spider` or `ml spider`
- Modules depending only on what is currently loaded:
  `module avail` or `ml av`
- See which modules are currently loaded:
  `module list` or `ml`
- Example: loading a compiler toolchain, here for GCC:
  `module load foss/version` or `ml foss/version`
- Example: Unload the above module:
  `module unload foss` or `ml -foss`
- More information about a module:
  `ml show <module>` or `module show <module>`
- Unload all modules except the 'sticky' modules:
  `ml purge`

The Module System
Compiler Toolchains

Compiler toolchains load bundles of software making up a complete environment for compiling/using a specific prebuilt software. Includes some/all of: compiler suite, MPI, BLAS, LAPACK, ScaLapack, FFTW, CUDA.

- Some currently available toolchains (check `ml av` for versions and full, updated list):
  - **GCC**: GCC only
  - **gcccuda**: GCC and CUDA
  - **foss**: GCC, OpenMPI, OpenBLAS/LAPACK, FFTW, ScaLAPACK
  - **gimkl**: GCC, IntelMPI, IntelMKL
  - **gimpi**: GCC, IntelMPI
  - **gompi**: GCC, OpenMPI
  - **gompic**: GCC, OpenMPI, CUDA
  - **goolfc**: gompic, OpenBLAS/LAPACK, FFTW, ScaLAPACK
  - **icc**: Intel C and C++ only
  - **iccifort**: icc, ifort
  - **iccifortcuda**: icc, ifort, CUDA
  - **ifort**: Intel Fortran compiler only
  - **iimpi**: icc, ifort, IntelMPI
  - **intel**: icc, ifort, IntelMPI, IntelMKL
  - **intelcuda**: intel and CUDA
  - **iomkl**: icc, ifort, Intel MKL, OpenMPI
  - **pomkl**: PGI C, C++, and Fortran compilers, IntelMPI
  - **pompi**: PGI C, C++, and Fortran compilers, OpenMPI

Compiling and Linking with Libraries

Linking

Figuring out how to link

- Intel and Intel MKL linking:

  `https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor`

- GCC, etc. **Use buildenv**
  - After loading a compiler toolchain, load `'buildenv'` and use `'ml show buildenv'` to get useful linking info
  - Example, foss (add relevant version):

    ```
    ml foss/version
    ml buildenv
    ml show buildenv
    ```

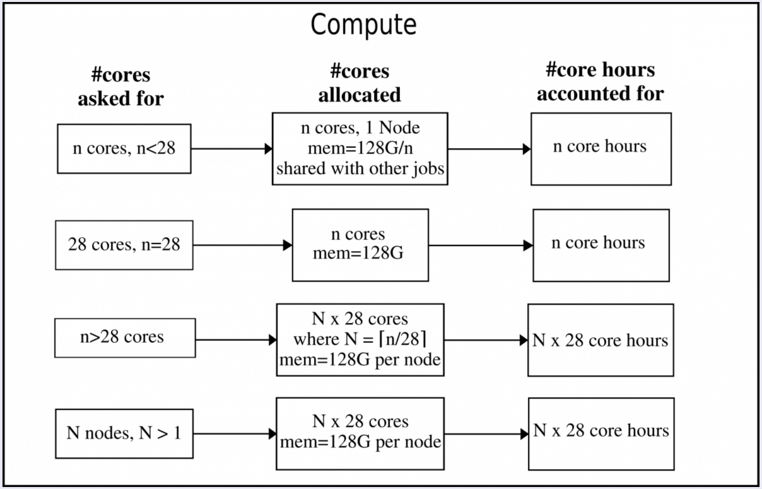
  - Using the environment variable (prefaced with $) for linking is highly recommended!
  - You have to load the buildenv module in order to use the environment variable for linking!

