technische universität
dortmund

LiDO3

First Contact

IT & Medien Centrum | CC HPC

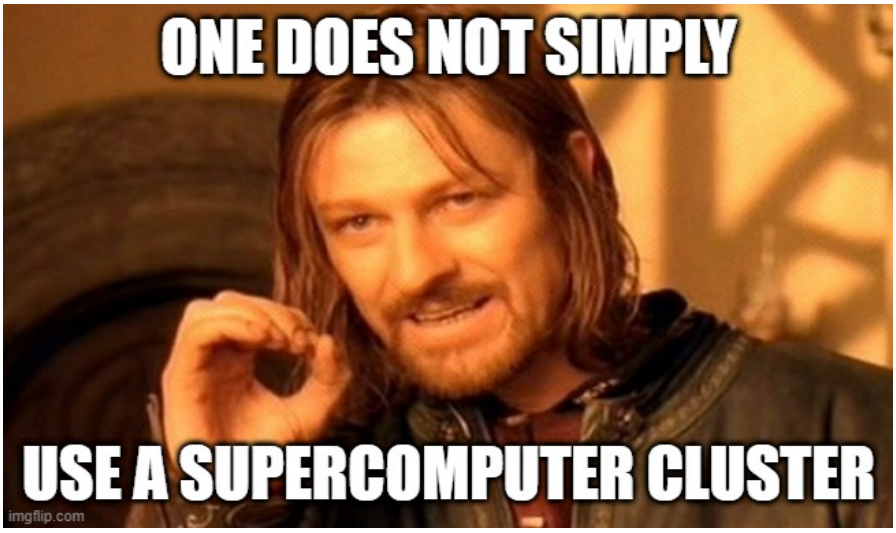
March 13, 2021

Table Of Contents

Chapter 1

LiDO3 - first contact

1.1 Introduction



You may have a laptop or PC with 8 to 16 cores, several terabytes of hard disk space and several gigabytes of main memory – or access to a comparably equipped server. Because you have determined that this equipment is not sufficient for the simulations you intend to run with your application, you were redirected to the HPC cluster LiDO3.

However, LiDO3 is not **one** particularly powerful and well-equipped server with thousands of cores, petabytes of hard disk space and terabytes of main memory. In other words, a scaled-up version of your own equipment. Instead it is more a scaled-out version: it consists of several hundred individual servers with an average equipment

like the laptop or PC or server mentioned at the beginning, plus a jointly usable hard disk space every one of these servers can use concurrently, connected via a special low-latency and fast network.

In this respect, the use of LiDO3 differs from the use of your laptop:

You can **not** simply run your application on one of the LiDO3 gateways. You could, but in this case the resources of the gateways (40 cores, 256 GiB RAM) would already be exhausted by the need of simulations of individual users! Attempts to perform large calculations on the gateways is not prevented from the outset, but is sanctioned in the case of discovery with the temporary blocking of the user account.

As – in contrast to your laptop/PC/server – several hundred scientists and students have access to LiDO3 and want to run potentially dozens to thousands of simulations simultaneously on LiDO3, there is a scheduler that attempts to broker between computational demand and the available computational resources. It distributes the simulations over the available compute nodes in a way that not several users share the same ressources (CPU/RAM) or, in the worst case, eat them away one another.

This scheduler is called Slurm[1]. It provides interactive and non-interactive (so-called batch jobs) sessions.

The interactive workflow differs from the way you are used to work on your laptop only by an additional command in advance (look for the `srun` command in the remainder of this document or any Slurm documentation). Be aware, though, that for an interactive Slurm session to work as intended, you do rely on a uninterrupted network connection to LiDO3 for the entire duration of the execution of your simulation! In addition, it may happen that sufficient resources for your interactive Slurm job become available only after some time of waiting (few hours or days of waiting time, depending on the size of your resource request). Possibly at an inconvenient time, e.g. at night at 2 a.m., a time you do not want to work or are not at your computer at all.

That is why the use of batch jobs for large production calculations is generally preferable.

If you happen to not unterstand how to use the Slurm scheduling system after reading this document, please contact us *beforehand*, thus we can clarify on how your application could be ported and executed correctly to LiDO3.

[1] https://en.wikipedia.org/wiki/Slurm

1.2 Scope

This document intends to guide you through the first steps on LiDO3, TU Dortmund's high performance cluster (HPC): to get access to the system and a job running.

We renounce the explicit mention of the female form and hope that this omission allows fluent reading of the instructions.

1.3 Non-scope

Programming, especially parallel programming and the usage of libraries like MPI[2] is not subject of this document. Neither is it a guide for structuring workload to scale on a HPC environment.

[2] https://en.wikipedia.org/wiki/Message_Passing_Interface

Chapter 2

Prerequisites

2.1 How do I get / extend a user account?

2.1.1 Application

Applications must be submitted by permanent employees of the *Technische Universität Dortmund* or the institution of the applicant.

In most cases, employees of the *Technische Universität Dortmund* can use the service portal[1] to submit an application online.

In this application form, it is mandatory to provide information about association (e. g. faculty and chair), the intended purpose of LiDO usage, termination date of LiDO usage, name and email address of your approver (your supervising professor) and your public SSH key which you are supposed to have generated before submitting the form. For generating your public and private key pair see page 12.

The LiDO application form is not visible for students without a student assistant contract ("Hiwi-Vertrag") nor for external PhD students. If such a student is supposed to use the LiDO cluster, his supervisor must submit the form and is encouraged to use the text box labelled "Additional information" to add the remark that the account application is actually for a student, supplying additionally the student's real name, login name, email address and phone number.

[1] https://service.tu-dortmund.de/group/intra/lido3neuantrag

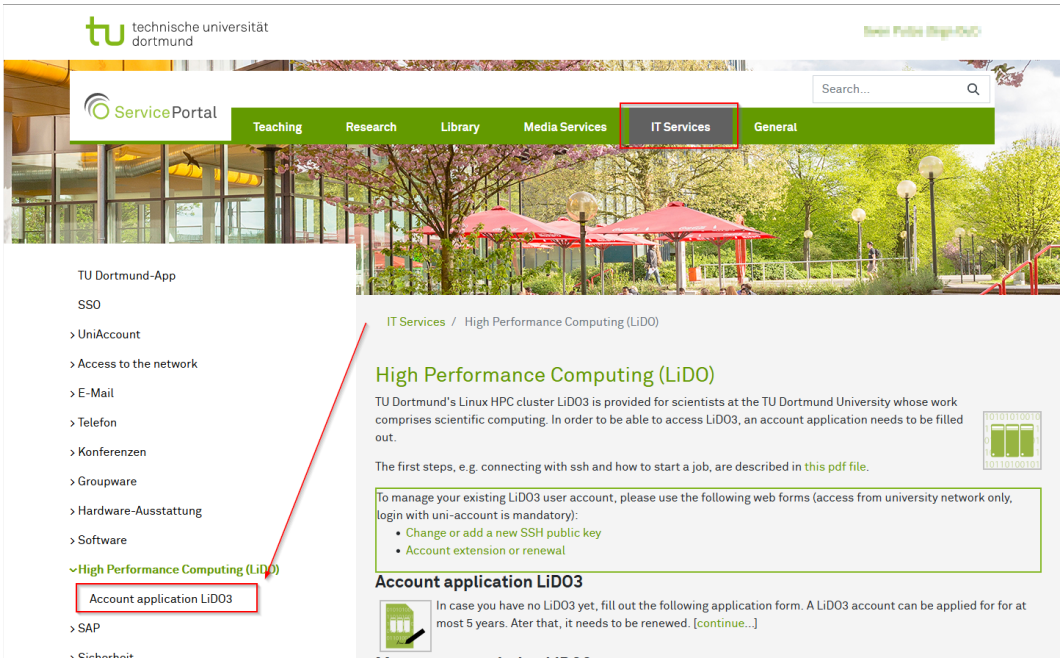


Figure 2.1: Use *"Neuantrag"* to submit the application.

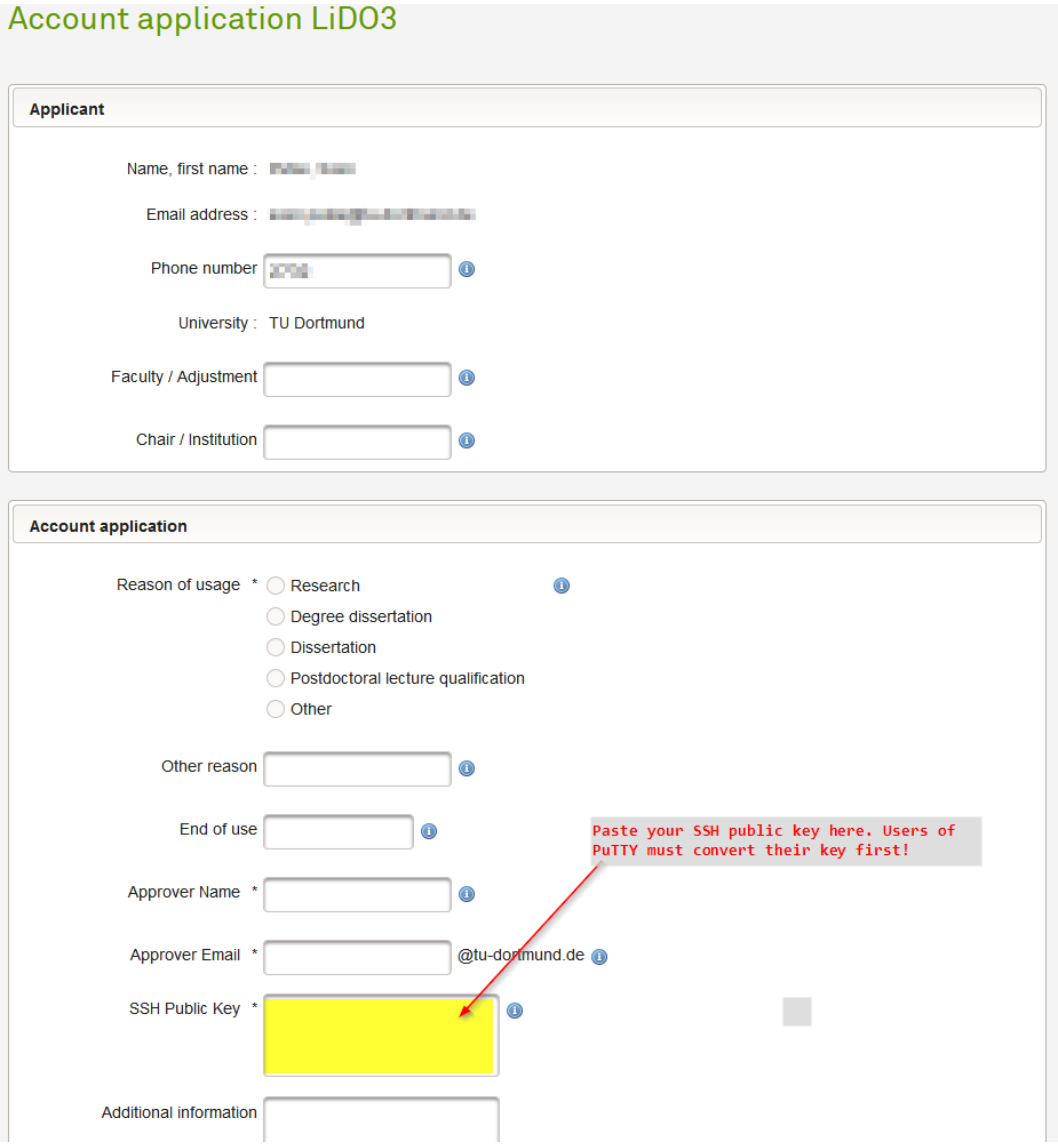


Figure 2.2: Insert your generated public key into the *"SSH Public Key"* field to submit your public key.

If the project is funded by the *Fachhochschule Dortmund* or *UA Ruhr* within the framework of a cooperaton with the *Technische Universität Dortmund*, you have to apply directly at the *Service Desk* with the following filled out form[2]. Please note that this application can only be used by professors for their own projects, doctoral or post-doctoral research.

[2] https://www.itmc.tu-dortmund.de/cms/Medienpool/pdfs/Hochleistungsrechnen/hauptantrag-lido-externe.pdf

```
Zentrale und Serviceeinrichtungen- IT und Medien Centrum (ITMC)
Servicedesk
Otto-Hahn-Str. 12, room E.037
D-44227 Dortmund
```

If you have questions regarding the account or are an applicant from a university outside Dortmund, please contact

```
Maria Pefferkuch
Zentrale und Serviceeinrichtungen- IT und Medien Centrum (ITMC)
Otto-Hahn-Str. 12, room E.036
Phone: +49 231 / 755 - 2367
```

Users from the *UA Ruhr*, please contact the *Service Desk* by telephone to make an appointment.

To manage your existing LiDO3 user account, please use the web forms in the LiDO3 user management portal[3] (access from university network only, SSO login with uni-account is mandatory):

2.1.2 Approval

Upon submitting the application form service portal[4], a ticket is generated in the ITMC ticket system that involves informing your approver (your supervising professor) via e-mail about your account application. The approver is kindly requested to accept or decline your application. Once the approver has accepted or declined your LiDO3 account application, the LiDO team gets informed. If the approver does not react, the LiDO team is, unfortunately, not informed about the pending account application. The *Service Desk*, however, can check for pending account applications and by this check whether your LiDO3 account application was received by the ITMC ticket system at all.

[3] https://l3umw.lido.tu-dortmund.de:8193/usermanagement/static/index.html

[4] https://service.tu-dortmund.de/group/intra/lido3neuantrag

2.1.3 Account creation

Once an approver has accepted your account application, the LiDO team gets informed by the ticket system about it. Typically, within a work day or two your account is then semi-automatically created. If it takes considerably longer and you do not get any feedback about your account creation, it is almost certain the LiDO team has not yet been informed about your pending account application, but that the approver has overlooked the Matrix42 e-mail that asks for approval or denial of your account application. In that case, you may want to check with your approver first before contacting the *Service Desk*.

2.2 SSH Key

SSH keys are used to identify yourself to a computer using public-key cryptography[5] instead of a password. On one hand, this is done for security reasons – a SSH key is much harder to crack than a password, if at all – and on the other hand for user comfort.

The internet is full of tutorials that show *how to create and use a SSH key*, so we will just refer to one example for Linux users[6] and one for Windows/PuTTY users[7].

The use of SSH-Keys is mandatory. You **cannot** log into LiDO3 with a username and password. In case you are prompted for a password other than your SSH key passphrase when you try to log in to either one of the gateway servers, something went wrong with your SSH keys: either the public key entered in the application form got scrambled or the private key does not match the public key (any more).

2.2.1 Create SSH Key on Unix

Open a shell and enter

```
$ ssh-keygen -t rsa -b 4096 -C "your.email@tu-dortmund.de"
```

[5] https://en.wikipedia.org/wiki/Public-key_cryptography
[6] https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2
[7] https://www.howtoforge.com/how-to-configure-ssh-keys-authentication-with-putty-and-linux-server-in-5-quick-steps

If you already have other SSH-keys, you can change the filename here, otherwise just use the default.

```
Generating public/private rsa key pair.
Enter file in which to save the key
 (/home/<username>/.ssh/id_rsa):
```

When prompted, enter a secure passphrase to protect[8] your – private – SSH key.

```
Enter passphrase (empty for no passphrase):
 [Type a passphrase]
Enter same passphrase again:
 [Type passphrase again]
Your identification has been saved in
    ↪ /home/<username>/.ssh/id_rsa.
Your public key has been saved in
    ↪ /home/<username>/.ssh/id_rsa.pub.
(...)
```

Copy and paste **only** the public key (typically marked by the .pub file extension) into the user application form (see page 10) after the successful creation.

2.2.2 Create SSH Key on Windows

Starting from Windows 10, Version 1809 (Oktober 2018 Update; you can check the version by pressing `Win-key+R and then invoking the command \lstinline`winver!), an OpenSSH port is available in the command line (Eingabeaufforderung). This includes the program `ssh-keygen` described in the previous section 2.2.1.

If you want to rely on a GUI-based solution, you can use `puttygen` from the PuTTY Software Suite.[9] Click on *Generate*. To build the key pair, it is important to move your mouse in random directions while the key is generated.

[8] If someone gains access to your computer, they also gain access to **every** system that uses your SSH-key-pair. To add an extra layer of security, you **should** add a passphrase to your SSH key.

[9] https://www.chiark.greenend.org.uk/~sgtatham/putty/

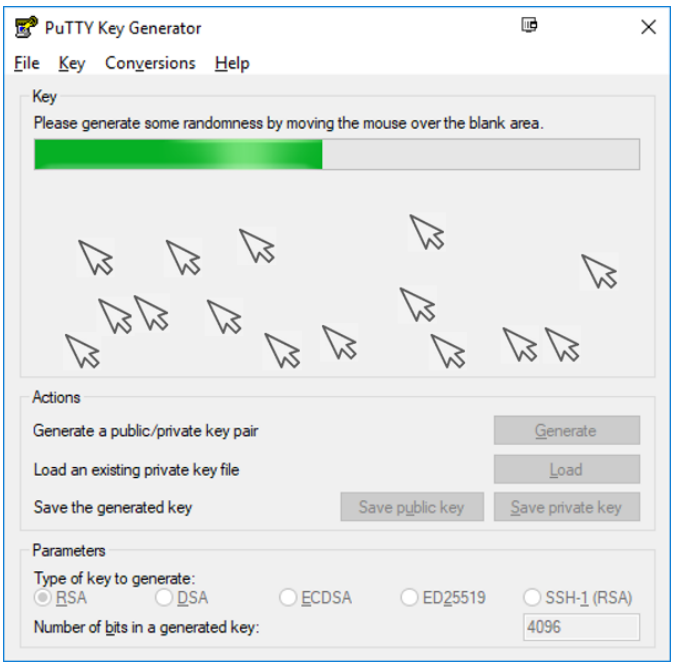


Figure 2.3: Random movements of the mousepointer are used in order to create the key pair.

After that you have to enter a password which is later used to protect your private key.

