Introduction to High-Performance Computing

Session 03

Basic Cluster Usage II:

Environment: File Systems, Modules, Compiler and Toolchains

HPC User Environment

the user environment on a HPC cluster consists of:

- the operating system (OS)
  - e.g. RHEL Linux (all HPC systems in top500 have Linux-like OS)
  - basic functionality (login, create and edit files, …)
- data storage
  - one or more file systems
  - temporary, short and long term storage
- software
  - scientific applications
  - libraries
  - compiler
- job scheduler

File Systems

HPC File Systems

http://www.fz-juelich.de/ias/jsc/EN/Expertise/Datamanagement/JUDAC/Filesystems/filesystems_node.html

- typically on a HPC system different file systems are available

Name	Description	Features
$TMPDIR or /scratch	temporary storage provided on a per job basis, deleted after job often local disk or similar	very fast I/O, up to a few TB, no backup
$WORK	temporary storage for job data, maybe kept after job, typically parallel file system attached to interconnect	fast, parallel I/O, up to PB, no backup
$DATA	mid-term storage for job output, parallel filesystem or NFS	up to PB, maybe with backup
$HOME	NFS storage, long term and secure, for program codes, initial conditions	few 100GB, full backup, snapshots
$ARCH	permanent archive, storage for finished projects, tape library	few PB, possible slow read

File Systems

http://wiki.hpcuser.uni-oldenburg.de/index.php?title=File_system_and_Data_Management

- central Enterprise Spectrum Scale storage (ESS)
  - used for HOME, DATA, GROUP and OFFSITE directories
  - NFS mounted over 2x 10Gb Ethernet
  - full backup and snapshot functionality
  - can be mounted on local workstation using SMB
- shared parallel storage (GPFS)
  - used for WORK directory only
  - data transfer over FDR Infiniband
  - no backup
  - can also be mounted on local workstation using SMB
- local disks or SSDs for scratch
  - CARL compute nodes have local storage (1-2TB per node)
  - EDDY compute nodes have 1GB RAM disk (for compatibility)
  - usable during job run time

Directory Structure

- on every filesystem (**$HOME**, **$DATA**, **$WORK**) users will have their own subdirectory
  - e.g. for **$HOME**

```
drwx------    abcd1234      agsomegroup   /user/abcd1234
```

  - default permissions prevent other users from seeing the contents of their directory
  - user can give permissions to others to access files or subdirectory as needed (user‘s responsibility)
  - file and directory access can be based on primary (the working group) and secondary (e.g. the institute) Unix groups
  - recommendation: keep access restricted on **$HOME** and if needed share files/dirs. on **$DATA** or **$WORK**

https://wiki.hpcuser.uni-oldenburg.de/index.php?title=File_system_and_Data_Management#Managing_access_rights_of_your_folders

File Systems

File System	Env. Variable	Path	Used for
Home	`$HOME`	`/user/abcd1234`	critical data that cannot easily be reproduced (program codes, initial conditions, results from data analysis)
Data	`$DATA`	`/nfs/data/abcd1234`	important data from simulations for on-going analysis and mid term (project duration) storage
Work	`$WORK`	`/gss/work/abcd1234`	data storage for simulation runtime, pre- and post-processing, short term (weeks) storage
Scratch	`$TMPDIR`	`/scratch/<job-dir>`	temporary data storage during job runtime
Offsite	`$OFFSITE`	`/nfs/offsite/user/abcd1234`	long term storage for inactive data, only available on login nodes

- HOME, DATA and OFFSITE have backup for disaster recovery and daily snapshots for file recovery
- special quota rule for WORK

Quotas

https://wiki.hpcuser.uni-oldenburg.de/index.php?title=File_system_and_Data_Management#Quotas

- on every file system default quotas are in place
  - HOME, DATA and OFFSITE have 1TB, 20TB and 25TB, respectively
  - WORK has 50TB
  - maybe increased upon request (if resources are available)

- special quota on WORK
  - in addition to hard limit above, WORK also has soft quota of 25TB
  - if usage is over soft quota a grace period of 30 days is triggered
  - after grace period no data can be written to WORK by user

➔ clean up your data on work regularly

Group Directories

- group directories are available upon request
  - storage on the ESS
  - can be mounted via SMB (only version 2 or better)
  - path: **`$GROUP`** or **`/nfs/group/agyourgroup`**
  - should be used for data shared among members of the same group, in particular to avoid multiple copies of the same file
  - group leader is owner of directory
  - default rights are set so that anyone in group can read and write to group directory

File System Shares

https://wiki.hpcuser.uni-oldenburg.de/index.php?title=Local_Mounting_of_File_Systems

- you can mount your **$HOME**, **$DATA** and **$WORK** as well as **$OFFSITE** and **$GROUP** directories on your local workstation
- server address for mounting are