The Batch System (SLURM)

- Large/long/parallel jobs **must** be run through the batch system
- SLURM is an Open Source job scheduler, which provides three key functions
  - Keeps track of available system resources
  - Enforces local system resource usage and job scheduling policies
  - Manages a job queue, distributing work across resources according to policies
- In order to run a batch job, you need to create and submit a SLURM submit file (also called a batch submit file, a batch script, or a job script).
- Guides and documentation at: http://www.hpc2n.umu.se/support

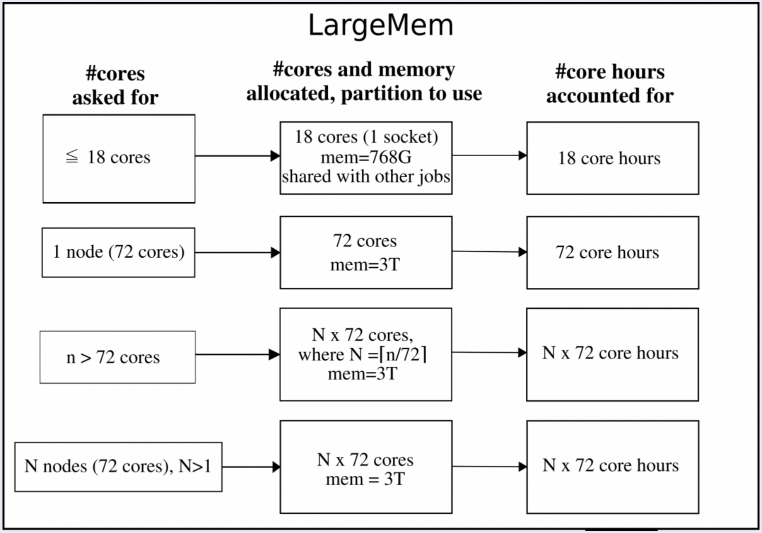
The Batch System

Accounting, Compute nodes, Kebnekaise



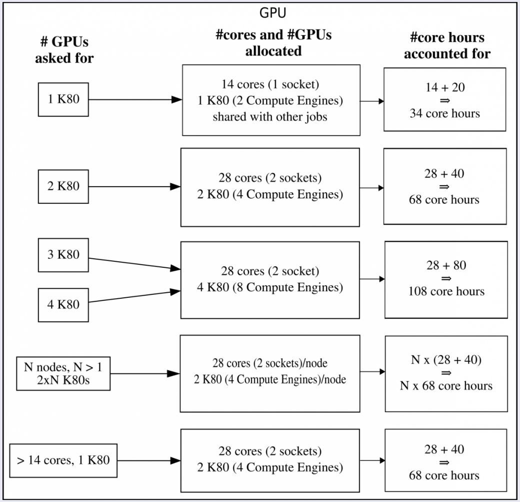
The Batch System

Accounting, largemem nodes, Kebnekaise



The Batch System

Accounting, GPU nodes, Kebnekaise. Same for the V100 as for the K80.



Note: V100s accounts like K80s and have **one** engine per card.

The Batch System (SLURM)

Useful Commands

- Submit job: `sbatch <jobscript>`
- Get list of your jobs: `squeue -u <username>`
- `srun <commands for your job/program>`
- `salloc <commands to the batch system>`
- Check on a specific job: `scontrol show job <job id>`
- Delete a specific job: `scancel <job id>`
- More detailed info about jobs:

  `sacct -l -j <jobid> -o jobname,NTasks,nodelist,MaxRSS,MaxVMSize...`

  - More flags can be found with `man sacct`
  - The output will be **very** wide. Use something like

    `sacct -l -j .......  | less -S`

    to view (makes it sideways scrollable, using the left/right arrow key)

Use `man sbatch`, `man srun`, `man ....` for more information

The Batch System (SLURM)

Job Output

- Output and errors in:
  `slurm-<job id>.out`
- Look at it with vi, nano, emacs, cat, less...
- To get output and error files split up, you can give these flags in the submit script:

  ```
  #SBATCH --error=job.%J.err
  #SBATCH --output=job.%J.out
  ```

The Batch System (SLURM)

Using different parts of Kebnekaise

- To run on the 'fat' nodes, add this flag to your script: `#SBATCH -p largemem` (Kebnekaise - largemem does not have general access)
- Specifying Intel Broadwell or Skylake CPUs only (Kebnekaise):
  `#SBATCH --constraint=broadwell`
  or
  `#SBATCH --constraint=skylake`
- Using the GPU nodes (Kebnekaise
  `#SBATCH --gres=gpu:<type-of-card>:x` where <type-of-card> is either k80 or v100 and x = 1, 2, or 4 (4 only for the K80 type).