Save your private and your public key on the Windows machine. PuTTY uses its own file format (suffix `.ppk`) which can not be used on Linux directly. Therefore copy and paste **only** the public key into the user application form (see figure 2.4 on page 10).

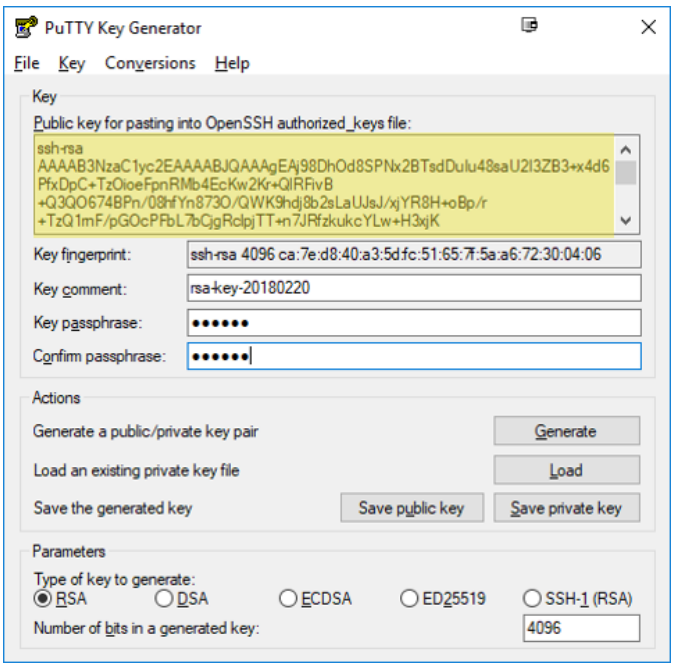


Figure 2.4: Save the private and the public key. Copy and paste the public key (marked in yellow) to the user application form.

2.2.3 Changing your SSH public key

Unlike on other Unix systems your SSH key will not be visible in `~/.ssh/authorized_keys` on LiDO3. Thus any changes to your key must be advertised in the LiDO3 user management portal[10] (access from university network only, SSO login with uni-account is mandatory).

[10] https://l3umw.lido.tu-dortmund.de:8193/usermanagement/static/index.html

Chapter 3

Publications

Please drop us a short e-mail with a citation reference for publications for which LiDO3 has been used. We need this information in our reports to DFG (German Research Foundation) that funded the LiDO3 acquisition.

It would be appreciated if you could include a short acknowledgement in your paper, something along the lines of:

```
The authors gratefully acknowledge the computing time
provided on the Linux HPC cluster at Technical University
Dortmund (LiDO3), partially funded in the course of the
Large-Scale Equipment Initiative by the German Research
Foundation (DFG) as project 271512359.
```

or

```
Die erforderlichen Berechnungen wurden auf dem
Linux-HPC-Cluster der Technischen Universität Dortmund
(LiDO3) durchgeführt, in Teilen durch die
Forschungsgroßgeräte-Initiative der Deutschen
Forschungsgemeinschaft (DFG) unter der Projektnummer
271512359 gefördert.
```

Chapter 4

Working with LiDO3

4.1 Basic workflow

The basic workflow is

- Connect to one of the gateway servers via SSH[1].
- Create a *job*.
- Enqueue the *job* into the *job queue*.
- One or more nodes calculate the result.
- Receive the result on a gateway server.

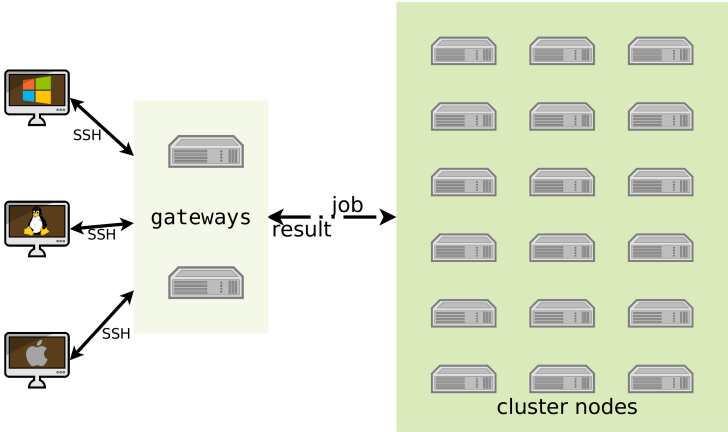


Figure 4.1: Clients connect to one of the gateway servers and transmit jobs.

[1] http://en.wikipedia.org/wiki/Secure_Shell

4.2 Connect

As long as your operating systems has an up-to-date version of a SSH[2]-client, you can connect to one of the gateway servers:

- `gw01.lido.tu-dortmund.de`
- `gw02.lido.tu-dortmund.de`

Both gateways have the same software stack and allow access to all jobs and files, it does not matter which one you choose. If one gateway is down due to maintenance or failure, there is still a second one.

The login credentials consist of your unimail username and the private key of the key pair you provided us in the application form.

If you used a passphrase to protect your private SSH key – what we recommend –, the SSH client (or an authentication agent like `pageant`) will prompt you for that passphrase.[3] Typically, you have about one minute to answer the passphrase prompt until the SSH key exchange is severed by the LiDO3 gateway you are trying to connect to.[5] If otherwise the requested password is not related to the private key file, but to the actual login, e.g.

```
<username@gw01.lido.tu-dortmund.de's password:
```

something is either wrong in your setup or the private SSH key does not match the public SSH key stored on LiDO3.

Please note that LiDO3 is only reachable inside the university network! If you want to use LiDO3 from outside the university, e.g. from home or at a conference, it is mandatory to establish a VPN connection to the TU Dortmund network first. If you try to create a SSH connection to LiDO3 without a VPN connection from outside the TU Dortmund network, you will get a network time out error message. With Putty, the error message looks like depicted in figure 4.2.[6] Given that your SSH connection will

[2] http://en.wikipedia.org/wiki/Secure_Shell

[3] Depending on the actual SSH Client, you might not get any visual or acoustic feedback while you type your passphrase.[4] For instance with Putty, it might seem your keyboard entries are completely ignored until you press the `enter key`.

[5] After that grace period to enter the passphrase has expired, `Putty`, e.g., will report a `Fatal Error` and that the remote side unexpectedly closed the network connection.

[6] See the ServicePortal[7] for up-to-date information and tutorials on how to establish a VPN connection to the TU Dortmund network.

be severed every time you VPN connection gets reset, it is recommended to connect to LiDO3 inside a remote desktop session that runs on a server inside the TU Dortmund network. So, establish a VPN connection to the TU Dortmund network, (re-)connect to a remote desktop session and from within that session create a SSH connection to LiDO3. This way, whenever you are working interactively on LiDO3, e.g. when using a graphical program like Abacus, DDT, Totalview, your entire workflow does not terminate in case your VPN connection gets interrupted. Remote Desktop sessions are available for Windows, Mac and Linux[8]. For non-graphical LiDO3 usage, you may want to use a terminal multiplexing software on the LiDO3 gateways directly, e.g. tmux[10]. This has a lower overhead that a remote desktop session and still protects you from loosing your environment if the network connection to LiDO3 gets interrupted.

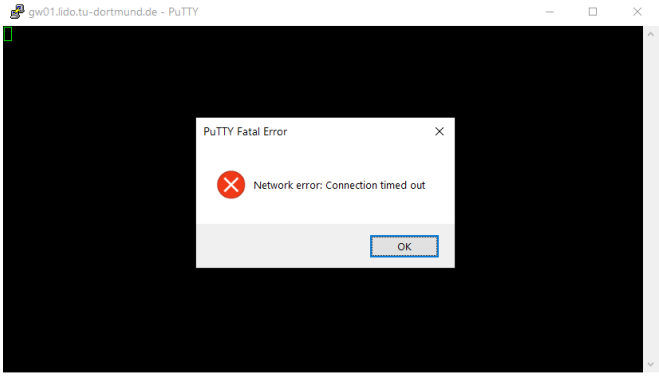


Figure 4.2: Putty reports a Fatal Error while connecting to LiDO3 gateways from outside the TU Dortmund network, without an active VPN connection to the TU Dortmund.

4.2.1 Unix

On any Unix-style operating system you should be able to connect from a terminal via

```
ssh -i <private ssh-key> <account_name>@<gateway_name>
```

replacing `<private ssh-key>` with the path/filename of your private *SSH Key* (see page 12), `<account_name>` with your LiDO-account-name and `<gateway_name>` with one of the pre-mentioned names of the gateway servers.

[8] Look into Cendio ThinLinc software[9] when connecting to LiDO3 from Linux.
[10] https://github.com/tmux/tmux

If you connect to one of the gateway servers for the first time, you will be asked if the key fingerprint from this server is correct. This is done for security reasons to make sure that this *really* is one of the servers you want to connect to. The correct key fingerprints are as follows:

```
$ ssh-keygen -lf <(ssh-keyscan gw01.lido.tu-dortmund.de)
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
256 SHA256:SxL75DVFyNKVbSMkB1M/fPTy5qcPtWa5M9iHHe9OETU
    ↪ gw01.lido.tu-dortmund.de (ECDSA)
2048 SHA256:rG0Cmye6DibyWvaqjHcma6vnwsvTfYATy1JM/O200Ns
    ↪ gw01.lido.tu-dortmund.de (RSA)
256 SHA256:lUQLD2VY/pTVpsSPwuUwvHA8jm/tNiGJ+GbaHP9sBPo
    ↪ gw01.lido.tu-dortmund.de (ED25519)

$ ssh-keygen -lf <(ssh-keyscan gw02.lido.tu-dortmund.de)
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
2048 SHA256:rG0Cmye6DibyWvaqjHcma6vnwsvTfYATy1JM/O200Ns
    ↪ gw02.lido.tu-dortmund.de (RSA)
256 SHA256:SxL75DVFyNKVbSMkB1M/fPTy5qcPtWa5M9iHHe9OETU
    ↪ gw02.lido.tu-dortmund.de (ECDSA)
256 SHA256:sYjJMuRut7jSomxbluWOf0YKE1y5QE5esAQovRBveHo
    ↪ gw02.lido.tu-dortmund.de (ED25519)
```

When using Putty, the fingerprints are given in the MD5 format instead of the SHD256 format.

```
root@gw01: /root>ssh-keygen -E md5 -lf <(ssh-keyscan
    ↪ gw01.lido.tu-dortmund.de)
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
2048 MD5:a5:e3:b4:7f:cc:53:15:32:89:17:d7:ed:14:d5:6e:9d
    ↪ gw01.lido.tu-dortmund.de (RSA)
256 MD5:c6:ee:1d:4b:da:c9:dc:6c:86:08:30:14:f8:ff:18:f8
    ↪ gw01.lido.tu-dortmund.de (ECDSA)
256 MD5:a0:f4:f8:63:e8:79:e5:88:23:2d:1c:44:de:fc:18:81
    ↪ gw01.lido.tu-dortmund.de (ED25519)

root@gw02: /root>ssh-keygen -E md5 -lf <(ssh-keyscan
    ↪ gw02.lido.tu-dortmund.de)
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
```

```
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
2048 MD5:a5:e3:b4:7f:cc:53:15:32:89:17:d7:ed:14:d5:6e:9d
    ↪ gw02.lido.tu-dortmund.de (RSA)
256 MD5:c6:ee:1d:4b:da:c9:dc:6c:86:08:30:14:f8:ff:18:f8
    ↪ gw02.lido.tu-dortmund.de (ECDSA)
256 MD5:2f:58:ae:c4:eb:aa:bb:88:cf:5f:a1:fa:fc:49:0b:64
    ↪ gw02.lido.tu-dortmund.de (ED25519)
```

4.2.2 Windows

Older versions of Microsoft Windows come with no built-in SSH[11]-client-software, so you have to download and install a third-party tool. Having Windows 10, you can install a Linux subsystem[12] and use that to start a connection.

Starting from Windows 10, Version 1809 (Oktober 2018 Update; you can check the version by pressing `Win-key+R and then invoking the command \lstinline`winver!), an OpenSSH port is available in the command line (Eingabeaufforderung). This includes the program `ssh` described in the previous section 4.2.1.

PuTTY[13] is a well known and sufficient ssh-client and it's free. MobaXterm[14] is a fork based on PuTTY with X11 server, tabbed SSH client and network tools.

4.2.2.1 PuTTY

Since Windows users may not be used to connect to other computers via SSH[15], we will describe it more detail here — assuming you use PuTTY[16] as client software. Of course you can use other SSH client software if that suits you better.

Replace `<gateway_name>` at Connection→SSH with one of the pre-mentioned names of the gateway servers, enter the path to your *SSH Key* (see page 13) at Connection→SSH→Auth and click on *Open*.

[11] http://en.wikipedia.org/wiki/Secure_Shell
[12] https://docs.microsoft.com/en-us/windows/wsl/install-win10
[13] http://www.chiark.greenend.org.uk/~sgtatham/putty/
[14] http://mobaxterm.mobatek.net/
[15] http://en.wikipedia.org/wiki/Secure_Shell
[16] http://www.chiark.greenend.org.uk/~sgtatham/putty/

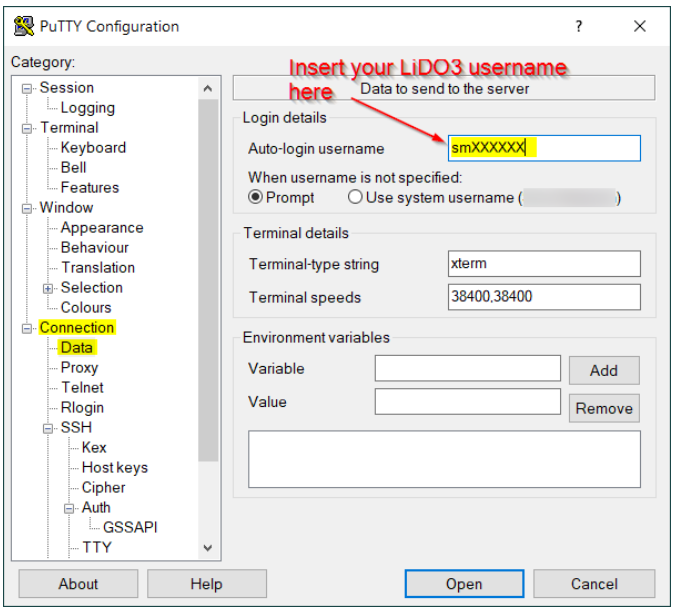


Figure 4.3: Optional: hardcode your LiDO3 user name in your Putty session.

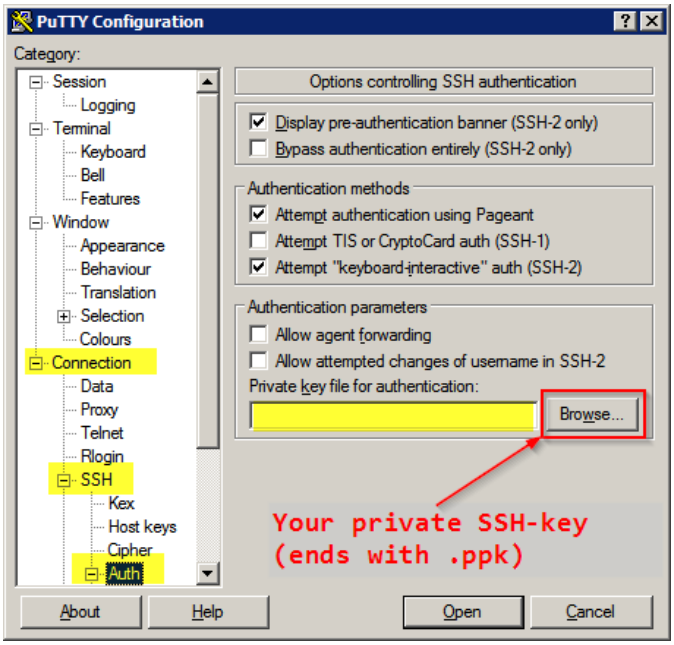


Figure 4.4: Mandatory: Enter the path and filename to your personal private SSH-key.

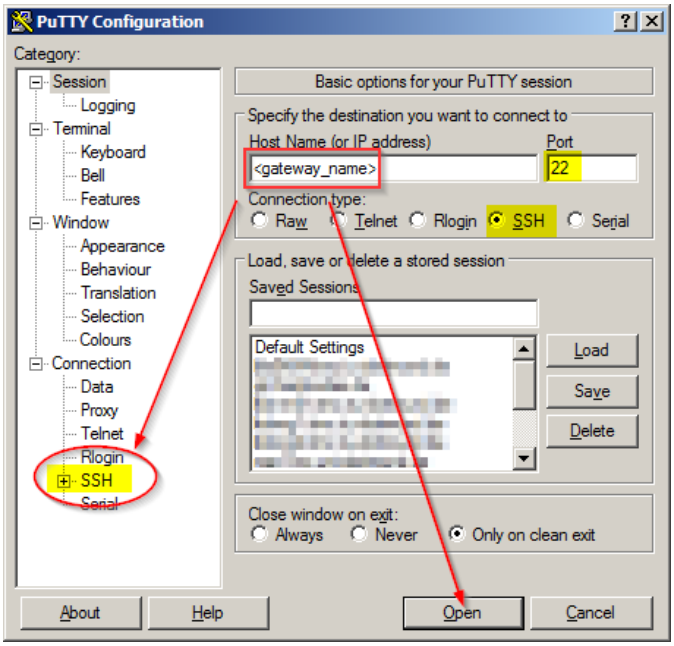


Figure 4.5: Mandatory: Enter the gateway name, path to your private SSH key. Then click *Open*.