$HOME	**//smb.uni-oldenburg.de/hpc_home**
$DATA	**//smb.uni-oldenburg.de/hpc_data**
$WORK	**//smb.hpc.uni-oldenburg.de/hpc_work**
$OFFSITE	**//smb.uni-oldenburg.de/hpc_offsite**
$GROUP	**//smb.uni-oldenburg.de/<groupname>**

  - for Windows connect a network drive (and replace “/” with “\”)
  - for Linux add information in **/etc/fstab**

File System Use

- applications with high I/O demands can put a lot of stress on the used file system
- I/O-performance depends on the I/O profile
  - I/O with few but large files is better than many small files
  - sequential I/O is better than random access
- pick the right file system for your I/O profile
  - local disks or SSDs are best for I/O with small block sizes
  - parallel files system (WORK) is best for large files and parallel I/O
  - HOME and DATA (and all NFS mounted directories) should be avoided for I/O at runtime

simple I/O performance tests can be done with **dd**

https://www.thomas-krenn.com/de/wiki/Linux_I/O_Performance_Tests_mit_dd

Best Practices for File System Use

- if your job is doing heavy I/O use $WORK or $TMPDIR
  - I/O bandwidth to $WORK is >10GB/s (shared for the whole cluster), compared to 100MB/s at most to $HOME and $DATA
  - try to use parallel I/O and avoid using many small files
  - $TMPDIR is best for small files and random access (in particular on the bignodes)

- keep your data on $WORK while it is being processed
  - data that is currently not needed can be moved to $DATA
  - consider creating compressed archives and organise your data
  - only keep important data and delete as much as possible when a project is finished
  - use $GROUP if you frequently need to share data within your group to avoid unneccessary copies of data

Final Remarks File Systems

- setting file permissions
  - add execute (x) permission to directories to allow cd
  - add read (r) permission to directories to all ls
  - avoid adding write (w) permission for group or others on directories (you cannot change ownership of files)

- checking quotas
  - use the **`lastquota`** command to find out how much diskspace your are using
  - also weekly e-mails to all users

Software and Modules

Software

- software is installed centrally on the cluster
  - /cm/shared/uniol/software
  - user can use preinstalled software
  - software can be optimized for system
  - own software can be installed too

- installed software includes
  - compilers
  - libraries (MPI, numerical libraries,…)
  - scientific application
  - overview and help in the HPC wiki

Modules

- Linux settings are defined by environment variables

```
$ echo $HOME                # home directory
/user/lees4820
$ echo $PATH                # where to look for applications
/cm/shared/apps/slurm/current/sbin:/cm/shared/apps/slurm/
current/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/
sbin:/opt/ibutils/bin:/user/lees4820/.local/bin:/user/lee
s4820/bin
$ env                       # full list
HOSTNAME=hpcl002
TERM=xterm
. . .
```

  - applications require correct settings of environment variables

Modules

- the environment settings for installed applications are managed using modules

```
$ module list          # show loaded modules
Currently Loaded Modules:
  1) slurm/current   2) hpc-uniol-env

$ module av            # show available modules

------------- /cm/shared/uniol/modules/core -------------------
   hpc-uniol-env (L)  slurm/current (L)

------------- /cm/shared/uniol/modules/bio --------------------
   BCFtools/1.3.1     CD-HIT/4.6.4          SOAPdenovo2/r240
   BEDTools/2.26.0    FASTX-Toolkit/0.0.14  Stacks/1.42
. . .
```

Module Commands

https://wiki.hpcuser.uni-oldenburg.de/index.php?title=User_environment_-_The_usage_of_module_2016

- find modules

```
module available [module-name]
module spider [module-name]
```

  - list all modules [with given module name]
  - spider is case-insensitive and understands reg-exp

- load/unload

```
module load <module-name>
module remove <module-name>
```

  - to return to a default state

```
module restore
```

- information about modules

```
module list
module help <module-name>
module spider <module-name>
```

Examples: Module Commands

```
$ module list
1) hpc-uniol-env  2) slurm/current
$ module load GCC/4.9.4
$ module list
1) hpc-uniol-env  2) slurm/current  3) GCC/4.9.4
4) …
$ module swap GCC/4.9.4 GCC/5.4.0
$ module restore
$ module purge
$ module load hpc-uniol-env
```

hpc-env Modules

- in the module core-section you can find a number of **hpc-env** modules

```
$ ml av
------------ /cm/shared/uniol/modules/core ---------------
  hpc-env/6.4    (D)    hpc-uniol-env
  hpc-env/8.1           hpc-uniol-new-env
  hpc-env/8.2    (L)
```

