Using the GWDG Scientific Compute Cluster

by Azat Khuziyakhmetov and Marcus Boden

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Am Fassberg, 37077 Göttingen

Fon: 0551 201-1510 Fax: 0551 201-2150
gwdg@gwdg.de www.gwdg.de

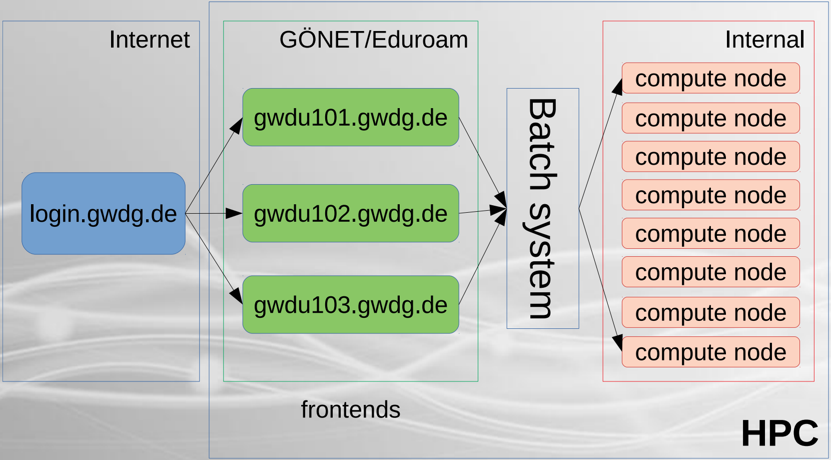
Outline

1. Connecting to the frontends
2. The most important Linux commands
3. Specifics of GWDG HPC cluster
4. Preparing the environment with “modules”
5. Compiling Software
6. Using Slurm - Basics
7. Using Slurm - Advanced
8. Getting Help

Section 1

Connecting to the frontends

Network


Internet
login.gwdg.de
GÖNET/Eduroam
gwdu101.gwdg.de
gwdu102.gwdg.de
gwdu103.gwdg.de
frontends
Batch system
Internal
compute node
compute node
compute node
compute node
compute node
compute node
compute node
compute node
HPC

Frontends

- gwdu101 and gwdu102: Cascade Lake Intel Silver 4214
  - ➥ processor features identical to agqXXX, agtXXX, ampXXX
  - ➥ new nodes in gpu and medium partitions
  - ➥ access to /scratch
- gwdu103: Broadwell Intel E5-2650 v4
  - ➥ processor features identical to dfaXXX, dmpXXX, dgeXXX, dteXXX
  - ➥ nodes in fat, medium and gpu partitions
  - ➥ access to /scratch2

Old frontends and HW update

Out of service from mid November 2020

- gwdu101: Abu-Dhabi AMD Opteron 6220
  - ➥ access to /scratch
- gwdu102: Sandy-Bridge Intel E5-2670 v1
  - ➥ processor features identical to gwddXXX
  - ➥ older nodes in medium-partition
  - ➥ access to /scratch

Further instructions about the hardware update will be sent via **hpc-announce** mailing list.

Accounts activated for HPC are subscribed automatically.

For those who have student accounts: you can subscribe to mailing list at `https://listserv.gwdg.de`.

ssh to the frontends

From the Internet connect to “login.gwdg.de” first in similar way as shown below. Afterwards to the frontend node.

You need SSH keys to connect to the cluster

- Linux or OS X:
  ssh gwdu101.gwdg.de -l {GWDG-USERID} -i {YOUR-KEY}
- Windows: in newer versions you can use native “ssh” in power shell or download *putty.exe* from `https://www.putty.org`
  - ➥ Run it. Enter “gwdu101.gwdg.de” in *hostname*
  - ➥ In the menu `SSH->Connection->Auth` select your private key and click open
  - ➥ Select “Yes” to trust the connection
  - ➥ Login as: {GWDG-USERID}

```
The authenticity of host 'gwdu101.gwdg.de (134.76.8.101)' can't...
ECDSA key fingerprint is SHA256:sIJNEepmILeEq/7Zqq4HCtpTM8L98ar...or
ECDSA key fingerprint is 7c:52:2b:17:f8:ba:29:bd:c5:45:d1:1a:9e...or
RSA key fingerprint is b9:f9:46:0f:23:c8:8d:76:b9:83:b9:1b:f6:5...or
ED25519 256 key fingerprint is e3:ef:39:f5:df:4f:c2:e2:c4:d0:28...
Are you sure you want to continue connecting (yes/no)?
```

Section 2

The most important Linux commands

Navigation

ls list the current directory you are in

cd change directory

Listing files and directories

- List the current directory you are in, “ls”
  - ➥ List the “hidden” files (beginning with “.”) too, “ls -a”
  - ➥ All files in an extended manner, “ls -la” or just type “l”
- Let’s look at three lines of the output

```
drwxrwxrwx   3 akhuziy users    4096  4. Apr 17:29 test
-rw-r--r--   1 akhuziy users     283 24. Sep 2019  Info.txt
lrwxrwxrwx   1 root root          23  Jul 22 12:10 passwd -> /etc/passwd
```