More on
https://www.hpc2n.umu.se/documentation/guides/using_kebnekaise

The Batch System (SLURM)

Simple example, serial

Example: Serial job on Kebnekaise, compiler toolchain 'foss'

```
#!/bin/bash
# Project id - change to your own after the course!
#SBATCH -A SNIC2019-5-172
# Asking for 1 core
#SBATCH -n 1
# Asking for a walltime of 5 min
#SBATCH --time=00:05:00

# Purge modules before loading new ones in a script.
ml purge
ml foss/2019a

./my_serial_program
```

Submit with:

```
sbatch <jobscript>
```

The Batch System (SLURM)

Example, MPI C program

```
#include <stdio.h>
#include <mpi.h>

int main (int argc, char *argv[])

int myrank, size;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

printf("Processor %d of %d:  Hello World!\n", myrank,
size);

MPI_Finalize();
```

The Batch System (SLURM)

Simple example, parallel

Example: MPI job on Kebnekaise, compiler toolchain 'foss'

```
#!/bin/bash
#SBATCH -A SNIC2019-5-172
#SBATCH -n 14
#SBATCH --time=00:05:00
##SBATCH --exclusive
#SBATCH --reservation=intro-cpu

module purge
ml foss/2019a

srun ./my_parallel_program
```

The Batch System (SLURM)

Simple example, output

Example: Output from a MPI job on Kebnekaise, run on 14 cores (one NUMA island)

```
b-an01 [~/pfs/slurm]$ cat slurm-15952.out

The following modules were not unloaded:
   (Use "module --force purge" to unload all):

  1) systemdefault   2) snicenvironment
Processor 12 of 14: Hello World!
Processor 5 of 14: Hello World!
Processor 9 of 14: Hello World!
Processor 4 of 14: Hello World!
Processor 11 of 14: Hello World!
Processor 13 of 14: Hello World!
Processor 0 of 14: Hello World!
Processor 1 of 14: Hello World!
Processor 2 of 14: Hello World!
Processor 3 of 14: Hello World!
Processor 6 of 14: Hello World!
Processor 7 of 14: Hello World!
Processor 8 of 14: Hello World!
Processor 10 of 14: Hello World!
```

The Batch System (SLURM)

Starting more than one serial job in the same submit file

```
#!/bin/bash
#SBATCH -A SNIC2019-5-172
#SBATCH -n 5
#SBATCH --time=00:15:00

module purge
ml foss/2018b

srun -n 1 ./job1.batch &
srun -n 1 ./job2.batch &
srun -n 1 ./job3.batch &
srun -n 1 ./job4.batch &
srun -n 1 ./job5.batch
```

The Batch System (SLURM)
Multiple Parallel Jobs Sequentially

```
#!/bin/bash
#SBATCH -A SNIC2019-5-172
#SBATCH -n 14
# Remember to ask for enough time for all jobs to complete
#SBATCH --time=02:00:00

module purge
ml foss/2019a

# Here 14 tasks with 2 cores per task.  Output to file.
# Not needed if your job creates output in a file
# I also copy the output somewhere else and then run
# another executable...

srun -n 14 -c 2 ./a.out > myoutput1 2>&1
cp myoutput1 /pfs/nobackup/home/u/username/mydatadir
srun -n 14 -c 2 ./b.out > myoutput2 2>&1
cp myoutput2 /pfs/nobackup/home/u/username/mydatadir
srun -n 14 -c 2 ./c.out > myoutput3 2>&1
cp myoutput3 /pfs/nobackup/home/u/username/mydatadir
...
```

The Batch System (SLURM)

Multiple Parallel Jobs Simultaneously

Make sure you ask for enough cores that all jobs can run at the same time, and have enough memory. Of course, this will also work for serial jobs - just remove the srun from the command line.

```
#!/bin/bash
#SBATCH -A SNIC2019-5-172
# Total number of cores the jobs need
#SBATCH -n 56
# Remember to ask for enough time for all of the jobs to
# complete, even the longest
#SBATCH --time=02:00:00

module purge
ml foss/2018b

srun -n 14 --cpu_bind=cores ./a.out &
srun -n 28 --cpu_bind=cores ./b.out &
srun -n 14 --cpu_bind=cores ./c.out &
...
wait
```

Questions and support

Questions? Now: Ask me or one of the other support or application experts present.

OR

- Documentation: `https://www.hpc2n.umu.se/support`
- Support questions to: `https://supr.snic.se/support/` or `support@hpc2n.umu.se`