If you connect to one of the gateway servers for the first time, you will be asked if the key fingerprint from this server is correct. This is done for security reasons to make sure that this *really* is one of the servers you want to connect to. The correct key fingerprints are as follows:

```
$ ssh-keygen -lf <(ssh-keyscan gw01.lido.tu-dortmund.de)
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
256 SHA256:SxL75DVFyNKVbSMkB1M/fPTy5qcPtWa5M9iHHe9OETU
    ↪ gw01.lido.tu-dortmund.de (ECDSA)
2048 SHA256:rG0Cmye6DibyWvaqjHcma6vnwsvTfYATy1JM/O200Ns
    ↪ gw01.lido.tu-dortmund.de (RSA)
256 SHA256:lUQLD2VY/pTVpsSPwuUwvHA8jm/tNiGJ+GbaHP9sBPo
    ↪ gw01.lido.tu-dortmund.de (ED25519)

$ ssh-keygen -lf <(ssh-keyscan gw02.lido.tu-dortmund.de)
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
2048 SHA256:rG0Cmye6DibyWvaqjHcma6vnwsvTfYATy1JM/O200Ns
    ↪ gw02.lido.tu-dortmund.de (RSA)
```

```
256 SHA256:SxL75DVFyNKVbSMkB1M/fPTy5qcPtWa5M9iHHe9OETU
    ↪ gw02.lido.tu-dortmund.de (ECDSA)
256 SHA256:sYjJMuRut7jSomxbluWOf0YKE1y5QE5esAQovRBveHo
    ↪ gw02.lido.tu-dortmund.de (ED25519)
```

When using Putty, the fingerprints are given in the MD5 format instead of the SHD256 format.

```
root@gw01: /root>ssh-keygen -E md5 -lf <(ssh-keyscan
    ↪ gw01.lido.tu-dortmund.de)
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
2048 MD5:a5:e3:b4:7f:cc:53:15:32:89:17:d7:ed:14:d5:6e:9d
    ↪ gw01.lido.tu-dortmund.de (RSA)
256 MD5:c6:ee:1d:4b:da:c9:dc:6c:86:08:30:14:f8:ff:18:f8
    ↪ gw01.lido.tu-dortmund.de (ECDSA)
256 MD5:a0:f4:f8:63:e8:79:e5:88:23:2d:1c:44:de:fc:18:81
    ↪ gw01.lido.tu-dortmund.de (ED25519)

root@gw02: /root>ssh-keygen -E md5 -lf <(ssh-keyscan
    ↪ gw02.lido.tu-dortmund.de)
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
2048 MD5:a5:e3:b4:7f:cc:53:15:32:89:17:d7:ed:14:d5:6e:9d
    ↪ gw02.lido.tu-dortmund.de (RSA)
256 MD5:c6:ee:1d:4b:da:c9:dc:6c:86:08:30:14:f8:ff:18:f8
    ↪ gw02.lido.tu-dortmund.de (ECDSA)
256 MD5:2f:58:ae:c4:eb:aa:bb:88:cf:5f:a1:fa:fc:49:0b:64
    ↪ gw02.lido.tu-dortmund.de (ED25519)
```

Accept the key with a click on *Yes*.

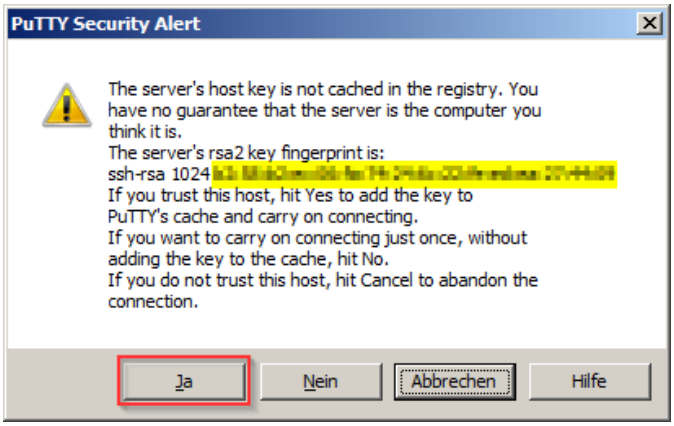


Figure 4.6: Accept the key fingerprint with a click on *Yes*.

Replace `<account_name>` with your LiDO-account-name and press the `[Enter]` key.

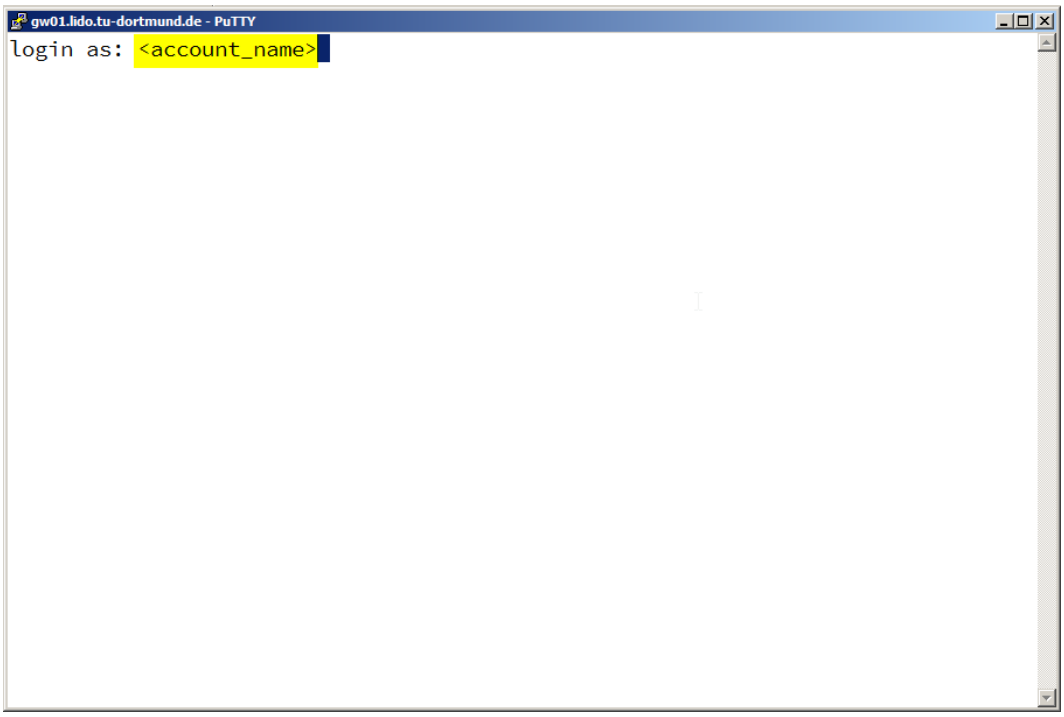


Figure 4.7: Enter your account name and press *[Enter]*.

Enter the password for your *private key* and press the `[Enter]` key. Now you are logged in, welcome to the world of high performance computing.

```
@gw01:~
login as:
Authenticating with public key
Last login: Thu Aug 10 09:25:59 2017 from 129.217.
      / \ / \ / \ / \
     ( L | i | D | O )   Welcome to
      \_/ \_/ \_/ \_/    LiDO3 - Linux cluster DOrtmund, 3rd edition

       I   T   M   C
        TU Dortmund


Operating system is CentOS 7.3

Notes:
1) $HOME is subject to quota restrictions and only writable to on the gateway no
des.
   This is to refrain users from running Slurm jobs directly from their home dir
ectory
   as this would result in a severe performance penalty.
   $WORK (also accessible via $HOME/nobackup) points to the parallel file system
. Slurm
   jobs should be run from $WORK.

2) Slurm is used as Resource/Queue/Workload Manager. If you are familiar with a
```

Figure 4.8: Succesful login to one of the LiDO3-gateways.

You end the session with the command `exit`.

4.2.2.2 WinSCP

If you only want to copy some files to/from LiDO3, you can skip the Terminal/GUI solutions and reside to `scp`, which is copy over SSH[17], i.e. an encrypted file transfer over the network. A widely used `scp` GUI for Windows is called WinSCP[18]. It provides a NortonCommander-like GUI where you can easily transfer files from one side to the other, literally.

The initial setup consists of converting your SSH private key, adding it to WinSCP and adding the LiDO3 gateway URL. By default, the login dialog opens directly. If not, it can be triggered via the button 'New Sessions' (or 'Neue Sitzung' if you use German language settings) and from the menu 'Session' ('Sitzung').

[17] http://en.wikipedia.org/wiki/Secure_Shell
[18] https://winscp.net/eng/index.php

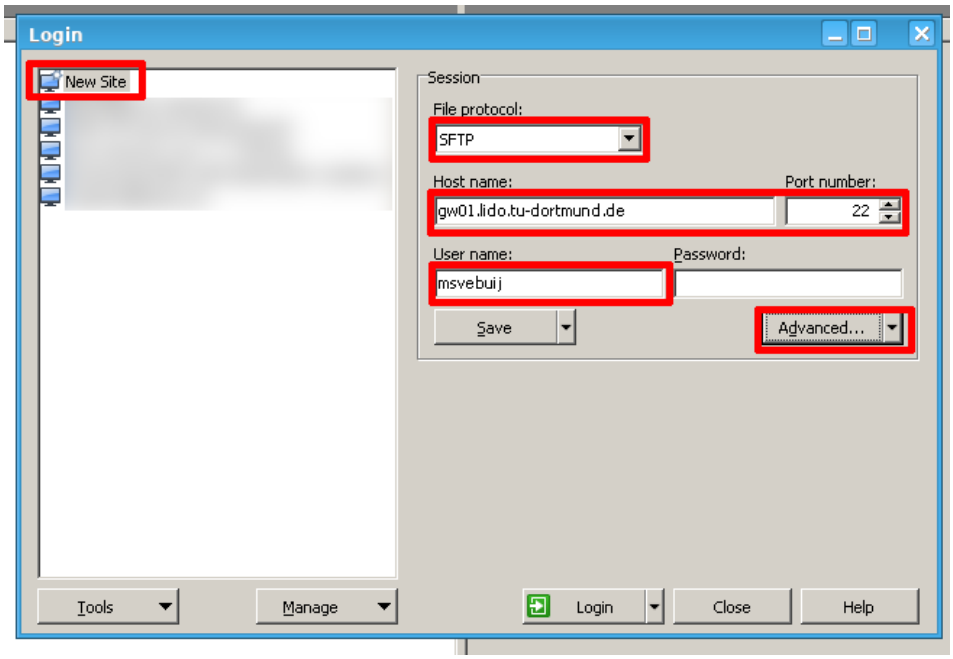


Figure 4.9: Setting up protocol, server and username and opening advanced settings

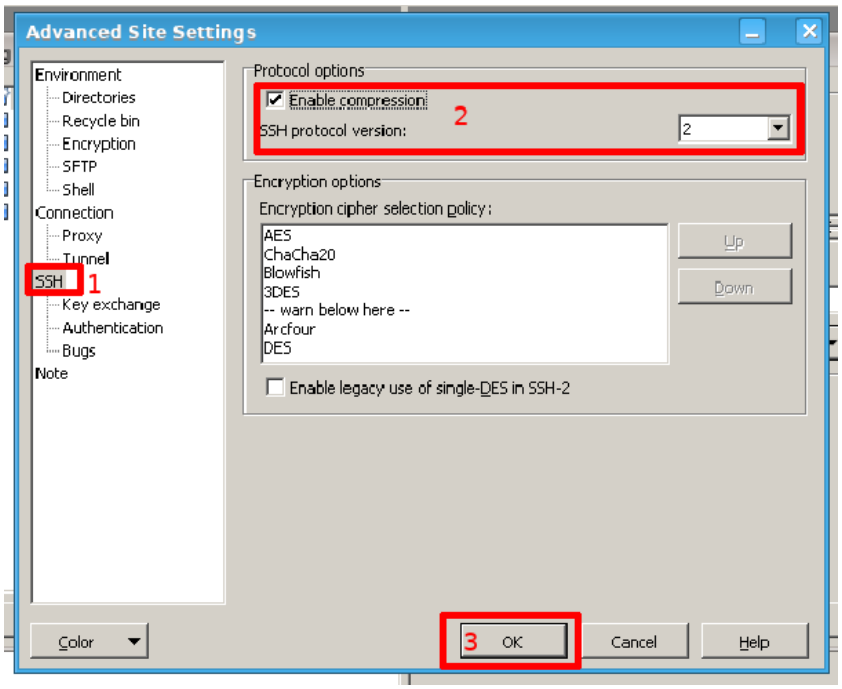


Figure 4.10: Enable SSH compression.

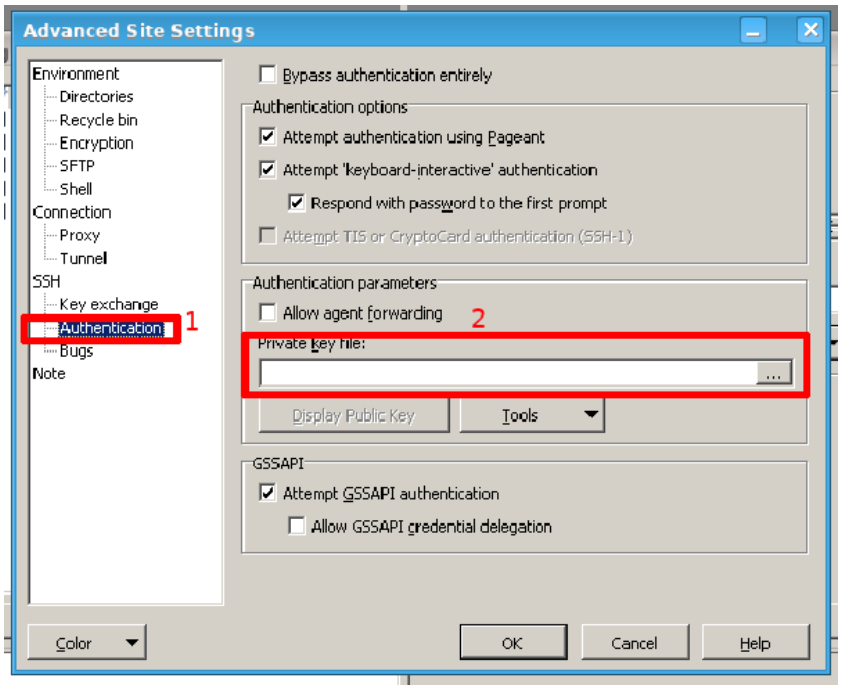


Figure 4.11: Open private key selection.

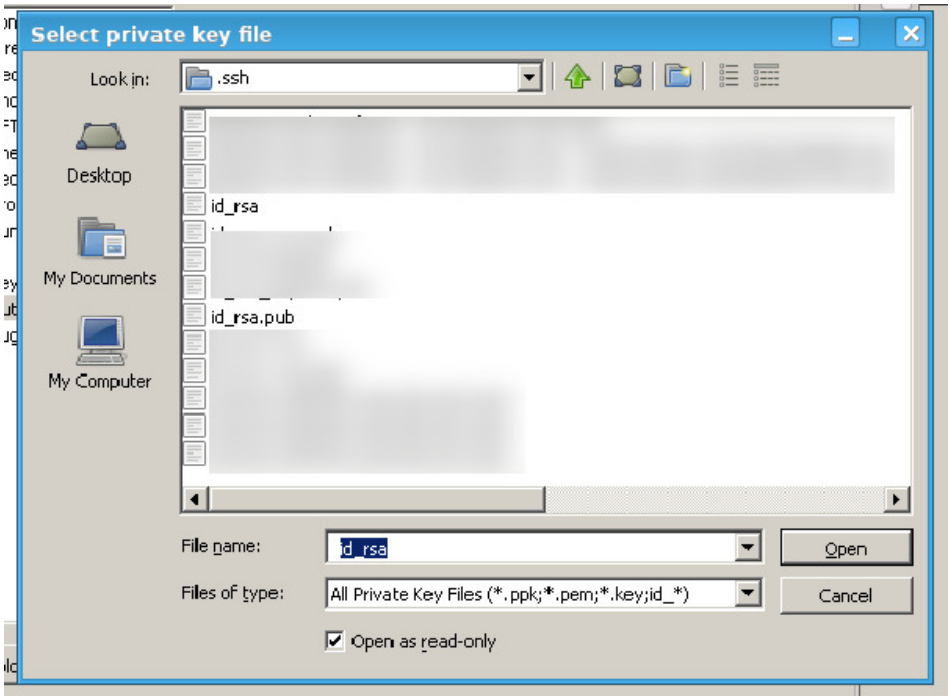


Figure 4.12: Select private key.

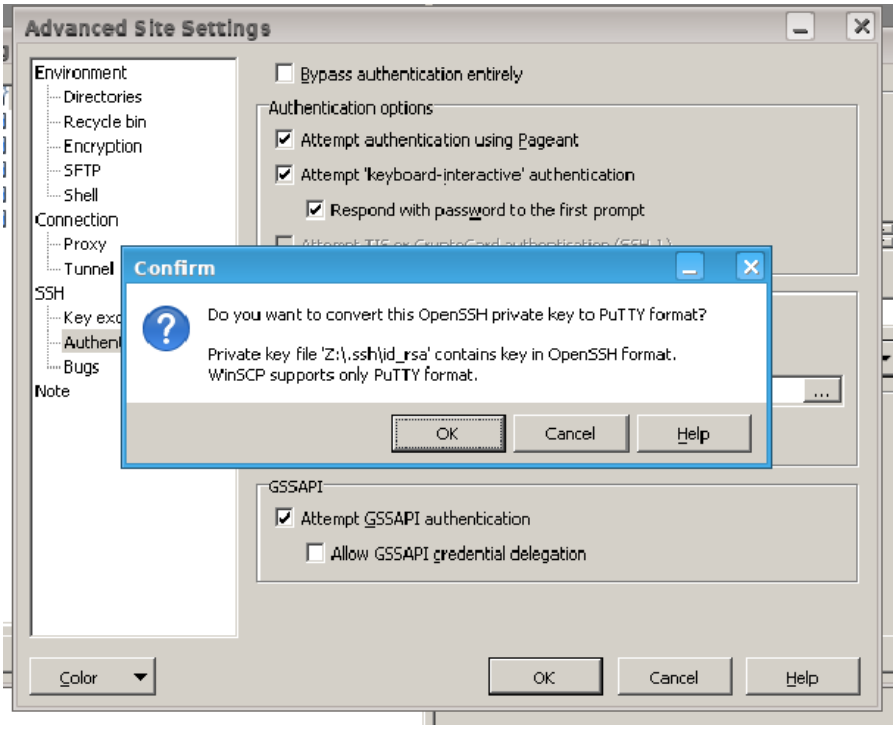


Figure 4.13: Confirm private key conversion.