1	2	3	4	5	6	7	8	9	10

ten permission flags:

1 directory flag, “d”: directory, “-”: normal file, “l”: symlink

2,3,4 read, write, execute permission for **U**ser (Owner of the file)

5,6,7 read, write, execute permission for **G**roup

8,9,10 read, write, execute permission for **O**thers

Changing the language, what if I don’t undestand German

```
> echo $LANG
de_DE.UTF-8
> rm test
rm: reguläre leere Datei "test" entfernen?
```

```
> export LANG=en_US.UTF-8
> rm test
rm: remove regular empty file ‘test’?
```

For persistent English language, put it in your “.profile”:

```
echo ’export LANG=en_US.UTF-8’ >> ~/.profile
```

File operations, processes and file system

touch create file / update timestamps

other file ops `cp, rm, mv, mkdir, rmdir, ln`

htop display Linux processes

ps display current processes, imp. opt. a [all sessions], u [owner], x [all], w [wide], ww [even wider]

du display file space usage, `du -hs`

df display filesystem usage, `df -h, df -hl`

chmod

- Files attributes (mode bits) can be changed with `chmod`
- chmod can be used in two ways:
  - ➥ user friendly form:
    u (user) g (group) o (others) a (all)
    `chmod a+r {file}, chmod g=rwx,o+r {file}`
  - ➥ tell the mode bits:
    `chmod 744 {file}`

chmod (2)

- 0-7 are 3 bits: 111 → 7
- same order, like in dir listing: r,w,x

000 0 → `---` no read write or execute allowed

001 1 → `--x` (last bit is set)

010 2 → `-w-` (middle bit is set)

011 3 → `-wx` (last 2 bits are set)

100 4 → `r--` (first bit is set)

101 5 → `r-x` (first and last bits are set)

110 6 → `rw-` (first and second bits are set)

111 7 → `rwx` (all 3 bits are set)

`chmod 456`: owner - read; group - read and execute; others - read and write

chmod (3)

- In sum we have 9 bits now in 3 groups (user, group, others)
- But there is a 4th group: SUID/SGID/sticky-bits
- SUID/SGID means that the called program will run with the UID or GID of the owner
  - ➥ e.g. if the program owns root and has SUID set, you run the program as root
  - ➥ `chmod u+s {file}`, or `chmod g+s {file}`, `chmod a+s {file}` would set both
  - ➥ Since we are normal users on the system, this is very seldom needed.
- sticky-bit is more relevant for you, if you open a directory for colleagues to write (`chmod g=rwx {dir}`)
  - ➥ the stick-bit prevents others from deleting files, they do not own. (`chmod +t {dir}`)
  - ➥ e.g. if you create a file, others cannot delete it, even though they have write permission to the directory.

Editors, help and Bash

- **nano**, vi/vim, mcedit, joe

For most commands you can read the manual pages, just type “`man {COMMAND}`”.

The prompt is a so called “Shell” with built-in commands and functions. We are using the “bash”. Type “man bash” to get an impression about the power and flexibility of that shell.

Editors, help and Bash

- **nano**, vi/vim, mcedit, joe

For most commands you can read the manual pages, just type “`man {COMMAND}`”.
The prompt is a so called “Shell” with built-in commands and functions. We are using the “bash”. Type “man bash” to get an impression about the power and flexibility of that shell.

Environment variables

Where the system gets all the commands we learned today?

Bash searches all paths in the environment variable **PATH**.

```
gwdu101:84 15:03:22 ~ > echo -e ${PATH//:/:\\n}
/opt/slurm/bin:
/usr/lib64/qt-3.3/bin:
/usr/local/bin:
/usr/bin:
/usr/local/sbin:
/usr/sbin:
/sbin:
/usr/sbin:
/cm/local/apps/environment-modules/3.2.10/bin
```

The first Shell-Script

For our first Shell script we need additional information

- “`mktemp -d /scratch/${USER}/XXXXXXXX`” will create a unique directory, e.g. /scratch/akhuziy/XymeK4nq and echo it to stdout
- To store an output of a program in a variable, we write “`TEMPDIR=$(mktemp -d /scratch/${USER}/XXXXXXXX)`”

Let's write a little Shell script...

The first Shell-Script

For our first Shell script we need additional information

- "`mktemp -d /scratch/${USER}/XXXXXXXX`" will create a unique directory, e.g. /scratch/akhuziy/XymeK4nq and echo it to stdout
- To store an output of a program in a variable, we write "`TEMPDIR=$(mktemp -d /scratch/${USER}/XXXXXXXX)`"

Let's write a little Shell script...

Section 3

Specifics of GWDG HPC cluster

Filesystem

2 filesystems

1. **HOME** filesystem
2. **SCRATCH** filesystem

HOME

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

SCRATCH

- Stores your *temporal* data used for computations or projects.
- Fast and large filesystem.
- No Quota, but there are some rules to use it.