  - these modules provide some basic settings (e.g. **$DATA**, loading the Slurm module) and make a specific module stack available
  - the version corresponds to a specific GCC version and all modules in the stack are based on this GCC version
  - the non-version modules are older and not based on a specific GCC
  - most software is installed in **hpc-uniol-env** and **hpc-env/6.4**
  - if you login you will find **hpc-uniol-env** loaded, this can be changed (e.g. with **module save**)
  - only one **hpc-env** module can be loaded at any time

Modules

- why use modules
  - modules allows multiple versions of the same application to be installed
  - modules change all the environment settings as needed
  - modules know about dependencies and conflicts

- modules and jobs
  - modules have to be loaded within a job script (as needed)
  - modules loaded when the job is submitted are remembered by SLURM
    (but you may submit a job later again with different modules loaded)

Compiler, Libraries and Toolchains

Compiler

- different compilers available (from vendors and also open-source)

```
---------- /cm/shared/uniol/modules/compiler -----------
   CUDA-Toolkit/8.0.44              NAG_Fortran/5.2
   GCC/4.9.4-2.25                   PGI/12.10
   GCC/5.4.0-2.26                   PGI/15.10
   GCC/6.2.0-2.27           (D)     PGI/16.10          (D)
   LLVM/3.8.1-goolf-5.2.01          icc/2016.3.210
   LLVM/3.8.1-intel-2016b           ifort/2016.3.210
   LLVM/3.9.0-intel-2016b   (D)
```

  - Intel compiler usally gives very good performance (icc and ifort)
  - using different compilers may help to better understand your code
  - some compiler support special hardware (e.g. GPUs by PGI)
  - always load one compiler (don‘t use OS GCC)

Example: RandomWalk.cpp

- download the code RandomWalk.cpp (and the other RandomWalk files) from Stud.IP
  - the code simulates a 2d random walk, each step of length one in random direction, prints out distance from start after N steps
  - expected distance is SQRT(N)
  - compile with GCC or ICS

```
$ g++ RandomWalk.cpp -o RandomWalk
```

or

```
$ icpc RandomWalk.cpp -o RandomWalk
```

  - run with one argument for seed, e.g.

```
$ ./RandomWalk 12345
```

  - timing with

```
$ time ./RandomWalk 12345
```

Libraries

- libraries are available as modules
  - numerical libraries provide optimized solutions of general problems

```
------------ /cm/shared/uniol/modules/numlib -------------
   ATLAS/3.10.2                  Octave/4.0.3
   Armadillo/7.500.1             OpenBLAS/0.2.19
   CLHEP/2.2.0.4-intel-2016b     Qhull/2015.2
   Eigen/3.2.9                   ScaLAPACK/2.0.2
   FFTW/3.3.5-gompi-5.2.01       SuiteSparse/4.5.3
   FIAT/1.6.0-intel-2016b        cuDNN/5.1-CUDA-8.0.44
   GMP/6.1.1    (D)              cvx/2.1
   GSL/2.1                       imkl/11.3.3.210
   Hypre/2.11.1                  leda/6.3
   LinBox/1.4.0                  maple/18
   MATLAB/2016b                  maple/2016            (D)
   MPFR/3.1.4                    stata/13
   NTL/9.8.1
```

Example: Matrix-Matrix Multiplication

- basic linear algebra is available in many different numerical libraries
  - OpenBLAS, Lapack, MKL, …
  - Basic Linear Algebra Subprograms (BLAS) contain e.g. a General Matrix Multiplication (gemm) of the form:

$$C = \alpha A \cdot B + \beta C$$

  - original version written in Fortran
  - used in the mm.cpp example (cblas_dgemm is the C-interface for double precision gemm)

```
// A, B, and C are objects of class SqMatrix but A[0] etc. are
// pointers to first element in matrix which is what dgemm expects
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            n, n, n, alpha, A[0], n, B[0], n, beta, C[0], n);
```

Toolchains

http://easybuild.readthedocs.io/en/latest/eb_list_toolchains.html

- some modules are called toolchains
  - provide a collection of compiler, MPI, and/or numerical libraries

```
------------ /cm/shared/uniol/modules/toolchain -----------------
 foss/2016b      gompi/5.2.01        iimpi/2013b    intel/2016b (D)
 gimpi/6.2016    gompi/6.2.01 (D)    iimpi/2016b (D)
 gompi/4.1.10    goolf/5.2.01        intel/2013b
```

- examples:
  - goolf: GCC, OpenMPI, OpenBLAS, ScaLAPACK, FFTW
  - foss: free and open source software (same as goolf currently)
  - gompi: GCC, OpenMPI
  - intel: Intel compilers, MPI, MKL