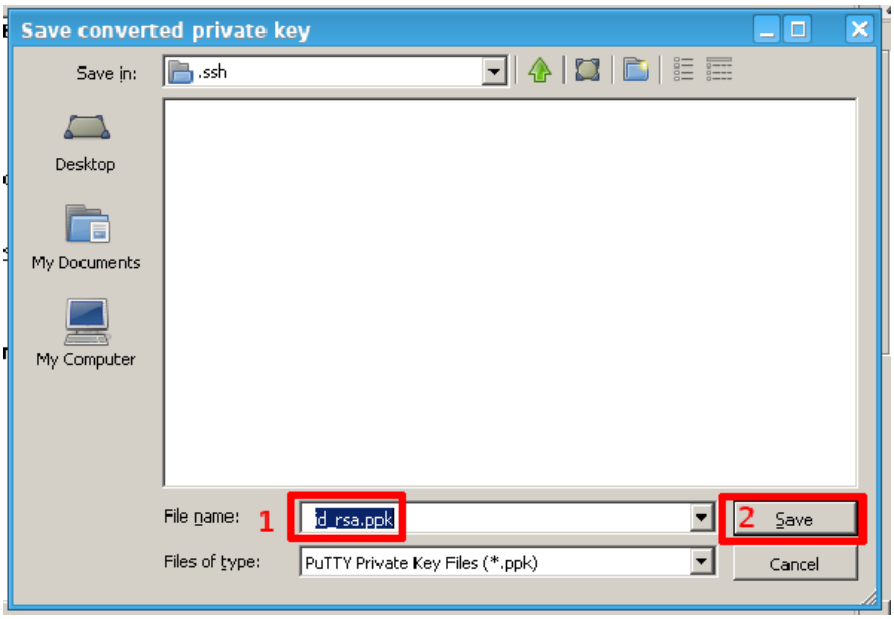


Figure 4.14: Save converted key file.

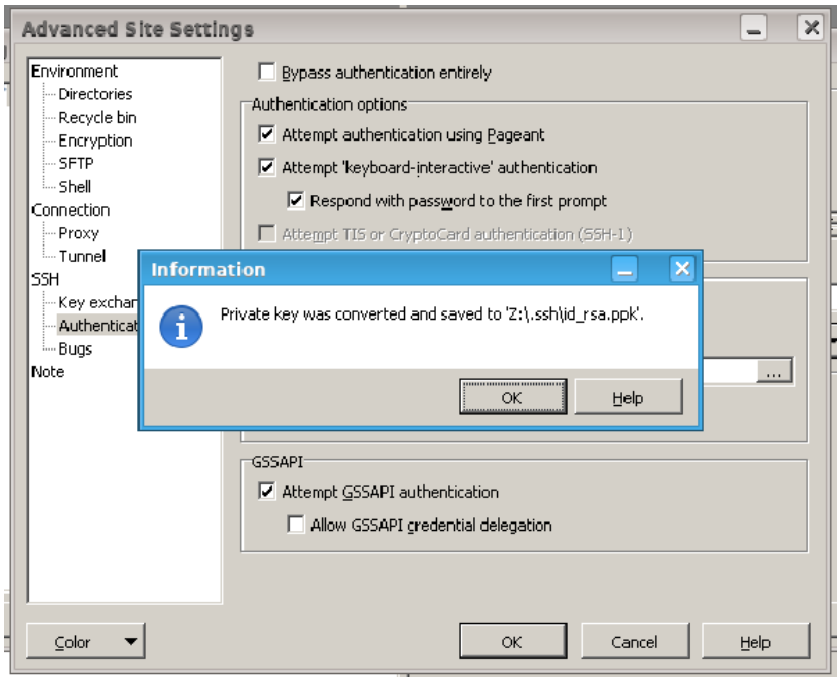


Figure 4.15: Acknowledge success information.

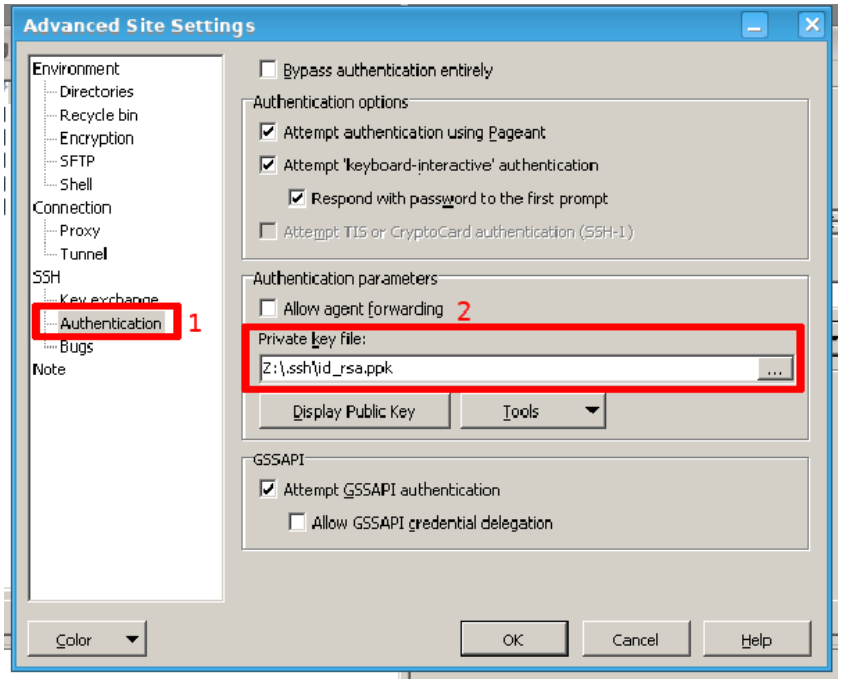


Figure 4.16: Confirm key selection.

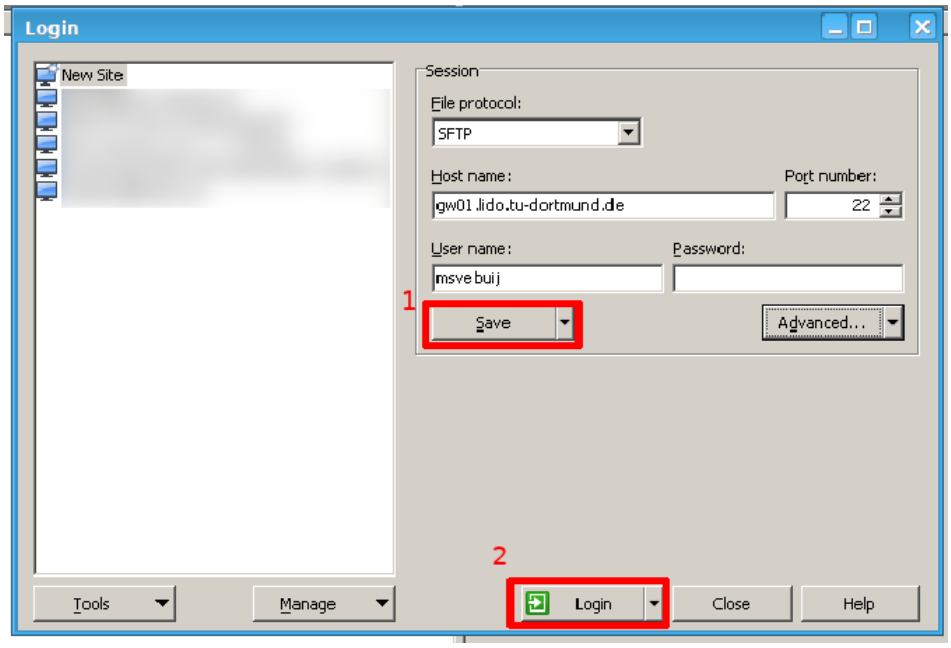


Figure 4.17: Save connection setup and open connection.

After you are connected, you can copy files around by drag-drop moving them from one window to the other or using menu entries and keyboard shortcuts, respectively. You can move (i.e. up-/download, then delete at source location) files to/from your local client, move them on the server side to different server-side locations, rename, edit and delete them.

4.2.3 Inter-node connections

Similar to connections from the outside, connections between LiDO3 compute nodes and gateways are only allowed via password-less, public key-based SSH authentification. Preferably, those SSH keys are not protected by a passphrase either.

In order to generate an additional SSH key pair that allows password-less SSH logins between LiDO3 gateway servers and compute nodes (e.g. to be able to use software like ANSYS Fluent), proceed as follows:[19]

Step 1)

In a login shell on one of the LiDO3 gateway servers, invoke the following command

[19] Please note that the leading dollar sign, $, in the commands listed below are meant as a mere placeholder for your prompt and **should not** be entered, too.

```
$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa.lido3intern
```

You will be prompted for an optional passphrase. Twice, in order to confirm your input. Omit a passphrase by just pressing the enter key when prompted for the passphrase.[20] The command will generate, besides an output along the following lines

```
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in
  ↪ /home/user/.ssh/id_rsa.lido3intern.
Your public key has been saved in
  ↪ /home/user/.ssh/id_rsa.lido3intern.pub.
The key fingerprint is:
SHA256:[...] user@gw01
The key's randomart image is:
+---[RSA 4096]----+
|       +         |
|           ~     |
|    .    -      .|
|     . .       ++|
|.     . S  . .++*|
|   o   o oo +ooo=|
|      o = .*+=o+.|
|    +  + +o+o.*=*|
|      . ... o++E@|
+----[SHA256]-----+
```

a new SSH key pair consisting of a private and a public key, in the appropriate file formats "PEM RSA private key" and "OpenSSH RSA public key", respectively.

[20] You may wonder whether this is a security concern: No, *it is not* because this SSH key pair will **only** be used inside LiDO3. While it is strongly advised to passphrase-protect the SSH private key that is stored on your local computer and that you use to log in to LiDO3 from your local computer (together with the corresponding unprotected SSH public key that is stored on LiDO3), there is no additional security gain by protecting any private SSH key that you use merely for SSH connections between LiDO3 compute nodes: anyone who gets access to your home directory on LiDO3 (e.g. by getting a copy of your not passphrase-protected private SSH key or your passphrase-protected private SSH key plus the passphrase) does not gain anything additionally with your unprotected SSH key pair for LiDO3 inter-node SSH connections.

Use the generated SSH key pair for LiDO3 inter-node connections **only** and **do not** use SSH keys from other systems. In case of a security breach on LiDO3, those (private) SSH keys might be stolen and used to connection from LiDO3 to other computer systems. Private SSH keys that allow to log in to multiple systems outside LiDO3 impose the danger of compromising additional systems outside LiDO3.

Step 2)

Tweak your SSH configuration on LiDO3 to use the new SSH private key by default when making logins. The easiest way to do this is to create or modify the file `~/.ssh/config`. By default, this configuration file does not exist. Verify that the file `~/.ssh/config` does not exist yet by invoking

```
$ ls ~/.ssh/config
```

If it does not exist, you will get the following error message

```
ls: cannot access /home/<your username>/.ssh/config: No such
↪ file or directory
```

In this case, you can create the file and store the required information in a single step by simply invoking

```
$ echo "IdentityFile ~/.ssh/id_rsa.lido3intern" >
↪ ~/.ssh/config
```

You can check the contents of the newly created file with either one of the commands

```
$ cat ~/.ssh/config
$ more ~/.ssh/config
$ less ~/.ssh/config
```

If the file `~/.ssh/config` does exist for you, you are most likely experienced enough to customize it appropriately with an editor of your choice (on LiDO3, e.g. `nedit`, `pico`, `nano`, `emacs`, `vim`, or on your local computer, additionally transferring the newly created file to LiDO3 afterwards in the latter case) such that we can refrain from detailing how to store the following line in the appropriate line:

```
IdentityFile ~/.ssh/id_rsa.lido3intern
```

Make sure that the file `~/.ssh/config` has file permissions as the SSH client expects it – otherwise its content will be completely ignored and you will still not be able to use password-less logins. Proper file permissions can be imposed by running any of the following three commands in a shell on LiDO3:

```
$ chmod u=rw,g=r,o=r ~/.ssh/config
$ chmod 644 ~/.ssh/config
$ chmod g-w ~/.ssh/config
```

Step 3)

Configure your own SSH setup on LiDO3 such that this newly generated SSH key pair is used and accepted when attempting to SSH connect to a compute node. Do this by appending the content of the newly generated SSH public key file, `~/.ssh/id_rsa.lido3intern.pub` to the file `~/.ssh/authorized_keys`:

```
$ cat ~/.ssh/id_rsa.lido3intern.pub >> ~/.ssh/authorized_keys
```

Again, make sure that the file permissions of `~/.ssh/authorized_keys` are more restrictive than they are by default. Otherwise, password-less, public key-based SSH logins will silently fail, for no apparent reason. So, next invoke either one of the following commands:

```
$ chmod u=rw,g=r,o=r ~/.ssh/authorized_keys
$ chmod 644 ~/.ssh/authorized_keys
$ chmod g-w ~/.ssh/authorized_keys
```

Step 4)

In order to make sure that this new, passphrase-less SSH key pair is only used on LiDO3 and merely for internal logins, you can – using an editor of your choice (on LiDO3, e.g. `nedit`, `pico`, `nano`, `emacs`, `vim`, or on your local computer, additionally transferring the newly created file to LiDO3 afterwards in the latter case) – prepend the line you just appended to the file `~/.ssh/authorized_keys` with a `from-string`. With the prefix

```
from="10.10.*"
```

the new SSH key pair will only be accepted for logins from within LiDO3. So, after editing the file `~/.ssh/authorized_keys` in step 3 and 4, its content (see above for how to view its content with, e.g., the command line tools `cat`, `more` or `less`) should look something along the lines of

```
from="10.10.*" ssh-rsa AAAAB3NzaC1yc2EA[....] 5sJ5Qw==
↪ user@gw01
```

Once you have traversed the steps above, you can try out password-less logins to a compute node by first requesting, for instance, a small, 5-minute interactive shell from Slurm via

```
$ srun --partition=short --nodes=1 --cpus-per-task=1
↪ --time=00:05:00 --pty bash
```

Once your interactive Slurm job starts on, say, `cstd01-001` (and for as long as that interactive Slurm job is running, in this example for 5 minutes), you should be able to login - from a different login on a LiDO3 gateway - to that very same compute node via SSH, too, without being asked for a password or passphrase.

Due to step 2, it is not necessary to tweak your existing Slurm job scripts in any kind in order for this new SSH key pair to be used implicitly.

The new SSH key pair will not interfere with the pre-existing SSH key pairs you use to log into LiDO3 itself. The new SSH key pair will never be queried when connecting from outside to LiDO3. Regardless whether you took the optional step 4 or not.

4.2.4 Troubleshooting

4.2.4.1 Keyfile permissions

The SSH clients are somewhat picky regarding the file permissions of the private key files and the personal SSH configuration file.
In Linux, you can set the right file permissions via

```
chmod 600 <file name>
```

In Windows, you can set the right file permissions via

```
Icacls <file name> /Inheritance:r
Icacls <file name> /Grant:r "%Username%":"(R)"
```

4.2.4.2 Getting prompted for a password on login

If you are asked for a password (other then the password securing your private key) you most probably provided no or the wrong private key. In this case, SSH automatically skips forward to the next authentification method, i.e. password authentification. On LiDO3 password authentification is disabled and thus this login method will fail, regardless of which password you provide in this step.

4.2.4.3 Rejected connections

After a few failed login attempts, your IP address is blocked for 30 minutes to prohibit brute force attacks. After 30 minutes, connections are accepted again.

4.3 Linux Environment

4.3.1 Working with the Linux shell

If you have never worked with the Linux Shell Bash[21] before, you can find more than one tutorial[22] in the internet.

4.3.1.1 Editing files

Working with a Linux Shell and with LiDO3 means working with textfiles. Here is a list of installed text editors:

- vi[23]
- emacs[24]
- gedit[25]

[21] https://en.wikipedia.org/wiki/Bash_%28Unix_shell%29
[22] http://tldp.org/LDP/Bash-Beginners-Guide/html/
[23] https://en.wikipedia.org/wiki/Vi
[24] https://en.wikipedia.org/wiki/Emacs
[25] https://en.wikipedia.org/wiki/Gedit

- nedit[26]
- nano[27]
- pico[28]

Choose the one that suits your needs.

Some of the editors might seem rather strange for Windows users and if desired, one can create and edit the text files locally on the Windows workstation and copy them via to one of the gateway server or vice versa.

Just keep in mind that the newline[29] character is handled differently on Linux and Windows. You want to use a feature like *ASCII mode = newline conversion* in your SSH client software - if available.

4.3.2 Filesystems

4.3.2.1 /home and /work file systems

On LiDO3 there are two file systems available on both gateway servers and all compute nodes:

- `/home` and
- `/work`

On both of them user quotas are enabled. Available disk space quota and current quota usage is automatically shown on login.

We would like to point your attention to the different properties of the two file systems `/home` and `/work` available on LiDO3:

- `/home` has a quota of 32 GiB for user data, but its content is backed up on tape such that in case of a file system problem the `/home` file system and its data can be restored. On login, the current quota usage is displayed. It can be manually queried by running

```
df -h $HOME
```

[26] https://en.wikipedia.org/wiki/NEdit
[27] https://en.wikipedia.org/wiki/GNU_nano
[28] https://en.wikipedia.org/wiki/Pico_(text_editor)
[29] https://en.wikipedia.org/wiki/Newline

`/home` is provided by two redundant NFS servers and is hence a network file system, but not a parallel file system.

`/home` is **read-only**, i.e. **write-protected** on the compute nodes! If the software you execute on the compute nodes needs to write to the home directory, you have two options:

- Redefine `HOME` before invoking the command. Bash users can prepend the actual command with `HOME=/work/${USER}`.
- Create symbolic links in your home directory to an alternate writable location. See on page 110 for some examples of already existing software.