Filesystem Quotas

HOME

- Quota is set per user basis.
- Find it out using Quota command

```
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

```
gwdu101:14 11:55:47 ∼ > df -h /scratch
Filesystem      Size  Used Avail Use% Mounted on
beegfs_nodev    328T  227T  101T  70% /scratch
```

Filesystem /local

- **local** filesystem is NOT shared, but fast.
- On some nodes very fast because of SSD.
- 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 recommended to compress directories as tar files
- if you want to archive directory data, call

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

The workflow with /scratch filesystem

Important

The Scratch filesystem is **NOT** a permanent storage

Recommended workflow

- Create directory for your project `/scratch/USER-PROJECT`
- Copy all necessary data there
- After completion of your jobs for the project, move the directory into archive and delete it from Scratch

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

Try it!

- Create a project directory for this course:

```
mkdir /scratch/${USER}-scc-course
```

- Add some files in it

```
echo "a" > /scratch/${USER}-scc-course/file1
echo "b" > /scratch/${USER}-scc-course/file2
```

- Compress the folder and send to archive

```
tar -czvf $AHOME/scc-course.tar.xz /scratch/$USER-scc-course
```

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-scc.gwdg.de

- reachable only from GÖNET
- `HOME` and `/scratch` 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}
```

Screen. Resuming interactive sessions

Screen – is the utility which allows you to resume the sessions.

Usage

screen	starts a screen session
screen -S SName	starts a named screen session
screen -r SName	resume the screen SName
screen -ls	list all your available screens

within the screen you work as in usual shell
all screen commands start with `Ctrl + a`

Ctrl + a d	detach from a screen session
Ctrl + a c	create a new window
Ctrl + a 0	switch to window 0, or use another number

Section 4

Preparing the environment 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.

Section 5

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 …
  - ➥ (`sudo` or `su` won't work)
  - ➥ but you can use Singularity containers!

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/3.2.1
```

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.

Try it!

Getting and Unpacking the Source Code

- Source code is usually packaged as “tarball”
  - ➥ Look for file extensions “`tar.gz`”, “`tar.bz2`”, “`tgz`”
  - ➥ Naming convention is often `{NAME}-{VERSION}.tar.gz`
- If the tarball is available on the web use “`wget`” to download
- Use “`tar`” to unpack the tarball
  - ➥ Use “`tar xvzf`” for ‘`tar.gz`”, “`tgz`”
  - ➥ Use “`tar xvjf`” for “`tar.bz2`”

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
- For better performance: Use Intel compilers and MKL
- For MPI (distributed parallel) applications: Use Intel MPI

About "--prefix"

- "--prefix" is used to specify the base diretory 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`)
- Other compilers available at GWDG: PGI, Open64
  - ➥ For special cases and users willing to try several approaches for best performance

Recipe: Using Intel Compilers

Building and installing software with Intel compilers

```
> module load intel/compiler
> 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/compiler
> CC=icc; CXX=icpc; FC=ifort; F77=ifort; F90=ifort
> export CC CXX FC F77 F90
> module load intel/mkl
> 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

MPI programs

- MPI programs are meant to run distributed across several computers
- They require to be linked to an MPI library
- The recommended MPI library at GWDG is Intel MPI
- Others available are OpenMPI (tested), MVAPICH, and MVAPICH2

Recipe: Building MPI programs

Building MPI programs with Intel MPI

```
> module load intel/compiler
> module load intel/mpi
> CC=mpiicc; CXX=mpiicpc; FC=mpiifort; F77=mpiifort; F90=mpiifort
> export CC CXX FC F77 F90
> module load intel/mkl
> 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
```

Recipe: Building Rmpi for R

Preparation

```
> module load openmpi/gcc
> export OMPI_MCA_mtl=^psm
> echo $MPI_HOME
/cm/shared/apps/openmpi/gcc/64/1.10.1
> R
```

R command line

```
> install.packages("Rmpi", dependencies=TRUE,
    configure.args=c("--with-mpi=/cm/shared/apps/openmpi/gcc/64/1.10.1"
    ))
> install.packages(c("foreach", "doMPI"))
```

Table of Contents, Part II

1. Connecting to the frontends
2. The most important Linux commands
3. Specifics of GWDG HPC cluster
4. Preparing the environment with “modules”
5. Compiling Software
6. Using Slurm - Basics
7. Using Slurm - Advanced
8. Getting Help

Section 6

Using Slurm - Basics

Getting started with Slurm

How to use the cluster

- Cluster divided into frontends and compute nodes
- Compute nodes to all calculations
- You cannot connect directly to the nodes
- You cannot run heavy calculations on the frontends

- So how do you use the compute nodes?

Use our scheduler: Slurm!

