Cluster introduction

Marcus Boden

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Burckhardtweg 4, 37077 Göttingen

Fon: +49 551 39-30000 gwdg@gwdg.de www.gwdg.de

Learning Objectives

- Describe the Architecture of the Scientific Compute Cluster
- Using the module system on the Cluster
- Compiling Software using Build Systems

Section 1

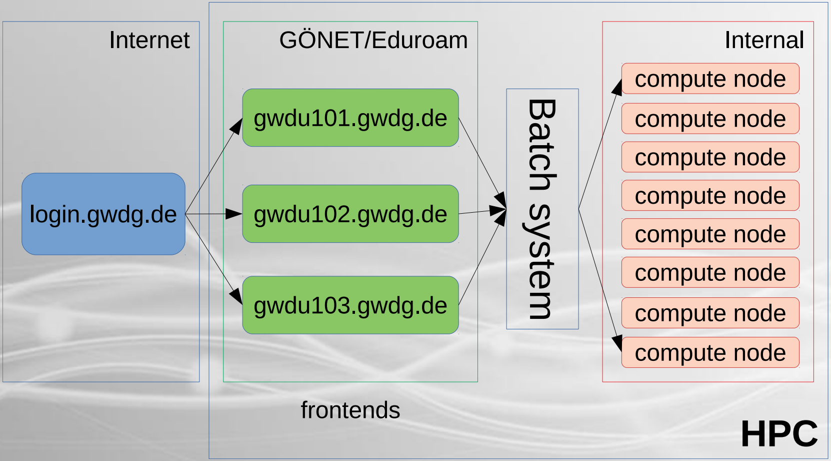
Cluster Overview

What is an HPC Cluster?

High-performance computing clusters usually have:

- a few frontends
  - ➥ for logging in and working interactively
- a lot of Nodes
  - ➥ this is where the work gets done
- one or more shared file system
  - ➥ High performance I/O
  - ➥ Parallel writing
- a high speed network
  - ➥ Inter-node communication
  - ➥ Storage access

Hardware and Network



Structure

Two sites:

- Modular Data Center (MDC)
  - ➥ Frontends: login-mdc.hpc.gwdg.de (gwdu101 and gwdu102)
  - ➥ Nodes: agqXXX, agtXXX, ampXXX
  - ➥ Intel Cascade Lake
  - ➥ access to /scratch1
- Fassberg
  - ➥ Frontend: login-fas.hpc.gwdg.de (gwdu103)
  - ➥ Nodes: dfaXXX, dmpXXX, dgeXXX, dteXXX
  - ➥ Intel Broadwell
  - ➥ access to /scratch2

Filesystem

2 file systems

1. **HOME** file system
2. **SCRATCH** file system

HOME

- Stores your *permanent* data.
- There is a quota. It could be extended on request.
- Has a backup mechanism.

SCRATCH

- Stores your data used for computations or projects.
- Fast and large file system.
- No Quota, but also no backup.

Filesystem Quotas

HOME

- Quota is set per user basis.
- Quota command displays current limits

```
gwdu101:14 11:55:41 ~ > Quota

Global Filesystem KBytes: used softlimit hardlimit ...
UNI11                   370216         0         0
UNI05                 65316256 104857600 419430400
```

SCRATCH

- No Quota per user. However, storage is limited.

```
gwdu101:45 10:52:46 ~ > df -h /scratch
Filesystem       Size  Used Avail Use% Mounted on
beegfs_nodev     1.6P  750T  796T  49% /scratch1
```

Filesystem /local

- **local** file system is NOT shared, but fast (SSDs).
- Use it for temporal data on every node
- The size of it rather small

```
bash-4.2$ df -h /local
Filesystem          Size  Used Avail Use% Mounted on
/dev/sda6            78G   57M   74G   1% /local
```

Data archiving

Archive location

- Personal archive is located at `/usr/users/a/USERNAME`
- You can get the path from `$AHOME` variable

Usage

- It is necessary to compress directories as tar or zip files
- if you want to archive directory `data`, call

```
tar -czvf $AHOME/data.tgz data
```

or faster (uses 4 cores and faster compression)

```
PIGZ="-1 -p 4 -R" tar -I pigz -cvf $AHOME/data.tgz data
```

Exercises

- connect to the frontends
- check your HOME quota
- check out both scratch file systems. How big are they, how much space is currently available?
- You downloaded a large genome database (100GB) from NCBI. Where would you store it and why?
- use scratch and archive:
  - Create a project directory on scratch
  - Add some files in it (e.g. `date > file1.txt`)
  - Compress the folder and send to archive

Time: 10 minutes

The workflow with /scratch file system

Important

The Scratch file system is **NOT** a permanent storage

Recommended workflow

- Create directory for your project
  `/scratch/users/$USER/PROJECT`
- Copy all necessary data there
- Run your compute jobs
- After completion of your jobs, save important results, that you need for further work to your home directory
- Delete all temporary files and broken runs
- Move the rest of the directory, that you want to keep for reference, into the archive and delete it from Scratch

```
tar -czvf $AHOME/PRJ.tar.xz /scratch/users/$USER/PROJECT
rm -rf /scratch/users/$USER/PROJECT
```

Data transfer

There are 2 transfer servers that can be used to transfer data from your machine to HPC.

transfer.gwdg.de

- reachable from the Internet
- only `HOME` is mounted

transfer-mdc.hpc.gwdg.de

- reachable only from GÖNET
- `HOME` and `/scratch1` are available

transfer-fas.hpc.gwdg.de

- reachable only from GÖNET
- `HOME` and `/scratch2` are available

Data transfer. Usage

SCP

works on Linux, macOS, and latest Windows

```
scp -rp {SRC-DIR} {USER}@transfer.gwdg.de:{DST-DIR}
```

to transfer back, simply swap the arguments

```
scp -rp {USER}@transfer.gwdg.de:{SRC-DIR} {DST-DIR}
```

Filezilla

works on all platforms. GUI. Open source software.