- `/work` has different characteristics: it has a default quota of 2 TiB[30] for user data, but the files are ***not*** saved externally - due to financial limitations (human resources, backup capacity and intra-university network bandwidth). It is provided by several redundant file servers, uses the parallel file system `BeeGFS` and has a total size of 1.28 PiB. `/work` can be read from and written to on both gateways and all compute nodes. The *link* in your home directory called "nobackup" leads to the `/work/${USER}` directory.

  The quota can be manually queried by running

  ```
  beegfs-ctl --getquota --uid $USER
  ```

  In case of a severe file system problem the data might get ***LOST*** completely.

  This is no mere theoretical risk, on its predecessor cluster LiDOng it **has happened multiple times**. Please keep this in mind and *backup important files in `/work` yourself* at regular intervals. If it is technically possible when an emergency situation arises, we will grant a two days window to make backups. Don't firmly rely on this chance, though, and keep in mind that when storing terabytes of data on LiDO3 your network bandwidth might not suffice to transfer all your data from LiDO3 within two days.

[30] Since 2020-05-15 this quota is not only shown but also enforced! Exceptions may be granted after sending a written justification.

```
cd /home/<user>/nobackup/<my-app>
sbatch myjob.sh
```

Since it is in the nature of a high performance cluster that many nodes, cores and processes access data simultaneously on those file systems, the cluster uses a parallel distributed file system named BeeGFS[31].

While beeing a specialist for parallel access patterns, there is also a caveat: working with many small files and accessing the directory structures (in doing any equivalent of `ls`) stresses the parallel file system. **Do not do that!**

4.3.2.2 Read-only /home directory on compute nodes

X11 To be able to use X Window System software on compute nodes, the X11 magic cookie needs to be written to/updated in a file named `.Xauthority`. Typically, this file is stored in a user's home directory. To work around the fact that the `/home` directory can not be written to on the compute nodes, a workaround has been set up system-wide, the file `/work/${USER}/.Xauthority` is used instead.

GnuPG If you plan to use software that uses `gpg` to verify the signature of files, please note that `gpg` tries to create temporary files in `${HOME}/.gnupg` while doing so. In order to have `gpg` successfully verify signatures on compute nodes, you need to move the directory `${HOME}/.gnupg` to e.g. `/work/${USER}/.gnupg` and set a symbolic link to this new location in your home directory instead:

```
test -d ${HOME}/.gnupg || mkdir ${HOME}/.gnupg
mv ${HOME}/.gnupg /work/${USER}
ln -s /work/${USER}/.gnupg ${HOME}
```

4.3.2.3 Dealing with the disk space quotas

As stated before in 4.3.2.1, the maximum disk space usage in `/home` is restricted to 32GiB and in `/work` to 2 TiB. If you regularly reach these limits, there are several steps that might be helpful.

- obviously: delete programs, sourcecode and data, that you do not need anymore
- move everything that you can easily recover to `/work`.

[31] https://en.wikipedia.org/wiki/BeeGFS

- move all binaries, your own compilations and third-party programs, to `/work`
- if you have source code checkouts, that you do not change on your own on LiDO3, move them to `/work`
- store reproducable application output to `/work`
- move data, that you do not need on LiDO3 in the near future to other storage sites. This has the benefit of not loosing data on `/work` on a filesystem malfunction
- use binary/compressed output formats where available. The usual ASCII-based data storage is very wasteful
- compress application output directly in your Slurm script or at least afterwards, when you have finished your first-level analysis.

4.3.2.3.1 Compressing application data

There are several programs readily available on LiDO3 (gateway and compute nodes) to compress you application data.

zip:

```
# compress files
zip archive.zip file1 file2
# recursively compress complete directories
zip -r archive.zip directory1 directory2
# inspect
zipinfo archive.zip
# decompress
unzip archive.zip
```

tar with gzip:

```
# compress files
tar cvzf archive.tar.gz file1 directory2
# inspect
tar tvzf archive.tar.gz
# decompress
tar xvzf archive.tar.gz
```

tar with bzip2:

```
# compress files
tar cvjf archive.tar.bz2 file1 directory2
# inspect
tar tvjf archive.tar.bz2
# decompress
tar xvjf archive.tar.bz2
\textbf{tar with xz:}
\begin{lstlisting}
# compress files
tar cvJf archive.tar.xz file1 directory2
# inspect
tar tvjf archive.tar.xz
# decompress
tar xvJf archive.tar.xz
```

4.3.2.4 /scratch file system

If you need to do heavy I/O or parallel processing of data in files, consider using the `/scratch` file system. `/scratch` is a local file system on each node that can't be accessed from other machines.

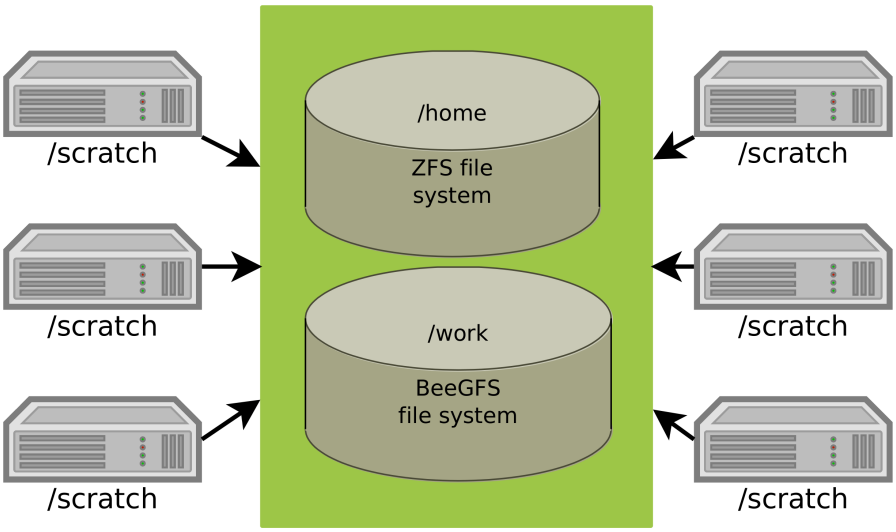


Figure 4.18: */home* and */work* can be accessed from any node, */scratch* is only a local file system.

The workflow would look something like this:

- Job starts
  - Copy data from `/work` to `/scratch`

- Job runs
  - Process data on `/scratch`
- Job ends
  - Copy data from `/scratch` to `/work`

It is a good approach to create a directory in `/scratch` consisting of your user name and job ID is as in `/scratch/<username>_<job_id>`.

4.3.3 Filetransfer between LiDO3 and external computers

The simplest approach is to use ssh, precisely scp, which is in some sense the cp replacement from the ssh suite. On an external linux/macos/unix/windows wsl machine, the command

```
my_pc# scp  -i <path-to-your-private-ssh-key>
    ↪ <path-to-local-file>
    ↪ lido-user-name@gw01.lido.tu-dortmund.de:/home/lido-user-name/
```

copies a file into your home directory on LiDO3. The command

```
my_pc# scp  -i <path-to-your-private-ssh-key>
    ↪ lido-user-name@gw01.lido.tu-dortmund.de:/home/lido-user-name/some_file
    ↪ <local-target-directory>
```

copies a file back to your local computer. The parameter '-r' copies complete directories recursively. See 'man scp' for further details.

There are also some GUI[32] clients for transfering the files back and forth from your Windows machine, e.g. FileZilla[33] and WinSCP[34]. For both programs, the respective websites explain how to set up SSH public key authentication[35],[36].

[32] https://en.wikipedia.org/wiki/Graphical_user_interface
[33] https://filezilla-project.org/
[34] http://winscp.net/
[35] https://wiki.filezilla-project.org/Howto
[36] https://winscp.net/eng/docs/guide_public_key

4.3.4 Shared file access

It is possible to grant other users read and/or write access to your own files and directories. One common solution to achieve this is by exploiting the group feature common to all unixoid operating systems.

You can ask the LiDO3 support team to create such a unix group containing multiple LiDO3 users to grant all of them read/write access on selected files and directories.

Usually, you or any other member of the same unix group will want to create a subdirectory in someone's home or work directory which is dedicated for this group's work. You need to share this directory's name with your unix group members as they – by default – can not list the content of your (home/work) directory. They can, however, once everything is set up, see everything that is stored in said subdirectory.

Technically speaking, if you grant write access to a shared subdirectory, its content – along with all files and directories underneath it – are owned not only by you, but by your unix group. For this, the setgid bit[37] needs to be set, such that all newly created files and directories are owned by this unix group, too.

Members of this unix group kann read any file if at least:

- The file belongs to the unix group.
- For all directories in the hierarchy leading to the, the x-bit is set for the group (or, if it is not set for the group, it is set for everyone).
- The r-bit of this file is set for the group (or, if it is not set for the group, it is set for everyone)

Example:
Users *sma* and *smb* are members of the group *uxg*. Group memberships can easily be checked by issuing the command `id`, optionally providing a single username, e.g. `id smb`. User *smb* wants to use a file in the home directory of user *sma*.

```
$ ls -lad /home
drwxr-xr-x 281 root root 282 Jul 10 16:09 /home
         ^
         |
         +---- active x-bit allows access for smb
```

[37] https://en.wikipedia.org/wiki/Setuid

smb is neither the owner of the directory `/home` (which is `root`) nor member of the unix group `root`), but does belong to the category *other* The x-bit is set for the topmost directory `/home` for category *other* such that every valid user, including *smb*, can enter this directory. (See Wikipedia on Unix Permissions[38] for details.) He can even issue an `ls` as the r-bit is set for *other* for this directory, too.

The next directory in the hierarchy towards the home directory of user `smb` is `/home/sma`. Because the x-bit is set for *other* for this directory, the user *smb* can enter this directory, too. Nevertheless, he cannot see the content of this directory, due to the missing r-bit for both the group triple and *other* triple.

```
$ ls -lad /home/sma
drwx-----x 7 sma sma 32 Jul 12 13:31 /home/sma
         ^
         |
         +---- active x-bit allows access for smb
```

Finally, the directory `/home/sma/shared-work`, which shall contain the actual shared files, belongs to the unix group *uxg*. The x-bit for this group allows user *smb* to enter this directory.

```
$ ls -lad /home/sma/shared-work
drwxr-x--- 4 sma uxg 4 Jul 12 13:50 /home/sma-shared-work
```

The r-bit for the group *uxg* allows user *smb* to see the contents of this directory, too. Other users that are neither member of the unix group *uxg* nor the user *sma* (a.k.a. the owner) itself cannot see the contents or even enter the directory, because the r-bit is not set for the third *other* triple.

If *sma* should ever change the name of the directory `sma-shared-work`, he would need to tell this to the other members of the unix group *uxg*, because they cannot find out the new name in `/home/sma` themselves. Given that they are not able to see its contents at all.

All newly created files and directories in and beneath `/home/sma-shared-work` will be read- and writeable for the user *sma* and all members of the group *uxg*. This (default) behaviour is controlled by the so-called umask[39] and its current values.

[38] https://en.wikipedia.org/wiki/File-system_permissions
[39] https://en.wikipedia.org/wiki/Umask

```
$ umask -S
u=rwx,g=rwx,o=rx
```

If, for example, you wanted to change the default setting such that other members of the group *uxg* can only read, but not write to newly created files, you could issue the command

```
umask -S u=rwx,g=rx,o=rx
```

once or add it to your `~/.bashrc` file for persistent impact.

4.3.5 Software modules

The software and tools needed for development and job execution are organized as *modules*. *Modules* dynamically modify the users environment and make it possible to

- get a clean environment with no software visible at all,
- install concurrent versions of the same software and
- use software that usually excludes each other.

Working with those modules is done with the module[40] command.

4.3.5.1 Loaded modules

The command `module list` shows the modules that are currently loaded in your environment:

```
$ module list
No Modulefiles Currently Loaded.
```

4.3.5.2 Available modules

To list the modules that can be potentially loaded, enter the command `module avail`.

```
$ module avail

--- /usr/share/Modules/modulefiles ---
```

[40] http://linux.die.net/man/1/module

```
dot   module-git   module-info modules   null   use.own

---- /cluster/sfw/modulefiles ---
abaqus/2016   gcc/6.4.0    openblas/0.2.19
clang/4.0.1   gcc/7.1.0    openmpi/mpi_thread_multiple/cuda/2.1.1
(...)
```

4.3.5.3 Load a module

To load a module into your environment, enter the command `module add`, followed by the `<MODULE_NAME>`:

```
$ module add clang
$ module list
Currently Loaded Modulefiles:
  1) clang/4.0.1
```

4.3.5.4 Unload a module

To unload a specific module, use the command `module rm`, followed by the `<MODULE_NAME>`:

```
$ module rm clang
$ module list
No Modulefiles Currently Loaded.
```

To unload all modules, use `module purge`.

Further documentation of the module concept is available at the HLRN[41].

Important: in order to make the activated modules available on the compute nodes (during execution time) as well, the command module add must be included in the user's shell init files (e.g. `.bash_profile` or *job script*).

4.3.5.5 Modules in job scripts

If you run a job that depends on modules, please ensure that these modules are included in the user's shell init files (e.g. `.bash_profile`), so that the job has a proper environment set up when being executed on the compute nodes! Alternatively, the following lines are to be included in the *Slurm* job script before starting the application:

[41] https://www.hlrn.de/home/view/System2/ModulesUsage

```
# Clean module environment
module purge
# Load modules needed
module load [compiler modules][MPI modules]
```

4.3.5.6 Compiler modules

Compilers and libraries are selected and activated via `module` commands (see section *4.3.5 Software modules*).

Table 4.1: Compilers

Compiler	Module	Commands
GNU Compiler Collection	`module add gcc`	`gcc`, `g++`, `gfortran`
Intel Studio XE	`module add intel`	`icc`, `icpc`, `ifort`
Portland PGI compiler	`module add pgi`	`pgcc`, `pgCC`, `pgf77`, `pgf95`
Oracle Solaris Studio	`module add oraclestudio`	`suncc`, `sunCC`, `sunf77`, `sunf95`
Clang compiler	`module add clang`	`clang`, `clang++`

If you want to compile a parallel program using *MPI* you can use the corresponding compiler wrappers from the *Open MPI* modules.

The naming scheme for the openmpi modules is as follows:

`openmpi//THREADINGSUPPORT/CUDASUPPORT/OPENMPIVERSION`

with

- THREADINGSUPPORT: whether build with thread multiple support :[42] mpi_thread_multiple/no_mpi_thread_multiple
- CUDASUPPORT: whether to enable the build-in support for data transfers between the GPUs and the network controller without explicit memory transfer statements.
- OPENMPIVERSION: the actual *Open MPI* version, e.g. `4.0.1`

For a complete overview of all modules available please see:

```
module avail openmpi
```

[42] https://www.open-mpi.org/doc/current/man3/MPI_Init_thread.3.php

4.3.6 Installing your own software

Many software packages can be installed in your own `/home` or `/work` directory. Admittedly, sometimes you are required to install – as a prerequisite for the software - certain libraries locally as well. Usually, you do not need any supervisor or admin privileges to do so.

In contrast to most manuals, which describe a single-user computer setting where one user is using one computer, LiDO3 is a multi-user system and thus some steps to install a software package will differ from common documentation.

First, you have no superuser rights nor any sudo rights. So, instead of installing any application system-wide via root or sudo commands, you need to limit yourself to an installation in your own directories. This implies especially no usage of commands like `apt`, `apt-get` or `yum` and nothing starting with `sudo`.

Instead of that, you need to search for installation modes called 'local' or 'single-user' or possibilities to change the 'installation target directory' or similar terms.

In the following we depict some common installation routines and how they need to be modified for local installations.

If the software you want to install happens to absolutely fail for a user-level installation, feel free to ask the LiDO3 team 4.8 for additional support.

4.3.6.1 configure-make-install

Most classic Unix/Linux software packages use GNU Autotools[43] (`aclocal`, `autoconf`, `automake`) for their build system. As a result, the software can be compiled and installed from its source code in four steps:

- configure
- make
- make check
- make install

The first step lays the proper groundwork for all following steps. The second command builds the actual binaries according to the rules determined in the first step. The third step optionally tests the created binaries while the last instruction copies all files to their final destinations. Usually, the configure script provides some informations on the available command line parameters by issuing

[43] https://en.wikipedia.org/wiki/Configure_script

```
./configure --help
```

You want to look out for something like 'prefix' which usually describes the directory where all files will finally be installed. Thus, simply create your own application installation directory and let `--prefix=$HOME/my_app_directory` hint to this directory.

For `cmake`-based[44] build systems, you can choose a different installation location by passing the command line option `-DCMAKE_INSTALL_PREFIX:PATH=$HOME/my_app_directory`.

Afterwards, `make install` should install all necessary files (including binaries, libraries and manpages) under this directory in your home or work directory. Note that you must not use the common phrase `sudo make install` but rather just `make install`.

If the configure script has no means to change the installation directory, it is often suficcient to stop after the `make check` step and use the binary created in the build directory as is. If its not in the top-most directory, look out for something called `bin` or a `build` subdirectory.

Finally you might want to add the installation directory to your `$PATH` environment variable.

4.3.6.2 pip

`pip` is a widespread tool to install additional Python modules. To install these modules into your home diretory, you need to use the parameter `--user` on every `pip` call:

```
pip3 install --user package_name
```

4.4 Resource management

LiDO3 uses the *Slurm Workload Manager*[45] to control batch jobs and cluster resources. *Slurm* takes care of running the users' jobs on allocated nodes and keeps track of the users' processes. Processes that are started directly on individual nodes – circumventing the queuing system – are immediately killed without further notification.

[44] https://cmake.org/
[45] https://slurm.schedmd.com/

Slurm comes with a built-in scheduling system with the purpose of finding and allocating the necessary resources for a user's job and organizes the usage between different users and jobs taking scheduling policies, dynamic priorities, reservations, and fairshare capabilities into account.

The entities managed by Slurm include:

nodes are the compute resource in Slurm.

partitions group nodes into logical – possibly overlapping – sets.

jobs allocate resources – inside a partition – to a user for a specified amount of time.

job steps are sets of – possibly parallel – tasks within a job (see page 101).

tasks The actual runing code.