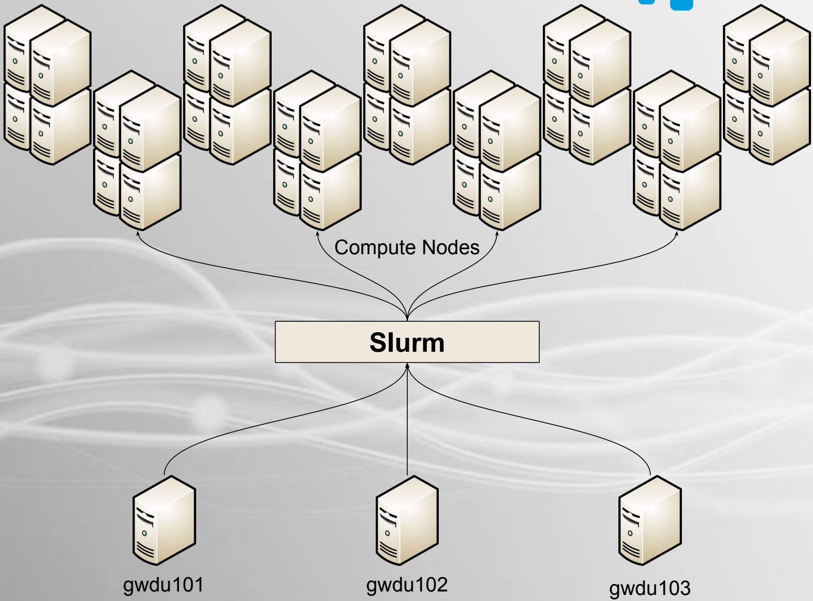
How to use the cluster

- Cluster divided into frontends and compute nodes
- Compute nodes to all calculations
- You cannot connect directly to the nodes
- You cannot run heavy calculations on the frontends

- So how do you use the compute nodes?

Use our scheduler: **Slurm!**

How to use the Cluster



Your first job

A job is a set of instructions for Slurm, including

- one or multiple programs to execute
- estimated runtime
- required resources (CPUs, GPUs, Memory)
- and more...

Your first job

Use srun to submit a job to slurm

```
    srun <program>
```

Example:

```
gwdu101:27 12:53:50 ~ > hostname
gwdu101
gwdu101:27 12:53:53 ~ > srun hostname
gwdd078
gwdu101:27 12:53:56 ~ > srun hostname -f
gwdd078.global.gwdg.cluster
```

Telling Slurm what to do

- `srun` submits information on your job to Slurm
  - ➥ What is to be done? (path to your program and required parameters)
  - ➥ What are its requirements? (e.g. which nodes, number of tasks, maximum runtime)
- Slurm matches the jobs requirements against the capabilities of our nodes
- When suitable free resources are found the job is started
- Slurm prioritizes the jobs based on a number of factors.

Partitions

- Different compute nodes have different features
- Slurm differentiates using **Partitions**

Available Partitions

General purpose partitions:

medium General purpose queue, well suited for large MPI jobs. Up to 1024 cores.

fat Up to 512 GB in one host.

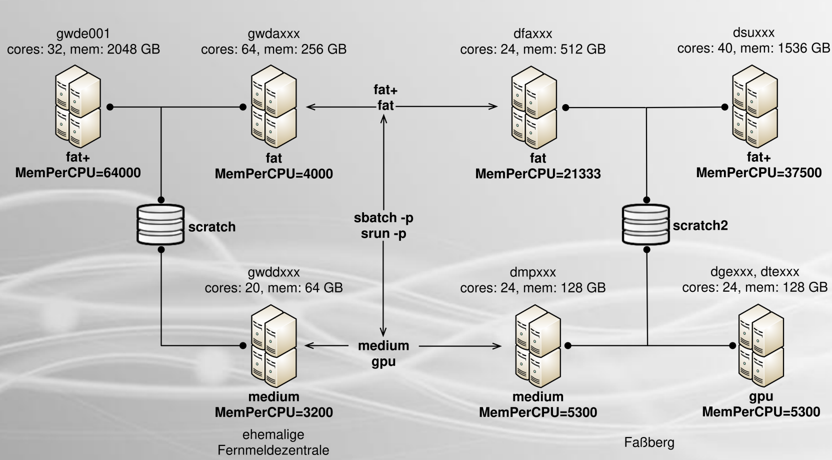
fat+ For extreme memory requirements. Up to 2048GB per host. Jobs need memory specifications.

Special purpose partitions:

gpu For jobs using GPU acceleration.

int For interactive jobs, i.e. jobs which require a shell or a GUI.

The GWDG Scientific Compute Cluster



Basic Concepts 1

Cluster A collection of networked computers intended to provide compute capabilities.

Node One of these computers, also called host or server.

frontend Special node provided to interact with the cluster. gwdu101, gwdu102, and gwdu103 in our case.

Job Program consisting of one or several parallel tasks.

Partition A group of nodes on which a job is intended to run

Batch System Management system distributing job tasks across job slots. We are changing from LSF to Slurm.

Submitting a job

```
srun <parameters> <program>
```

common parameters

-p <partition>	partition.
-t <hh:mm:ss>	Maximum runtime. If this is exceeded the job is killed.