Rsync

works on Linux, macOS

```
rsync -avvH {SRC-DIR} {USER}@transfer.gwdg.de:{DST-DIR}
```

to transfer back, simply swap the arguments

```
rsync -avvH {USER}@transfer.gwdg.de:{SRC-DIR} {DST-DIR}
```

Section 2

Modules and Containers

The modules system

Problem:

- HPC Systems have a complex software ecosystem
  - ➥ different versions needed
  - ➥ complicated compiler requirements
  - ➥ library dependencies
- Package manager (yum, apt, etc.) cannot satisfy these requirements
- Compilation can be complicated

Solution:

- We compile/install software as necessary
- Make the software available with “modules”

The modules system

- "`module avail`" find a list of installed modules
- "`module list`" list of currently loaded modules
- "`module load software/version`"
- "`module purge`" unload all modules
- "`module unload software`" unload a single module
- Most of the modules just append or prepend a path to PATH and MANPATH variables.
- Or set default variables to be found by compiler/configure scripts at compile time.

CPU architecture specific modules

- Software provided as modules are compiled for specific CPU architecture: Cascadelake or Haswell.
- Names of these modules are the same, the correct version is loaded depending on the node you(your jobs) are.
- If you compile your software for specific architecture, check the modules you are using with `module whatis` command. It contains the "Target".

```
> gwdu103 ~ > module whatis gromacs
> ...
> gromacs/2020.4       : Target : haswell
> gwdu101 ~ > module whatis gromacs
> ...
> gromacs/2020.4       : Target : cascadelake
```

Singularity containers

Singularity is the containerization system, just like Docker. However, we don't provide Docker in HPC for security reasons.

Usage

To load singularity use the modules

```
module load singularity
```

You can run either native Singularity or Docker images.

```
singularity run library://sylabsed/examples/lolcow
```

With Docker image

```
singularity run docker://godlovedc/lolcow
```

Some software packages provide Docker or Singularity images, if they do it will be easier to run them as containers.

Exercises

- Have a look at the available modules.
  - ➥ Load a module and see how your environment changes.
  - ➥ Log in and log out again. Are the modules still loaded?
- Run a singularity container.

Time: 5 Minutes

Section 3

Compiling Software

Why Compiling?

- Compiling means to create an executable – or a library – from the source code
- GWDG cannot install all software required by users (see modules for what is available)
- Scientific software is often only available as source code
- Compiling on the target system often yields better performance
- Prepackaged software typically requires administrator (root) privileges ...

Recipe: wget and tar

Using wget and tar to prepare the source code

```
> mkdir $HOME/build
> cd $HOME/build
> wget <tarball URL>
> tar xvzf <name-version>.tar.gz
> cd <name-version>
```

Compiling (or “Building”) the Software

- Standard method: “`./configure; make; [make check; make install]`”
- Without root privileges: “`--prefix`” at configuration

About "`--prefix`"

- "`--prefix`" is used to specify the base directory for your software
- use "`./configure --prefix=DIR`" to install directly in DIR.
- e.g. "`./configure --prefix=$HOME/software/<name-version>`" to install into a software specific directory.

Recipe: Basic Building and Installing

Building and installing software into a specific directory

```
> cd $HOME; mkdir software
> cd $HOME/build/<name-version>
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
> ln -s $HOME/software/<name-version>/bin/* $HOME/bin
> ln -s $HOME/software/<name-version>/lib/* $HOME/lib
> ln -s $HOME/software/<name-version>/include/* $HOME/include
```

Compilers

- The GNU compilers (`gcc`, `gfortran`) are the standard compilers in Linux
- Other compilers are often faster, especially for Fortran code
- Recommended for overall performance: Intel compilers (`icc`, `ifort`)

Recipe: Using Intel Compilers

Building and installing software with Intel compilers

```
> module load intel
> CC=icc; CXX=icpc; FC=ifort; F77=ifort; F90=ifort
> export CC CXX FC F77 F90
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
```

Intel Math Kernel Library (MKL)

- A (shared) library is a collection of thematically related subroutines ready to use in a program
- The process of connecting a library to the (compiled) program is called linking
- Intel's Math Kernel Library provides performance optimized linear algebra and Fourier transform functions

Recipe: Using the MKL

Example: linking programs to MKL

```
> module load intel
> CC=icc; CXX=icpc; FC=ifort; F77=ifort; F90=ifort
> export CC CXX FC F77 F90
> module load intel-parallel-studio
> export CPPFLAGS="-I${MKLROOT}/include -I${MKLROOT}/include/fftw"
> export LDFLAGS="-L${MKLROOT}/lib/intel64 -lmkl_intel_lp64\
> -lmkl_sequential -lmkl_core -lpthread -lm"
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
```

Use Intel MKL Link Line Advisor!

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

Exercises: Compile your own editor!

In this exercise, you will download and compile the latest version of the nano editor

- Download the latest version of nano using `wget` or `curl`
- extract it using `tar`
- create a build directory
- configure the software with a prefix
- compile and install it
- test it!

Time: 10 Minutes