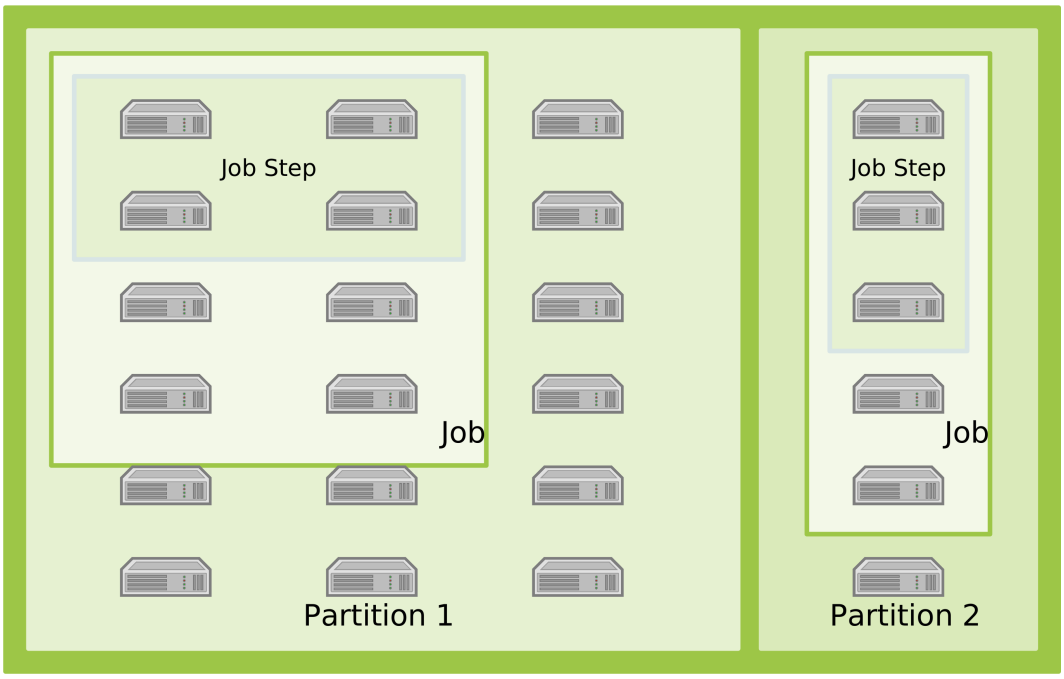


Figure 4.19: Slurm entities.

"The partitions can be considered job queues, each of which has an assortment of constraints such as job size limit, job time limit, users permitted to use it, etc. Priority-ordered jobs are allocated nodes within a partition until the resources (nodes, processors, memory, etc.) within that partition are exhausted. Once a job is assigned a set of nodes, the user is able to initiate parallel work in the form of job steps in any

configuration within the allocation. For instance, a single job step may be started that utilizes all nodes allocated to the job, or several job steps may independently use a portion of the allocation."

— Quoted from Slurm Quick Start User Guide

4.4.1 Partition

There are different *partitions* available on the LiDO3 cluster.

Table 4.2: Standard partitions

Queue	max. walltime	remarks
short	02:00:00	—
med	08:00:00	—
long	2-00:00:00	—
ultralong	28-00:00:00	no GPU or "non-blocking" nodes
testpart	02:00:00	use when instructed by LiDO3 administrators

Table 4.3: Partitions with faculty hardware

Queue	max. walltime	remarks
ext_phy_prio	28-00:00:0	Xeon Phi “KNL”
ext_phy_norm	02:00:0	Xeon Phi “KNL”
ext_iom_prio	28-00:00:0	members group `iom` only
ext_iom_norm	02:00:00	—
ext_trr188	28-00:00:0	members group `tr188` only
ext_vwl_prio	28-00:00:0	members group `vwl` only
ext_vwl_norm	02:00:0	—
ext_math_prio	28-00:00:0	members group `math` only
ext_math_norm	02:00:0	—
ext_chem_prio	28-00:00:0	members group `chem` only
ext_chem_norm	02:00:0	—
ext_biochem_prio	28-00:00:0	members group `kayserlab` only
ext_biochem_norm	02:00:0	—

The command `sinfo` provides an overview over the partitions.

```
[redacted]@gw01:~
[redacted]@gw01 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
short*       up    2:00:00      1  down* cquad02-002
short*       up    2:00:00     39  alloc cgpu01-[007-020],cstd01-[021-023,029-03
1,033,044,046-047,065,071-074,081,085,090-095,225,232]
short*       up    2:00:00    326   idle cgpu01-[001-006],cquad01-[001-028],cqua
d02-001,cstd01-[001-020,024-028,032,034-043,045,048-064,066-070,075-080,082-084,
086-089,096-224,226-231,233-244],cstd02-[001-072]
med          up    8:00:00      1  down* cquad02-002
med          up    8:00:00     39  alloc cgpu01-[007-020],cstd01-[021-023,029-03
1,033,044,046-047,065,071-074,081,085,090-095,225,232]
med          up    8:00:00    292   idle cgpu01-[005-006],cquad01-[004-028],cqua
d02-001,cstd01-[024-028,032,034-043,045,048-064,066-070,075-080,082-084,086-089,
096-224,226-231,233-244],cstd02-[008-072]
long         up 2-00:00:00      1  down* cquad02-002
long         up 2-00:00:00     29  alloc cgpu01-[007-020],cstd01-[065,071-074,08
1,085,090-095,225,232]
long         up 2-00:00:00    242   idle cquad01-[007-028],cquad02-001,cstd01-[0
61-064,066-070,075-080,082-084,086-089,096-224,226-231,233-244],cstd02-[023-072]
ultralong    up 28-00:00:0      2  alloc cstd01-[225,232]
ultralong    up 28-00:00:0     76   idle cquad01-[025-028],cstd01-[171-224,226-2
31,233-244]
[redacted]@gw01 ~]$
```

Figure 4.20: Gathering information about the partitions.

4.4.2 Working with partitions

Working with jobs is done by using *Slurm commands* that describe the resource characteristics of the job, e.g. number of nodes, processor cores needed and *Walltime*. This can be done interactively from the shell or in a *job script*.

To start a job in *Slurm*, it must be put into a *Partition*. This is done with one of these three commands:

srun *"Run a parallel job on cluster managed by Slurm. If necessary, `srun` will first create a resource allocation in which to run the parallel job."*

— Quoted from the `srun` manpage.

`srun` is typically used to start *jobsteps* inside a shell script that was launched with `sbatch`. This way the code for preparing the job and clean-up afterwards can run even if a job is terminated.

sbatch *"`sbatch` submits a batch script to Slurm. The batch script may be given to `sbatch` through a file name on the command line, or if no file name is specified, `sbatch` will read in a script from standard input. The batch script may contain options preceded with "`#SBATCH`" before any executable commands in*

the script.
`sbatch` exits immediately after the script is successfully transferred to the Slurm controller and assigned a Slurm job ID. The batch script is not necessarily granted resources immediately, it may sit in the queue of pending jobs for some time before its required resources become available.
By default both standard output and standard error are directed to a file of the name `"slurm-%j.out"`, where the `"%j"` is replaced with the job allocation number. The file will be generated on the first node of the job allocation. Other than the batch script itself, Slurm does no movement of user files.

When the job allocation is finally granted for the batch script, Slurm runs a single copy of the batch script on the first node in the set of allocated nodes.

When the job allocation is finally granted for the batch script, Slurm runs a single copy of the batch script on the first node in the set of allocated nodes. "

— Quoted from the `sbatch` manpage.

salloc *"`salloc` - Obtain a Slurm job allocation (a set of nodes), execute a command, and then release the allocation when the command is finished."*

— Quoted from the `salloc` manpage.

Partitions with long configured walltimes are popular from the users view but on the other hand they are somehow an unloved child from the cluster administrators perspective.

- When you as a user put a job inside a partition with a long configured walltime, chances are high that you have to wait quite a long time before your job gets even started. Statistics teaches us that the average waiting time is half of the maximum configured walltime per partition.
- The same goes for maintenance windows. We have to drain those partitions (i.e. starting of jobs is prohibited, submissions of jobs is still possible) very early to make sure that not too many jobs are still running when we shut down the cluster. All jobs still running need to be canceled when the maintenance starts. Closing those partitions early can have a negative impact on the utilization of the cluster: with long running jobs ending one by one and no new long running jobs being allowed to start, compute nodes may become idle if not enough requests are made for partitions with shorter maximum walltimes that are still open.

- In case of an emergeny shutdown of the cluster all currently running jobs will get canceled. This, obviously, translates to data loss for all those jobs. In a worst case scenario all calculated data from long running jobs gets lost maybe just a few minutes before its planned end of runtime.

  This is no mere theoretical risk, unscheduled emergency downtimes have happened before.

So, please consider to use checkpointing in your jobs and in your code in a way that enables you to restart a canceled job and resume the work after the last checkpoint.

And while you are at it, think about breaking your long running job up in to smaller parts that can run one after another in a partition with a shorter maximum walltime. Best aim for under two hours, so your job(s) will fit in the **short** partition.

LiDO4 will – like most HPC clusters – probably not provide partitions with a walltime greater than 24 hours.

4.4.2.1 `srun` - interactive execution and jobsteps

Slurm offers the possibility to execute jobs interactively. Execution of `srun` with the command line option `--pty bash` results in *Slurm* reserving the requested node – by using `salloc` under the hood (see page 60) – and starts `bash` on that node with a login prompt due to the `--pty` option and waits for its execution. Since no partition was given, the default `short` ist used. The user can then start his program from that interactive shell.

Example session:

```
[<username>@gw01 ~]$ srun --pty bash
[<username>@cstd01-214 ~]$ echo $SLURM_TASK_PID
163545
[<username>@cstd01-214 ~]$ exit
[<username>@gw01 ~]$
```

As soon as the walltime is exceeded, the shell is automatically terminated!

Other options to `srun` include number of nodes, *Walltime* etc., see also section *Slurm statements*.

Example session with 4 nodes and 3 tasks per node:

```
[<username>@gw01 ~]$ srun --nodes=4 --ntasks-per-node=3 --pty bash
[<username>@cstd01-214 ~]$ echo $SLURM_TASK_PID
166178
[<username>@cstd01-214 ~]$ exit
exit
[<username>@gw01 ~]$
```

If the `--pty` option is omitted, no login prompt will be given and any input will get run $12 = (\text{–nodes}{=}4) \times (\text{–ntasks-per-node}{=}3)$ times.

Example session with multiple times:

```
[<username>@gw01 ~]$ srun --nodes=4 --ntasks-per-node=3 bash
# there is no prompt, so enter blindly:
echo $SLURM_TASK_PID
121395
104316
105574
167463
121396
104317
121397
104318
167464
105575
167465
105576
exit
[<username>@gw01 ~]$
```

The following shell script `demoscript.sh` is used to start a job non-interactive:

```
#!/bin/bash -l
echo "START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID
    ↪ $SLURM_TASK_PID) on $SLURMD_NODENAME"
sleep 60
echo "STOP on $SLURMD_NODENAME"
```

Example session:

```
[<username>@gw01 ~]$ srun --nodes=2 --tasks-per-node=4
    ↪ demoscript.sh
START SLURM_JOB_ID 10894 (SLURM_TASK_PID 173171) on cstd01-214
```

```
START SLURM_JOB_ID 10894 (SLURM_TASK_PID 126888) on cstd01-215
START SLURM_JOB_ID 10894 (SLURM_TASK_PID 173173) on cstd01-214
START SLURM_JOB_ID 10894 (SLURM_TASK_PID 126889) on cstd01-215
START SLURM_JOB_ID 10894 (SLURM_TASK_PID 173174) on cstd01-214
START SLURM_JOB_ID 10894 (SLURM_TASK_PID 126891) on cstd01-215
START SLURM_JOB_ID 10894 (SLURM_TASK_PID 173172) on cstd01-214
START SLURM_JOB_ID 10894 (SLURM_TASK_PID 126890) on cstd01-215
STOP on cstd01-214
STOP on cstd01-214
STOP on cstd01-214
STOP on cstd01-215
STOP on cstd01-214
STOP on cstd01-215
STOP on cstd01-215
STOP on cstd01-215
[<username>@gw01 ~]$
```

Note that the execution with `srun` blocks your session. Only after `demoscript.sh` is run $8 = (--nodes = 2) \times (--ntasks-per-node = 4)$ times, you return to your login prompt.

If you close your SSH session, all jobs started by `srun` – directly from your shell – will be terminated!

4.4.2.2 `sbatch` - Submit a job script

If don't want to submit your jobs details by hand and stay in front of the terminal everytime, you can wrap the needed information into a *job script* for later execution. A *job script* is basically a shell script that contains *Slurm* statements in the header section. The rest of the script is code that should be executed AKA *the job* itself.

```
#!/bin/bash -l

#SBATCH --partition=short
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=3
#SBATCH --time=2:00
#SBATCH --mem-per-cpu=100
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/job.out.txt
...some code...
```

A script can be submitted to the batch system with the command `sbatch`, followed by `<SCRIPT_NAME>`. By using `salloc` under the hood (see page 60) the requested nodes are reserved and used for job execution.

```
sbatch my_submit_script.sh
```

Example of a job script:

```
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=3
#SBATCH --time=0:30
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID \
(SLURM_TASK_PID $SLURM_TASK_PID) on $SLURMD_NODENAME"
echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"
echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"
srun /home/<username>/workerscript.sh &
wait
echo "sbatch: STOP"
```

The job script spawns 12 job steps, each calling `workerscript.sh`:

```
#!/bin/bash -l
echo "worker ($SLURMD_NODENAME): start"
echo "executing SLURM_JOB_ID $SLURM_JOB_ID \
(SLURM_TASK_PID $SLURM_TASK_PID) \
on $SLURMD_NODENAME"
sleep 10
echo "worker ($SLURMD_NODENAME): stop"
```

Executing the job script:

```
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 11283

# waiting 10 seconds

[<username>@gw01 ~]$ cat /work/<username>/demo.out.txt
```

```
sbatch: START SLURM_JOB_ID 37170 (SLURM_TASK_PID 68044) on
    ↪ cstd01-205
sbatch: SLURM_JOB_NODELIST cstd01-[205-208]
sbatch: SLURM_JOB_ACCOUNT itmc
worker (cstd01-206): start
worker (cstd01-208): start
worker (cstd01-205): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 68077) on cstd01-205
worker (cstd01-207): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 66025) on cstd01-206
worker (cstd01-208): start
worker (cstd01-205): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 68078) on cstd01-205
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 72998) on cstd01-207
worker (cstd01-206): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 82054) on cstd01-208
worker (cstd01-205): start
worker (cstd01-207): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 66026) on cstd01-206
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 82053) on cstd01-208
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 68079) on cstd01-205
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 72999) on cstd01-207
worker (cstd01-206): start
worker (cstd01-208): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 82055) on cstd01-208
worker (cstd01-207): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 66027) on cstd01-206
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 73000) on cstd01-207
worker (cstd01-206): stop
worker (cstd01-208): stop
worker (cstd01-205): stop
worker (cstd01-207): stop
worker (cstd01-208): stop
worker (cstd01-206): stop
worker (cstd01-206): stop
worker (cstd01-208): stop
worker (cstd01-205): stop
worker (cstd01-205): stop
worker (cstd01-207): stop
worker (cstd01-207): stop
sbatch: STOP
```

Due to race conditions, the order is not predictable.

If you need to use third party software in your job script that is available via the module system, see section *Modules in job scripts* on page 46.

If the path to the output file does not exist or can not be written to (e.g. points outside of `/work`), the Slurm job will seemingly fail silently (unless mail notification is enabled). One can query the Slurm database explicitly for such failed jobs with `sacct --starttime=HH:MM --state=FAILED`.

4.4.2.3 `salloc` - Allocate nodes

Resources for a job can be allocated in real time with the command `salloc`. Those allocated resources are typically used to spawn a shell and – interactively – execute `srun` commands to launch parallel tasks.

Whereas `srun` uses `salloc` under the hood to acquire the needed resources, using `salloc` as a discrete command enables you to initiate different *job steps* inside an allocated set of nodes.

To allocate 10 nodes using the `--exclusive` option so no other users will be running jobs on the allocated nodes at the same time, enter

```
[<username>@gw01 ~]$ salloc --nodes=10 --exclusive
salloc: Granted job allocation 14008
salloc: Waiting for resource configuration
salloc: Nodes cstd01-[001-010] are ready for job
```

Now we will start 3 job steps on those 10 allocated nodes:

1. using 2 nodes (`--nodes=2`) starting with the first node (`--relative=0`) of the allocated range.
2. using 4 nodes (`--nodes=4`) starting with the third node (`--relative=2`) of the allocated range.
3. using 2 nodes (`--nodes=4`) starting with the seventh node (`--relative=6`) of the allocated range.

```
[<username>@gw01 ~]$ srun --nodes=2 --relative=0 --jobid=14008
    ↪ /usr/bin/sleep 300&
[<username>@gw01 ~]$ srun --nodes=4 --relative=2 --jobid=14008
    ↪ /usr/bin/sleep 300&
[<username>@gw01 ~]$ srun --nodes=4 --relative=6 --jobid=14008
    ↪ /usr/bin/sleep 300&
```

Since no `--time` option was used with `salloc`, the allocation will last as long as the timelimit of the partition. Further job steps can be initiated during that timespan.