Interactive Jobs

`srun`: Interactive jobs

--x11
: Adds X11 (GUI) forwarding. This requires that you connect to the frontend with `ssh -Y` and your local machine supports X-Windows.

-p int
: Use the interactive partition. In `int` the nodes have no slot limit. They will take jobs until their load crosses a specified threshold, so jobs start immediately.

--pty
: interactive mode

Interactive X11 Job

Running Matlab

```
> ssh -Y gwdu101.gwdg.de
> module load matlab/2015a
> srun --x11 -p medium matlab
```

- The job will be dispatched and as soon as an available node is found and the Matlab interface will start.
- If you have your own license for Matlab then you need to place your `license.lic` file in `$HOME/.matlab/R2015a_licenses` directory (dependent on the version you are using).

Interactive Console Job

Running R interactively

```
> ssh gwdu101.gwdg.de
> srun --pty -p medium R
```

Try it!

Basic Concepts 2

Serial job Job consisting of one task using one job slot.

SMP job Job with shared memory parallelization (often realized with OpenMP), meaning that all tasks need access to the memory of the same node. Consequently uses several job slots **on the same node**.

MPI job Job with distributed memory parallelization, realized with MPI. Can use several job slots on several nodes and needs to be started with a helper program, e.g., `mpirun` or `srun`.

Resource selection: CPU

`srun` options for parallel (SMP or MPI) jobs.

-N <min>-<max>, --nodes=<min>-<max>	Minimum and maximum node count. You can also specify the exact number.
-n,--ntasks=<n>	Number of tasks (not equally distributed!)
--tasks-per-node=<n>	Tasks per node. If used with `-n` it denotes the maximum number of tasks per node.
-c,--cpu-per-task=<n>	CPUs per tasks. Useful for hybrid jobs

A note on -n vs. -c

Rule of thumb

- `-c` for single node jobs
- `-n` for MPI jobs

Rule of thumb 2

If you are unsure if your program uses MPI, then it does not.

A note on -n vs. -c

Rule of thumb

- `-c` for single node jobs
- `-n` for MPI jobs

Rule of thumb 2

If you are unsure if your program uses MPI, then it does not.

Try it!

Execises

Try these job configurations

1. 10 processes
2. 10 processes distributed over 3 nodes
3. 3 nodes with 3 processes each
4. 1 process with 5 cores
5. 2 processes per node on 2 nodes with 4 cores per process

use `slurm-resources`-script to get see the resources of your job

Resource Selection: Memory

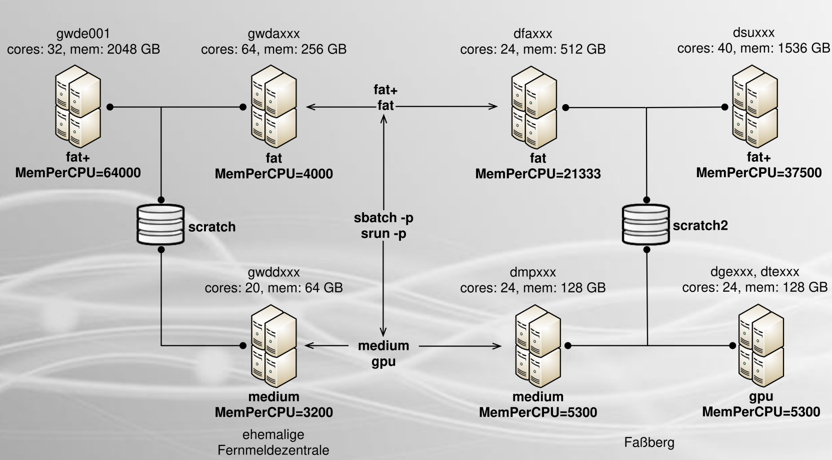
srun options

`--mem` <size[K|M|G|T] > Memory per node.

`--mem-per-cpu` <size[K|M|G|T] > Memory per task.

- without options:
  - each partition has a `DefMemPerCPU` option
  - can be retrieved via `scontrol show partition <name>`

The GWDG Scientific Compute Cluster



Try it!

Exercise:

Play with the combination of number of cores or tasks, nodes and their effect on your available memory:

1. 1 core and `--mem 4G`
2. 3 tasks and 2 nodes, see effect of `--mem` and `--mem-per-cpu`
3. 20 processes, see distribution of memory over hosts.

Non interactive Jobs

Problem

- if you have big jobs, you queue time will be long
- `srun` needs you to stay logged in
- jobs can run for days

Non interactive Jobs

Solution

```
sbatch <slurm options> jobscript
```

--mail-type=<TYPE> get mail notifications (type: BEGIN, END, etc.)