Example: Matrix-Matrix Multiplication

- the code mm.cp uses OpenBLAS which is included in the goolf-toolchain

```
$ ml restore
Resetting modules to system default
$ make clean
rm mm mm.o
$ make
g++ -O2  -c mm.cpp
mm.cpp:7:19: fatal error: cblas.h: No such file or directory
 #include "cblas.h"
                  ^
compilation terminated.
make: *** [mm.o] Error 1
$ ml foss
$ make
g++ -O2  -c mm.cpp
g++ -O2  -o mm mm.o -lopenblas
```

Example: Matrix-Matrix Multiplication

- alternatively the code can be compiled with Intel MKL
  - requires some code change (different header file)
  - requires changes to Makefile (different libraries to link)
  - result: code runs faster by 25%

```
$ sacct -j 2591679 -o JobID,JobName,Partition,Elapsed,MaxRSS,State,ExitCode
       JobID    JobName  Partition    Elapsed     MaxRSS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
2591679      run_mm.job     carl.p   00:06:21             COMPLETED      0:0
2591679.bat+      batch               00:06:21      7336K  COMPLETED      0:0
2591679.0            mm               00:00:33     37600K  COMPLETED      0:0
2591679.1            mm               00:00:32    113412K  COMPLETED      0:0
2591679.2            mm               00:00:33    412420K  COMPLETED      0:0
2591679.3            mm               00:00:32   1592064K  COMPLETED      0:0
2591679.4            mm               00:04:09   6310656K  COMPLETED      0:0
```

Job Arrays

Job Arrays

- task: run program isprime several (M) times
  - different input parameter every time
  - all input parameters are in file parameter.dat
  - simple approach: make M copies of job script, modifiy the input parameter in every file
  - clever approach: task array as prepared in prime.job
  - analysis after job array is completed e.g. with awk script

Job Arrays

- job or task arrays are defined by Slurm options

```
$ cat array.job
. . .
### settings for job array
#SBATCH --array 1-10:1%4     # define task array
                             # format range:step%tasklimit
. . .
```

  - range of tasks can be defined as from-to:increment
  - multiple ranges with comma-separated list
  - limiting the number of parallel tasks possible (if tasks have large requirements)

Job Arrays

- additional variable with task-ID is provided

```
$ cat prime.job
. . .
# get paramter from file for each task
parameter=$(awk "NR==$SLURM_ARRAY_TASK_ID {print $1}" parameter.dat)
echo -n "Task $SLURM_ARRAY_TASK_ID tested if $parameter is prime? "
./isprime $parameter
. . .
```

can be used

- e.g. to number input or output files
- computation in bash (limited)
- read specific line from input file

Job Arrays

- job array are a powerful tool for task parallel jobs
  - to be preferred over submitting many individual jobs
  - each tasks in a job array should be sufficiently long (e.g. > 1h), due to the overhead for a single task
- requires some strategy for post-processing
  - often Linux tools can do the trick, more complex tasks may require post-processing script in e.g. Python
- additional environment variables for first and last task
  - however, tasks may not complete in the correct order
  - alternatively job dependencies can be used

awk

http://www.gnu.org/software/gawk/manual/gawk.html

- powerful Linux tool that searches the lines of a file for patterns and performs an action on that line
  - similar tools are grep (pattern matching) and sed (streaming edit)
  - works well with data files (tables)
  - uses a C-like syntax

- example: prime.awk
  - reads all output files from the job array (using cat to combine them)
  - counts yes and no answers
  - prints final result

Job Arrays and Alternatives

- task-parallel jobs can be handled in different ways (from simple to more complex approaches)
  - single job containing a loop, useful if individual tasks only run for a very short time
  - job array (as shown before), simple and native approach
  - single job using the **`parallel`** command, again useful if individual tasks are short-running
  - parallel (MPI-)program that implements a master-worker scheme, useful e.g. if results from tasks generate new tasks

also see https://wiki.hpcuser.uni-oldenburg.de/index.php?title=How_to_Manage_Many_Jobs

Exercises

Exercises

a. Try to compile and run the mm-code
   a. Try to use different toolchains
b. Try to run a job script for an application
   a. See next slide for specific example Orca
c. Try to run and compile the RandomWalk code
   a. Try different compilers
   b. Run multiple times as job array

Exercise: ORCA Job

- examples for using installed software on the cluster can be found in the HPC wiki
  - e.g. ORCA (chemistry) http://wiki.hpcuser.uni-oldenburg.de/index.php?title=ORCA_2016
  - download the files for serial runs and submit job
  - use ORCA 3.0.3
  - the job script is rather complex
    - module is loaded
    - files are copied to $TMPDIR
    - application is started from $TMPDIR
    - output is copied to $WORK

Exercise: RandomWalk

- task: run RandomWalk several (M=10) times to get the average distance after N steps from multiple runs
  - different seed every time
  - each run as SLURM job
  - write job script based on the example from the lecture

  - think how to analyse after jobs are completed
    - how to combine the output of M different jobs