Allocations can also be used to start a session with the X Window System.

```
[<username>@gw01 ~]$ salloc --nodes=1 --exclusive
    ↪ --constraint=cstd01
salloc: Granted job allocation 14037
salloc: Waiting for resource configuration
salloc: Nodes cstd01-003 are ready for job
[<username>@gw01 ~]$ ssh -X cstd01-003
Warning: Permanently added 'cstd01-003,10.10.3.3' (ECDSA) to the
    ↪ list of known hosts.
[<username>@cstd01-003 ~]$ start-my-x-program
[<username>@cgpu01-003 ~]$ exit
logout
Connection to cgpu01-003 closed.
[<username>@gw01 ~]$ scancel 14037
[<username>@gw01 ~]$ salloc: Job allocation 14037 has been
    ↪ revoked.
```

4.4.2.4 `scontrol`, `squeue`, `showq` - Query Job status

The status of each *Slurm* job can be queried with `scontrol show job <job_id>` and `squeue`.

```
[<username>@gw01 ~]$ scontrol show job 11283
JobId=11283 JobName=demoscript
   UserId=<username>(<uid>) GroupId=<username>(<gid>)
   ↪ MCS_label=N/A
   Priority=21149 Nice=0 Account=itmc QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:47 TimeLimit=00:02:00 TimeMin=N/A
   SubmitTime=2017-08-11T14:20:13 EligibleTime=2017-08-11T14:20:13
   StartTime=2017-08-11T14:20:13 EndTime=2017-08-11T14:22:13
   ↪ Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=short AllocNode:Sid=gw01:60481
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=cstd01-[001-004]
   BatchHost=cstd01-001
```

```
   NumNodes=4 NumCPUs=12 NumTasks=12 CPUs/Task=1
   ↪ ReqB:S:C:T=0:0:*:*
   TRES=cpu=12,mem=1200M,node=4
   Socks/Node=* NtasksPerN:B:S:C=3:0:*:* CoreSpec=*
   MinCPUsNode=3 MinMemoryCPU=100M MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   Gres=(null) Reservation=(null)
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/<username>/my_submit_script.sh
   WorkDir=/home/<username>
   StdErr=/work/<username>/demo.out.txt
   StdIn=/dev/null
   StdOut=/work/<username>/demo.out.txt
   Power=
```

and `squeue` respectively.

Example session to get all own jobs:

```
[<username>@gw01 ~]$ squeue -u $USER
JOBID PARTITION     NAME      USER  ST  TIME  NODES
   ↪ NODELIST(REASON)
14004     short demoscri <username>  R  0:03      2
   ↪ cgpu01-[002-003]
13977     short     bash <username>  R  9:36      1 cgpu01-001
13978       med  glidein <username>  R  8:08      1 cstd01-021
#                                    ^
#                                    |
#                                    |
#                                  R = running
#                                 PD = pending
```

Example session to get information for a specific job:

```
[<username>@gw01 ~]$ squeue --jobs=14005
JOBID PARTITION     NAME       USER ST  TIME  NODES
   ↪ NODELIST(REASON)
14005     short demoscri <username>  R  0:04      2
   ↪ cgpu01-[002-003]
```

Example session to get information for a specific job including job steps:

```
[<username>@gw01 ~]$ squeue --job=14008 --steps
STEPID       NAME  PARTITION       USER  TIME NODELIST
```

```
14008.0       sleep     short <username>  0:35 cstd01-[001-002]
14008.1       sleep     short <username>  0:23 cstd01-[003-006]
14008.2       sleep     short <username>  0:13 cstd01-[007-010]
14008.Extern extern     short <username>  2:09 cstd01-[001-010]
```

Example session to get estimated starting time for all own jobs:

```
[<username>@gw01 ~]$ squeue --start -u $USER
JOBID PARTITION     NAME       USER ST  START_TIME          NODES
   ↪ NODELIST(REASON)
14005     short demoscri <username>  PD  2015-10-15T16:36:49  2
   ↪ (Ressources)
```

The estimated starting time needs to be taken with a ~~grain~~ truckload of salt: The slurm scheduler has to solve an NP-hard problem in optimising the cluster usage:

- The cluster should always be fully utilized. This is particulary achieved via backfilling, i.e. to start jobs with a smaller priority to use the reserved job slots, as long as these jobs do not delay the start of another job.
- Large jobs require the cluster to be nearly empty to start.
- The runtime estimates users provide in their SLURM job files (using the options `-t` or `--time`) are often not very accurate, typically they largely overestimate the actual runtime. Unanticipated program abortions (node failures, codes crashing etc.) completely thwart any prognosis the scheduler has come up with before about when compute nodes become idle.
- Arbitrary nodes may need to be drained for unplanned maintenance (for hardware repairs or to install critical security fixes)

That said, your average waiting time will be smaller if the total amount of computational time (number of computes cores times the wall clock time) is less. The lesser resources you request, the higher your job gets prioritised which – ignoring the backfilling mechanism – leads to the job getting started quicker. Hence:

- Do not simply request the maximum time limit a particular partitions allows if you know beforehand that your job will need less. E.g. do not ask for 28 days in partition **ultralong** if you know that your simulation will finish with 4-5 days.

- Statistics teaches us that the average waiting time for a particular partition is, in general, half the maxium time limit of said partition. Hence, your average waiting time will be, in comparison to the waiting times in partitions **large** or **ultralong**, much smaller if you use the **short** or **med** partition where possible.
- The fewer compute cores you request, the more likely your Slurm job will start.
- If applicable, do not request compute nodes exclusively such that compute nodes do not need to be completely drained for your job to start.

The third-party tool showq[46] mimics the functionality of the PBS/Torque tool `showq`. In particular, it gives a good sorted overview about all jobs and their respective status.

Example session to get all your jobs:

```
gw02: ~>$ showq -u $USER

SUMMARY OF JOBS FOR USER: <smdiribb>
ACTIVE JOBS--------------------
JOBID     JOBNAME    USERNAME      STATE   CORE   REMAINING
   ↪ STARTTIME
===============================================================================

WAITING JOBS------------------------
JOBID     JOBNAME    USERNAME      STATE   CORE     WCLIMIT
   ↪ QUEUETIME
===============================================================================
11048282  OSU        smdiribb      Waiting 2        0:15:00  Thu
   ↪ Jun  4 10:46:54
11566299  feat       smdiribb      Waiting 16       8:00:00  Thu
   ↪ Aug 13 00:30:01

Total Jobs: 2    Active Jobs: 0     Idle Jobs: 2    Blocked Jobs:
   ↪ 0
```

4.4.2.5 `scancel` - Cancel a queued job

A *Slurm* job can be removed from the job queue via `scancel <job_id>`.

```
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 11284
[<username>@gw01 ~]$ scancel 11284
[<username>@gw01 ~]$ scontrol show job 11284
JobId=11284 JobName=demoscript
```

[46] https://github.com/fasrc/slurm_showq

```
   UserId=<username>(<uid>) GroupId=<username>(<gid>)
   ↪ MCS_label=N/A
   Priority=21158 Nice=0 Account=itmc QOS=normal
   JobState=CANCELLED Reason=None Dependency=(null)
```

4.4.2.6 Decreasing job priority with `scontrol`, `sbatch`

You can manually decrease the job's priority by increasing the so-called nice value of a pending job. This can be appropriate if some of your jobs are not critical in terms of time, e.g. cleanup tasks. As it is very hard to estimate the effect of some specific nice value setting one usually goes all in and sets the nice value to the maximum possible value: 2147483645.
The nice value can be set at job submission via

```
sbatch --nice=2147483645 myjobscript.slurm
```

or afterwards via

```
scontrol update job myjobid nice=2147483645
```

4.4.2.7 `seff`, `sacct` - show post job performance analysis

In order to be able to see for yourself whether your job has efficently used the allocated ressources, the tool *seff* is available on LiDO3. Using this tool, one can run a short analysis on completed *Slurm* jobs. *seff* takes a the job ID as argument, example usage: `seff 12345`. Note that for job arrays, the full job ID is required, i.e. for example `seff 12345_7`, otherwise *seff* processes only the last array entry.

```
gw01: ~>$ seff 11401523
Job ID: 11401523
Cluster: lido3
User/Group: smdiribb/smdiribb
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 20
CPU Utilized: 00:03:54
CPU Efficiency: 2.79% of 02:19:40 core-walltime
Job Wall-clock time: 00:06:59
Memory Utilized: 376.64 MB
Memory Efficiency: 0.61% of 60.00 GB
```

The product of 'Nodes' and 'Cores per node' is the allocated CPU core number. In this example 2*20=40. The CPU time is the product of the 'Job Wall-clock time' and the number of cores. If the resulting 'CPU efficiency' is much smaller than 100%, there may be several reasons for this:

- the application used fewer cores than the allocated amount of cores;
- the application used all cores for part of the time, but not all cores were used for a significant period of time;
- the application is limited by memory size or memory transfer speed and thus CPU usage is no meaningful metric at all.

On the other hand is a high cpu efficiency not unconditionally equivalent to an optimal ressource usage. It can happen, that your applications starts a huge amount of threads (sometime hundreds) and thus the operationg system is busy switching contexts and your own application does not get cpu time at all. Despite your application making no progress, *seff* will asure you a high cpu efficiency.

Another approach is to use sacct for information gathering.

```
gw01: ~>$ sacct  --format="CPUTime,AveCPU,MaxDiskWrite" -j
    ↪ 11401523
    CPUTime     AveCPU MaxDiskWrite
    ---------- ---------- ------------
    02:19:40
    02:19:40   00:03:50       82.88M
    02:20:00   00:00:00            0
```

A complete list of possible data can be retrieved by

```
gw01: ~>$ sacct -e
Account             AdminComment        AllocCPUS
    ↪ AllocGRES
AllocNodes          AllocTRES           AssocID             AveCPU
AveCPUFreq          AveDiskRead         AveDiskWrite
    ↪ AvePages
AveRSS              AveVMSize           BlockID
    ↪ Cluster
Comment             Constraints         ConsumedEnergy
    ↪ ConsumedEnergyRaw
CPUTime             CPUTimeRAW          DBIndex
    ↪ DerivedExitCode
Elapsed             ElapsedRaw          Eligible            End
ExitCode            Flags               GID                 Group
```

```
JobID                JobIDRaw             JobName              Layout
MaxDiskRead          MaxDiskReadNode      MaxDiskReadTask
    ↪ MaxDiskWrite
MaxDiskWriteNode     MaxDiskWriteTask     MaxPages
    ↪ MaxPagesNode
MaxPagesTask         MaxRSS               MaxRSSNode
    ↪ MaxRSSTask
MaxVMSize            MaxVMSizeNode        MaxVMSizeTask
    ↪ McsLabel
MinCPU               MinCPUNode           MinCPUTask           NCPUS
NNodes               NodeList             NTasks
    ↪ Priority
Partition            QOS                  QOSRAW               Reason
ReqCPUFreq           ReqCPUFreqMin        ReqCPUFreqMax
    ↪ ReqCPUFreqGov
ReqCPUS              ReqGRES              ReqMem
    ↪ ReqNodes
ReqTRES              Reservation          ReservationId
    ↪ Reserved
ResvCPU              ResvCPURAW           Start                State
Submit               Suspended            SystemCPU
    ↪ SystemComment
Timelimit            TimelimitRaw         TotalCPU
    ↪ TRESUsageInAve
TRESUsageInMax       TRESUsageInMaxNode   TRESUsageInMaxTask
    ↪ TRESUsageInMin
TRESUsageInMinNode   TRESUsageInMinTask   TRESUsageInTot
    ↪ TRESUsageOutAve
TRESUsageOutMax      TRESUsageOutMaxNode  TRESUsageOutMaxTask
    ↪ TRESUsageOutMin
TRESUsageOutMinNode  TRESUsageOutMinTask  TRESUsageOutTot      UID
User                 UserCPU              WCKey
    ↪ WCKeyID
WorkDir
```

4.4.3 Constraints on node-features

The LiDO3-Team has assigned so-called *features* to the different nodes in the LiDO3-cluster. Those features can specifically requested with the `--constraint` parameter of the `srun`, `sbatch` and `salloc` commands.

Table 4.4: List of features.

Nodelist	Features	CPU Type GPU Type	max. cores	max. memory
cgpu01-[001-020]	all public cgpu01 xeon_e52640v4 gpu tesla_k40 ib_1to3	2 × Intel®Xeon E5 2640v4 2.4 GHz, L3 cache 25 MB 2 × Nvidia®Tesla K40	20	64 GB MaxMemPerNode=64300
cgpu02-[001-002]	all public cgpu02 xeon_e52690v4 p100 gpu tesla_p100 ib_1to3	2 × Intel®Xeon E5 2690v4 2.4 GHz, L3 cache 25 MB 1 × Nvidia®Tesla P100	28	256 GB MaxMemPerNode=257800
cknl01-[001-003]	all private cknl01 xeon_phi7210 ib_1to3	1 × Intel®Xeon Phi “KNL”7210	64	192 GB MaxMemPerNode=209400
cquad01-[001-028]	all public cquad01 xeon_e54640v4 ib_1to3	4 × Intel®Xeon E5 4640v4 2.1 GHz, L3 cache 30 MB	48	256 GB MaxMemPerNode=257800
cquad02-[001-002]	all public cquad02 xeon_e54640v4 ib_1to3	4 × Intel®Xeon E5 4640v4 2.1 GHz, L3 cache 30 MB	48	1024 GB MaxMemPerNode=1031900
cquad03-[001-002]	all public cquad03 xeon_gold_6230 ib_1to3	4 × Intel®Xeon Gold 6230 2.1 GHz, L3 cache 28 MB	80	512 GB MaxMemPerNode=498952

continued on next page . . .

. . . continued from previous page

Nodelist	Features	CPU Type GPU Type	max. cores	max. memory
cstd01-[001-244]	all public cstd01 xeon_e52640v4 ib_1to3	2 × Intel®Xeon E5 2640v4 2.4 GHz, L3 cache 25 MB	20	64 GB MaxMemPerNode=64300
cstd02-[001-072]	all public cstd02 xeon_e52640v4 ib_1to1 nonblocking_comm	2 × Intel®Xeon E5 2640v4 2.4 GHz, L3 cache 25 MB	20	64 GB MaxMemPerNode=64300
cstd03[001-004]	all public cstd03 xeon_e52690v4 ib_1to3	2 × Intel®Xeon E5 2690v4 2.4 GHz, L3 cache 35 MB	28	256 GB MaxMemPerNode=257800
cstd04[001-004]	all public cstd04 xeon_gold_6134 ib_1to3	2 × Intel®Gold 6134 CPU 2.4 GHz, L3 cache 24 MB	16	192 GB MaxMemPerNode=190000
cstd05[001-003]	all public cstd05 epyc_7542 ib_1to3	2 × AMD EPYC 7542 CPU 2.49 GHz, L3 cache 128 MB	64	1024 GB MaxMemPerNode=1031900
cstd06[001]	all public cstd06 xeon_gold_6242r ib_1to3	2 × Intel Xeon Gold 6242R CPU 3.1 GHz, L3 cache 35 MB	40	96 GB MaxMemPerNode=94500
cstd07[001]	all public cstd07 epyc_7542 ib_1to3	2 × AMD EPYC 7542 CPU 2.49 GHz, L3 cache 128 MB	64	256 GB MaxMemPerNode=257800

Example session for `srun`

```
[<username>@gw01 ~]$ srun --constraint=cstd01 --pty bash
[<username>@cstd01-019 ~]$ echo $SLURM_TASK_PID
166178
[<username>@cstd01-019 ~]$ exit
exit
```

Example session for `sbatch`

```
[<username>@gw01 ~]$ cat my_submit_script.sh
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=3
#SBATCH --time=2:00
#SBATCH --mem-per-cpu=100
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
#SBATCH --constraint=cstd01
srun echo "START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID
   ↪ $SLURM_TASK_PID) on $SLURMD_NODENAME"
srun echo "STOP on $SLURMD_NODENAME"

[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 13891
[<username>@gw01 ~]$ scontrol show job 13891
JobId=13891 JobName=demoscript
   UserId=<username>(<uid>) GroupId=<username>(<gid>)
   ↪ MCS_label=N/A
   Priority=28436 Nice=0 Account=itmc QOS=normal
   JobState=COMPLETED Reason=None Dependency=(null)
   (...)

#[<username>@gw01 ~]$ cat /work/<username>/demo.out.txt
START SLURM_JOB_ID 13891 (SLURM_TASK_PID 6217) on cstd01-019
START SLURM_JOB_ID 13891 (SLURM_TASK_PID 6217) on cstd01-019
START SLURM_JOB_ID 13891 (SLURM_TASK_PID 6217) on cstd01-019
START SLURM_JOB_ID 13891 (SLURM_TASK_PID 6217) on cstd01-019
START SLURM_JOB_ID 13891 (SLURM_TASK_PID 6217) on cstd01-019
START SLURM_JOB_ID 13891 (SLURM_TASK_PID 6217) on cstd01-019
STOP on cstd01-019
STOP on cstd01-019
STOP on cstd01-019
STOP on cstd01-019
STOP on cstd01-019
STOP on cstd01-019
```

```
[<username>@gw01 ~]$
```

As you can see with `man sbatch`, nodes can have *features* assigned to them by the Slurm administrator. Users can specify which of these *features* are required by their job using the constraint option. Only nodes having features matching the job constraints will be used to satisfy the request. Multiple constraints may be specified with `AND`, `OR`, matching `OR`, resource counts, etc. (some operators are not supported on all system types). Supported constraint options include:

Single Name Only nodes which have the specified feature will be used. For example, `--constraint="ib_1to1"`

Node Count A request can specify the number of nodes needed with some feature by appending an asterisk and count after the feature name. For example, `--nodes=16 ↪ --constraint=cstd01*4` indicates that the job requires 16 nodes and that at least four of those nodes must have the feature "cstd01."

AND If only nodes with all of specified features will be used. The ampersand is used for an AND operator. For example, `--constraint="xeon_e52640v4&gpu"`

OR If only nodes with at least one of specified features will be used. The vertical bar is used for an OR operator. For example, `--constraint="xeon_e52640v4|e54640v4"`

Matching OR If only one of a set of possible options should be used for all allocated nodes, then use the OR operator and enclose the options within square brackets. For example: `"--constraint=[rack1|rack2|rack3|rack4]"` might be used to specify that all nodes must be allocated on a single rack of the cluster, but any of those four racks can be used.

Multiple Counts Specific counts of multiple resources may be specified by using the AND operator and enclosing the options within square brackets. For example: `"--constraint=[rack1*2&rack2*4]"` might be used to specify that two nodes must be allocated from nodes with the feature of "rack1" and four nodes must be allocated from nodes with the feature "rack2".

4.4.4 Generic Resource (GRES) - request a GPU

Reserving a GPU node by using constraints (see page 67) is only one half of the story. Other users may be already using the GPU when your job starts on one of those nodes and they seem too valuable to use them just for CPU-bound tasks.