--mail-user=<address> Default: ${USER}@gwdg.de

-o/-e <file> Store job output in file (slurm-<jobid>.out by default). %J in the filename stands for the jobid.

sbatch: Using Job Scripts

A job script is a shell script with a special comment section.
The #SBATCH lines have to come first!

sbatch: Basic job script example

```
#!/bin/bash
#SBATCH -p medium
#SBATCH -t 10:00
#SBATCH -o job-%J.out

slurm_resources
```

Submit with:

```
sbatch <script name>
```

Jobscripts

- A job script is essentially a normal script
- usually bash/shell, but can be any scripting language (R, python, perl)
- `#SBATCH` lines need to be at the top!
- you can copy files, load modules, to scripting in them
- for MPI, use `srun` or `mpirun` to start your program

Recipe: Submitting an MPI job

Distributing tasks in the medium partition

```
#SBATCH -p medium
#SBATCH -n 240
#SBATCH -o job-%J.out

module purge
module load intel/compiler intel/mkl intel/mpi namd

srun namd2 +setcpuaffinity apoa1.namd
```

Recipe: Submitting an MPI job

Distributing tasks in the medium partition

```
#SBATCH -p medium
#SBATCH -N 10
#SBATCH --ntasks-per-node 24
#SBATCH -o job-%J.out

module purge
module load intel/compiler intel/mkl intel/mpi namd

srun namd2 +setcpuaffinity apoa1.namd
```

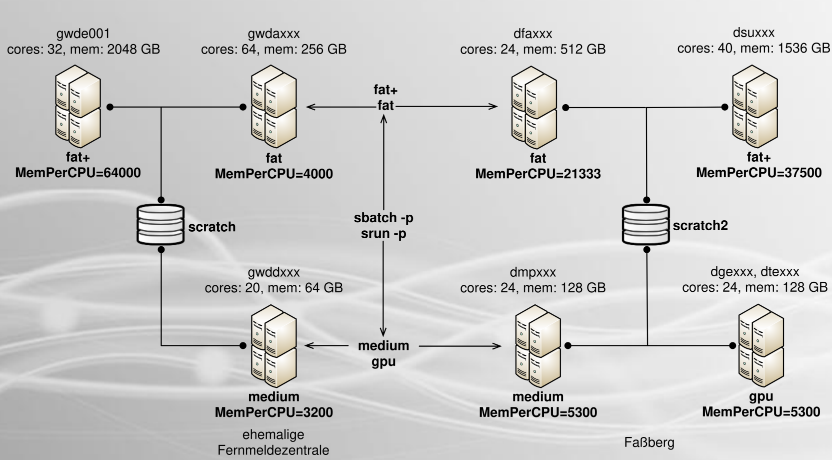
Job Disk Space Usage Options

/local
: Local hard disk of the node. SSD based on almost all nodes, therefore a very fast option for storing temporary data. Automatic file deletion. A temporary directory is created on all nodes at `$TMP_LOCAL`.

/scratch
: Shared scratch space, available on most nodes, but there are two instances (use `-C scratch` or `-C scratch2`). Very fast, no automatic file deletion, but also no backup! Files may have to be deleted manually when we run out of space.

$HOME
: Available everywhere, permanent, with backup. Personal disk space can be increased. Comparably slow.

The GWDG Scientific Compute Cluster



Recipe: Using /scratch

```
#!/bin/bash
#SBATCH -p fat
#SBATCH -n 64
#SBATCH -N 1
#SBATCH -C scratch
#SBATCH -t 1-00:00:00

export g09root="/usr/product/gaussian/g09/d01"
source $g09root/g09/bsd/g09.profile

MYSCRATCH=‘mktemp -d /scratch/${USER}/g09.XXXXXXXX‘
if [ ${MYSCRATCH} -a -d ${MYSCRATCH} ]; then
  export GAUSS_SCRDIR=${MYSCRATCH}
else
  export GAUSS_SCRDIR=/local
fi

g09 myjob.com myjob.log

if [ ${MYSCRATCH} -a -d ${MYSCRATCH} ]; then
  rm -rf ${MYSCRATCH};
fi
```

Try it!

Exercise

Write a job script, where you

- create a scratch directory
- copy data from your home file system to the scratch directory
- run a job with the data
- copy the results back
- delete the scratch directory

If you do not have a program/data to try this on, there is a small python program in `/scratch/scc-course/` and a bit of input data.

Exclusive jobs

- `#SBATCH --exclusive` in a job script denotes an exclusive job.
- An exclusive job uses all job slots (cores) of all its nodes.
- Using `--exclusive` together with `-N 1` reserves one complete node, independent of `-n`.
- You automatically get all the memory. Do not use `--mem` as that might limit you available memory.
- Disadvantage: You will have to wait until a whole node is free.

The fat+ partition

The `fat+` partition contains:

- 5 nodes with 1.5Tb Memory
- 1 node with 2Tb Memory

Usage recommendations:

- Work your way up. Start in `fat` and only use `fat+` if your jobs runs out of memory.
- Use `sacct` or `profit-hpc`, see if your job really is memory bound
- When unsure, ask us!
- `--mem`, `--mem-per-cpu` or `--exclusive` is mandatory
- You might get angry mails from me, if you waste resources here

Recipe: MPI jobs with `--exclusive`

Using exclusive jobs to get full nodes

```
#SBATCH -p medium
#SBATCH -N 4
#SBATCH --ntasks-per-node=4
#SBATCH -o job-%J.out
#SBATCH --exclusive

module purge
module load intel/compiler intel/mpi

srun big_mpi
```

Recipe: Combine shared memory and MPI

Running hybrid jobs

```
#SBATCH -p medium
#SBATCH -N 5
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=6
#SBATCH -o job-%J.out

module purge
module load openmpi/gcc

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun hybrid_job
```

Longer or shorter jobs

The --qos parameter

- Default maximum runtime: 2 days
- `--qos= <qos>` can select a QoS
- Two extra QoS available:

short for shorter jobs (max. 2h), has higher priority, limited job slots

long longer jobs (max. 5d), limited job slots.

But my job is even longer

- try parallelizing more
- break it down into smaller steps
- check, if your software supports checkpoints
- check again!
- contact us

Miscellaneous Slurm Commands

sinfo Info about the system and partitions.
`-p <partition>, -t <state>`

squeue Show the job queue.
`-p <partition>, --me`

scontrol show [partition|node|job] <x> where x should be a node name, jobID or partition name.

ssprio Priority information about pending jobs

sacct Get information about a job after it finished

```
-j <jobid>
--format=JobID,User,JobName,MaxRSS,Elapsed,Timelimit
```

`scancel`: Terminate your jobs

- Two use modes:
  1. `scancel <jobid>`: Kill job with specific jobid.
  2. `scancel <select options>`: Kill all jobs fitting the selection.
     Select option examples:
     - `-p <partition>`
     - `-u <$USER>`
     - `-s <state>`

Using the gpu partition

GPU parameters

-G | --gpus=[type:]<n> requests `n` GPUs of `type`

--gpus-per-task=[type:]<n> requests `n` GPUs of `type` per task

--gpus-per-node=[type:]<n> requests `n` GPUs of `type` per node

- CPUs are evenly distributed for every GPU
- Available types are:
  - gtx980
  - gtx1080
  - k40
- See: `sinfo -p gpu --format=%N,%G`

Debugging

- take a look at yoru output files, while the job is running:
  - ➥ tail -f /path/to/output
- take a look at the jobs, while it is running
  - ➥ you can `ssh` into every node that currently calculates your job
  - ➥ use `htop` to see the processor and ram usage

Debugging

Read the extra job information

```
======================================================================
JobID = 4383174
User = mboden, Account = admin
Partition = gpu, Nodelist = dge[001,006]
======================================================================
[job output]
============ Job Information =========================================
Submitted: 2020-04-24T17:35:41
Started: 2020-04-24T17:35:41
Ended: 2020-04-24T17:45:45
Elapsed: 10 min, Limit: 60 min, Difference: 50 min
CPUs: 2, Nodes: 2
============= ProfiT-HPC =============================================
To generate the ProfiT-HPC text report, run the following command
profit-hpc 4383174
======================================================================
```

Take a look at all the information. Is it as expected?

Debugging

Read your errors!

```
slurmstepd: error: Detected 1064 oom-kill event(s) in step XXXXXX.0 cgroup.
Some of your processes may have been killed by the cgroup out-of-memory handler.
srun: error: gwda024: task 3: Out Of Memory
```

Might have something to do with memory!
Have a look at your jobs memory with:

```
sacct -j JOBID -o jobid,MaxRSS,MaxRSSNode
```

And for more advanced job statistics, use profit-hpc

Section 7

Using Slurm - Advanced

Even more possibilities!

Job Arrays

Job arrays are a way to submit many similar jobs at one.

-a | --array=<n-m> creates a job array with indices n to m.

- control jobs via environment variables:
  - $SLURM_JOBID
  - $SLURM_ARRAY_JOB_ID
  - $SLURM_ARRAY_TASK_ID

Job array environment variables

```
sbatch --noinfo -a 1-3 array_test.sh
gwdu101:30 18:37:19 ~ > cat slurm-4383909_*
SLURM_ARRAY_JOB_ID=4383909
SLURM_ARRAY_TASK_ID=1
SLURM_JOBID=4383910
SLURM_ARRAY_JOB_ID=4383909
SLURM_ARRAY_TASK_ID=2
SLURM_JOBID=4383911
SLURM_ARRAY_JOB_ID=4383909
SLURM_ARRAY_TASK_ID=3
SLURM_JOBID=4383909
```

Job Dependencies

Wait for a specific job to finish, before the next starts:

```
-d | --dependency=dependency_definition
```

where `dependency_definition` can be:

after:job_id[+time] After the specified jobs start or are cancelled

afterok:job_id After the specified jobs have successfully executed

afternotok:job_id After the specified jobs have terminated in some failed state

afterany:job_id After the specified jobs have terminated.

Miscellaneous Slurm Options

might be helful?

--wrap= wrap the specified command string in a simple "sh" shell script. Only for `sbatch`.

--test-only Check script and give estimate when it would run.

--open-mode=append|truncate append or overwrite job files

--export=NONE don't export user environment, helpful for reproducibility.

--signal=B:12@600 Send signal 12 to job when 600 seconds before time limit. You can catch the signal in the script:

```
[...]
trap 'cp -af ${TMP_LOCAL}/* /scratch/your_dir/; exit 12' 12
your_job &
wait
```

General slurm advice

- use job arrays where possible (don't sbatch in a for loop)
- set a reasonable time limit (not just 2 days)
- use the short QOS where applicable
- ask us!

Recipe: Iterators in R

Using the foreach package

```
library(foreach)

ls<-foreach(i=1:100) %do% {
    norm=rnorm(100000)
    summ=summary(norm)
    summ
    }
ls
```

Recipe: Parallelization in R with doMPI

Using doMPI as backend for foreach

```
library(doMPI)

cl <- startMPIcluster()
registerDoMPI(cl)

ls<-foreach(i=1:100) %dopar% {
    norm=rnorm(100000)
    summ=summary(norm)
    summ
    }
ls

closeCluster(cl)
mpi.quit()
```

Recipe: MPI and R

Using R with doMPI in a batch job

```
#SBATCH -p medium
#SBATCH -n 20
#SBATCH -o job-%J.out

module load openmpi/gcc

srun Rscript "doMPI_script.R"
```

Task parallelization with GNU parallel

- GNU parallel distributes a set of tasks to a set of cores
- Requirement: No dependencies and side effects between tasks (*embarrassingly parallel*)

Using parallel to run a program with multiple input files

```
parallel 'cp {} .; g09 {/} {/.}.log' \
::: $(find /usr/product/gaussian/g09/tests -name *.com -type f)

parallel 'cp {} .; if (eval "g09 {/} {/.}.log");
then echo {/} >> ok; else echo {/} >> failed; fi' \
::: $(find /usr/product/gaussian/g09/tests -name *.com -type f)
```

Recipe: GNU parallel in a batch job

Multiple input files with parallel in a batch job

```
#!/bin/bash

#SBATCH -p medium
#SBATCH --qos=short
#SBATCH -c 20
#SBATCH -N 1
#SBATCH -t 02:00:00
#SBATCH -C scratch|scratch2

module load gaussian
mkdir /scratch/${USER}/g09_ptest
cd /scratch/${USER}/g09_ptest

parallel \
  ’cp {} .;
  if (eval "g09 {/} {/.}.log");
    then echo {/} >> ok;
    else echo {/} >> failed;
  fi’ \
::: $(find /usr/product/gaussian/g09/tests -name *.com -type f)
```

Section 8

Getting Help

Information sources

- man pages
- Slurm online help
  - ➥ For example: `sbatch --help`
- GWDG scientific compute cluster documentation
  - ➥ `https://info.gwdg.de/docs/doku.php?id=en:services:application_services:high_performance_computing:start`
- GWDG scientific compute cluster user wiki
  - ➥ `https://info.gwdg.de/wiki/doku.php?id=wiki:hpc:start`
- HPC announce mailing list
  - ➥ `https://listserv.gwdg.de/mailman/listinfo/hpc-announce`

Adding to the Wiki

- Everyone with a cluster account can add to the Wiki!
- Please inform us of all changes and new articles at parallel@gwdg.de.
- Please add the category "*Scientific Computing*" to all contributions regarding the cluster.

Using the GWDG Support Ticket System

- Write an email to *hpc@gwdg.de*
- State your user id (`$USER`)
- If you have a problem with jobs, **always** include:
  - ➥ Job IDs
  - ➥ standard output (`-o <file>`)
  - ➥ standard output (`-e <file>`)
- If you have a lot of failed jobs send at least two outputs. You may also list the jobid's of all failed jobs.
- If you don't mind us looking at your files, please state this in your request
  - ➥ You may limit your permission to specific directories or files

Digression: Directory Structure 1

- Convention: Executables are stored in "`bin`", shared libraries in "`lib`" directories
- Directories in "`$PATH`" are searched for binaries, directories in "`$LD_LIBRARY_PATH`" for libraries
- Two strategies:
  1. Put everything directly under `$HOME/bin`, `$HOME/lib`
     - Easy to setup search paths
     - Difficult to remove software packages
  2. Put each software into its own subdirectory
     - Easy to remove software (with "`rm -rf` <subdirectory>")
     - Difficult to setup search paths

Digression: Directory Structure 2

- Or combine both strategies:
  - ➥ Put each software in its own subdirectory
  - ➥ Use “`ln -s`” to link everything to $HOME/bin and $HOME/lib, respectively
  - ➥ Use “`export LD_LIBRARY_PATH=$HOME/lib:$LD_LIBRARY_PATH; export PATH=$HOME/bin:$PATH`” in your shell and scripts
  - ➥ Use “`find $HOME/bin $HOME/lib -xtype l -delete`” after removing software