GPUs are defined as a *Generic Resorce* (short *GRES*) in *Slurm* and can be requested with the `--gres=gpu:tesla[:count]` option which is supported by the `salloc`, `sbatch` and `srun` commands. Where `count` specifies how many resources are required and has a default value of `1`.

- For the 20 nodes with 2 GPU NVIDIA® K40 GPUs each, `count` has a valid maximum of **2**.
- For the 2 nodes with 1 GPU NVIDIA® P100 GPU each, `count` has a valid maximum of **1**.

Each K40 GPU is bound to one CPU socket. Thus an allocation of more than 10 CPU cores and more than 1 GPU goes side by side. It is actually not possible to allocate 11 or more CPU cores without allocating both GPUs. This procedure is embedded to ensure that each GPU can be accessed by a process running on the corresponding CPU socket.
If one wants to use only one CPU socket and only one GPU, the *Slurm* parameter `--gres-flags=enforce-binding` ensures that only those CPU cores corresponding to the corresponding CPU socket are allocated.

```
# reserve 1 non-exclusive node and both GPUs on it
[<username>@gw01 ~]$ salloc --nodes=1 --gres=gpu:tesla:2
salloc: Granted job allocation 14037
salloc: Waiting for resource configuration
salloc: Nodes cgpu01-003 are ready for job
[<username>@gw01 ~]$ ssh -X cgpu01-003
Warning: Permanently added 'cgpu01-003,10.10.3.3' (ECDSA) to the
    ↪ list of known hosts.
[<username>@cgpu01-003 ~]$ module load nvidia/cuda/8.0
[<username>@cgpu01-003 ~]$ nvvp -data $WORK -configuration $WORK
```

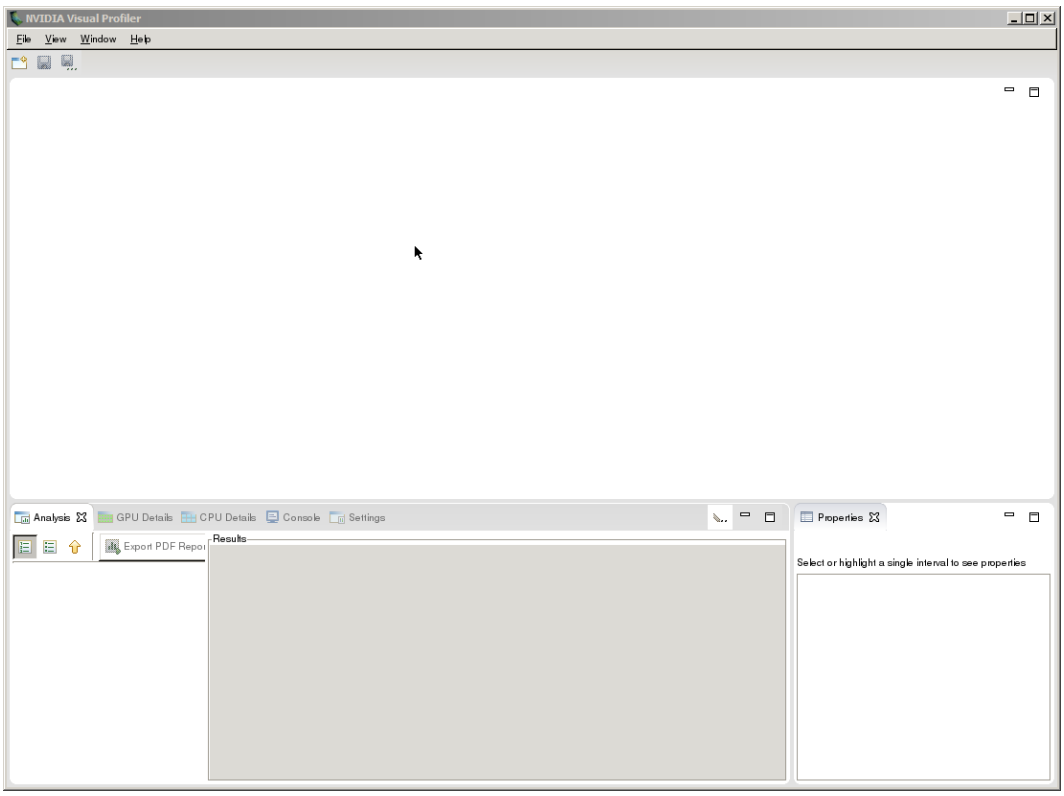


Figure 4.21: NVIDIA Visual Profiler on a Windows client.

```
[<username>@cgpu01-003 ~]$ exit
logout
Connection to cgpu01-003 closed.
[<username>@gw01 ~]$ scancel 14037
[<username>@gw01 ~]$ salloc: Job allocation 14037 has been
    ↪ revoked.
```

For each job step the environment variable `CUDA_VISIBLE_DEVICES` is set to determine which GPUs are available for its use on each node

Example script that is executed on GPU nodes:

```
#!/bin/bash -l
echo "worker ($SLURMD_NODENAME): start"
echo "executing SLURM_JOB_ID $SLURM_JOB_ID \
(SLURM_TASK_PID $SLURM_TASK_PID, \
CUDA_VISIBLE_DEVICES $CUDA_VISIBLE_DEVICES) \
on $SLURMD_NODENAME"
```

```
sleep 10
echo "worker ($SLURMD_NODENAME): stop"
```

Example batch script that is used to run `workerscript.sh` on each GPU node:

```
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=4
#SBATCH --exclusive
#SBATCH --gres=gpu:tesla:2
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID \
(SLURM_TASK_PID $SLURM_TASK_PID, \
CUDA_VISIBLE_DEVICES $CUDA_VISIBLE_DEVICES) \
on $SLURMD_NODENAME"
echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"
echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"
for RELATIVENODE in 0 1 2 3
do
   srun --nodes=1 \
        --relative=${RELATIVENODE} \
        --gres=gpu:tesla:$(($RELATIVENODE%2+1)) \
        --jobid=$SLURM_JOB_ID \
        /home/<username>/workerscript.sh &
done
wait
echo "sbatch: STOP"
```

Finally the excecution and output:

```
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 37141
[<username>@gw01 ~]$  cat /work/<username>/demo.out.txt
sbatch: START SLURM_JOB_ID 37171 (SLURM_TASK_PID 31707,
    ↪ CUDA_VISIBLE_DEVICES 0,1) on cgpu01-001
sbatch: SLURM_JOB_NODELIST cgpu01-[001-004]
sbatch: SLURM_JOB_ACCOUNT itmc
worker (cgpu01-004): start
executing SLURM_JOB_ID 37171 (SLURM_TASK_PID 8348,
    ↪ CUDA_VISIBLE_DEVICES 0,1) on cgpu01-004
worker (cgpu01-002): start
executing SLURM_JOB_ID 37171 (SLURM_TASK_PID 13088,
    ↪ CUDA_VISIBLE_DEVICES 0,1) on cgpu01-002
worker (cgpu01-003): start
```

```
executing SLURM_JOB_ID 37171 (SLURM_TASK_PID 8950,
    ↪ CUDA_VISIBLE_DEVICES 0) on cgpu01-003
worker (cgpu01-001): start
executing SLURM_JOB_ID 37171 (SLURM_TASK_PID 31755,
    ↪ CUDA_VISIBLE_DEVICES 0) on cgpu01-001
worker (cgpu01-002): stop
worker (cgpu01-004): stop
worker (cgpu01-001): stop
worker (cgpu01-003): stop
sbatch: STOP
```

Due to race conditions the order is not predictable. Although the option `#SBATCH ↪ --gres=gpu:tesla:2` is used, the number of GPUs must be expliticly required. The script alternated between `--gres=gpu:tesla:1` and `--gres=gpu:tesla:2` for every `srun`-call to show that effect

4.4.5 Memory management

Slurm monitors memory usage of a job in two different flavours:

- memory usage per node
- memory usage per core

Only one limit can be active at any time. If a job exceeds this limit, it is immediately abborted. The larger the data processed by your job, the larger this limit needs to be. The lower you set this limit, the easier it is for the *Slurm scheduler* to find a place for your job to run in the partition. The maximum upper limit per node (MaxMemPerNode) can be seen in table 4.4 on page 68. The maximum upper limit per core can be derived with the inequality

$$cpus - per - task \times mem - per - cpu < MaxMemPerNode$$

The number of cores times the memory per core must not exceed the maximum upper limit (MaxMemPerNode).

If no limit is provided by the job, a memory limit per core is set to DefMemPerCPU = 512 per node (512MB per core). If a job uses more than that, it is terminated with *job Exceeded job memory limit* error message.

You can set a larger limit per core by using the `--mem-per-cpu <memory>` option, where `<memory>` is the limit in `MB` — different units can be specified by using the suffix `[K|M|G|T]`.

```
[<username>@gw01 ~]$ cat my_submit_script.sh
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=3
#SBATCH --time=2:00
#SBATCH --mem-per-cpu=500M
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
#SBATCH --constraint=cstd01
srun echo "START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID
    ↪ $SLURM_TASK_PID) on $SLURMD_NODENAME"
srun sleep 30
srun sleep 30
srun sleep 30
srun echo "STOP on $SLURMD_NODENAME"
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 16571
```

If you are not sure what a good setting would be, you can try to determine an appropriate value by starting your job with a short runtime and a relatively large memory limit and then use the `sacct` command to monitor how much your job is actually using or has used.

```
[<username>@gw01 ~]$ sacct --format MaxRSS --job=16571
    MaxRSS
----------


      284K
       88K
       92K

[<username>@gw01 ~]$
```

To set the alternative limit for the full node memory consumption, one uses the `--mem <memory>` option, where `<memory>` is the limit in `MB` — different units can be specified by using the suffix `[K|M|G|T]`. The maximum upper limit per node (MaxMemPerNode) can be seen in table 4.4 on page 68.

Example session:

```
[<username>@gw01 ~]$ cat my_submit_script.sh
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=3
#SBATCH --time=2:00
#SBATCH --mem=500M
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
#SBATCH --constraint=cstd01
srun echo "START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID
    ↪ $SLURM_TASK_PID) on $SLURMD_NODENAME"
srun sleep 30
srun sleep 30
srun sleep 30
srun echo "STOP on $SLURMD_NODENAME"
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 16572
```

If you are not sure what a good setting would be, you can try to determine an appropriate value by starting your job with a short runtime and a relatively large memory limit and then use the `sacct` command to monitor how much your job is actually using or has used.

Example session:

```
[<username>@gw01 ~]$ sacct --format MaxRSS --job=16572
    MaxRSS
----------
       84K
[<username>@gw01 ~]$
```

The output is in KB, so divide by 1024 to get a rough idea of what setting to use with `--mem` (since you're defining a hard upper limit, round up that number a little bit). You can tell `sacct` to look further back in time by adding a start time with `--starttime YYYY-MM-DD` if your job ran too far in the past.

```
[<username>@gw01 ~]$ sacct --format MaxRSS --job=16572 \
--starttime 2017-08-23
    MaxRSS
----------
     3512K
         0
```

```
        84K
        92K
        92K
        92K
        84K
[<username>@gw01 ~]$
```

The `--mem` options sets the *maximum memory* used on any one node running your job parallel spanning multiple nodes; to get an even distribution of tasks per node, you can use run using the `--ntasks-per-node` option, otherwise the same job could have very different values when run at different times.

A memory size specification of zero is treated as a special case and grants the job access to all of the memory on each node. If multiple nodes with different memory layout are allocated for your job in the LiDO3 cluster, the node with the smallest memory size in the allocation defines the memory limit for each node of the allocation — the same limit will apply to every node.

The `--mem` option and the the `--mem-per-cpu` option are **mutually exclusive**!

4.4.6 Utilize complete nodes

If a user submits a job, it is very well possible that other jobs will run on the same nodes. To make a reservation for a complete node, use a `--exlusive` statement.

```
# Example reservation for 1 node:
[<username>@gw01 ~]$ salloc --nodes=1 --exclusive
salloc: Granted job allocation 140042
salloc: Waiting for resource configuration
salloc: Node cstd01-017 is ready for job
```

4.4.7 Slurm statements

Here is a non-exhaustive list of frequently used *Slurm* statements that can be used inside of a *job script* generated with help of `man sbatch`.

- `#SBATCH --job-name=<jobname>`

Specify a name for the job allocation. The specified name will appear along with the job ID number when querying running jobs on the system. The default is the name of the batch script, or just "sbatch" if the script is read on sbatch's standard input.

- `#SBATCH --nodes=<minnodes[-maxnodes]>`

  Request that a minimum of minnodes nodes be allocated to this job. A maximum node count may also be specified with maxnodes. If only one number is specified, this is used as both the minimum and maximum node count. The partition's node limits supersede those of the job. If a job's node limits are outside of the range permitted for its associated partition, the job will be left in a PENDING state. This permits possible execution at a later time, when the partition limit is changed. If a job node limit exceeds the number of nodes configured in the partition, the job will be rejected. Note that the environment variable `SLURM_NNODES` will be set to the count of nodes actually allocated to the job. If `-N` is not specified, the default behavior is to allocate enough nodes to satisfy the requirements of the `-n` and `-c` options. The job will be allocated as many nodes as possible within the range specified and without delaying the initiation of the job. The node count specification may include a numeric value followed by a suffix of "`k`" (multiplies numeric value by 1,024) or "`m`" (multiplies numeric value by 1,048,576).

- `#SBATCH --partition=<partition_names>`

  Request a specific partition for the resource allocation. If not specified, the default behavior is to allow the Slurm controller to select the default partition as designated by the system administrator. If the job can use more than one partition, specify their names in a comma separate list and the one offering earliest initiation will be used with no regard given to the partition name ordering (although higher priority partitions will be considered first). When the job is initiated, the name of the partition used will be placed first in the job record partition string.

- `#SBATCH --time=<time>`

  Set a limit on the total run time of the job allocation. If the requested time limit exceeds the partition's time limit, the job will be left in a `PENDING` state (possibly indefinitely). The default time limit is the partition's default time limit. When the time limit is reached, each task in each job step is sent `SIGTERM` followed by `SIGKILL`. The interval between signals is specified by the Slurm configuration parameter `KillWait`. (On LiDO3, `KillWait` is set to 30 s.) The `OverTimeLimit` configuration parameter may permit the job to run longer

than scheduled. (On LiDO3, `OverTimeLimit` is not configured.) Time resolution is one minute and second values are rounded up to the next minute. A time limit of zero requests that no time limit be imposed. Acceptable time formats include "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes" and "days-hours:minutes:seconds".

- `#SBATCH --output=<filename pattern>`

  Instruct Slurm to connect the batch script's standard output directly to the file name specified in the "filename pattern". By default both standard output and standard error are directed to the same file. For job arrays, the default file name is "`slurm-%A_%a.out`", "`%A`" is replaced by the job ID and "`%a`" with the array index. For other jobs, the default file name is "`slurm-%j.out`", where the "`%j`" is replaced by the job ID.

- `#SBATCH  --error=<filename pattern>`

  Instruct Slurm to connect the batch script's standard error directly to the file name specified in the "filename pattern". By default both standard output and standard error are directed to the same file. For job arrays, the default file name is "`slurm-%A_%a.out`", "`%A`" is replaced by the job ID and "`%a`" with the array index. For other jobs, the default file name is "`slurm-%j.out`", where the "`%j`" is replaced by the job ID.

- `#SBATCH --mail-type=<type>`

  Notify user by email when certain event types occur. Valid type values are `NONE`, `BEGIN`, `END`, `FAIL`, `REQUEUE`, `ALL` (equivalent to `BEGIN`, `END`, `FAIL`, `REQUEUE` and `STAGE_OUT`), `STAGE_OUT` (burst buffer stage out and teardown completed), `TIME_LIMIT`, `TIME_LIMIT_90` (reached 90 percent of time limit), `TIME_LIMIT_80` (reached 80 percent of time limit), `TIME_LIMIT_50` (reached 50 percent of time limit) and `ARRAY_TASKS` (send emails for each array task). Multiple type values may be specified in a comma separated list. The user to be notified is indicated with `--mail-user`. Unless the `ARRAY_TASKS` option is specified, mail notifications on job `BEGIN`, `END` and `FAIL` apply to a job array as a whole rather than generating individual email messages for each task in the job array. Omit for no email notification.

- `#SBATCH --mail-user=<user>`

User's email-address to receive email notification of state changes as defined by `--mail-type`. The default value is the submitting user. In contrast to the depiction in the man-page the value for `--mail-user` must be set if email notifcation is wanted for a submitting user (AKA Slurm account[47]) that is not the login user.

- `#SBATCH --export=<environment variables | ALL | NONE>`

  Identify which environment variables are propagated to the batch job. Multiple environment variable names should be comma separated. Environment variable names may be specified to propagate the current value of those variables (e.g. "`--export=EDITOR`") or specific values for the variables may be exported (e.g.. "`--export=EDITOR=/bin/vi`") in addition to the environment variables that would otherwise be set. This option is particularly important for jobs that are submitted on one cluster and execute on a different cluster (e.g. with different paths). By default all environment variables are propagated. If the argument is `NONE` or specific environment variable names, then the `--get-user-env` option will implicitly be set to load other environment variables based upon the user's configuration on the cluster which executes the job.

4.4.8 Slurm cheat sheet

Table 4.5: Slurm cheat sheet.

Action	Slurm
Job information	`squeue <job_id>` `scontrol show job <job_id>`
Job information (all)	`squeue -al` `scontrol show job`
Job information (user)	`squeue -u $USER` `showq -u $USER`
Queue information	`squeue`
Delete a job	`scancel <job_id>`

Submit a job	`srun <jobfile>`
	`sbatch <jobfile>`
	`salloc <jobfile>`

Interactive job	`salloc -N <minnodes[-maxnodes]> \` `-p <partition> sh`
Free processors	`srun -test-only -p <partition> \` `-n 1 -t <time limit> sh`
Expected start time [48]	`squeue --start -j <job_id>`
Queues/partitions	`scontrol show partition`
Node list	`sinfo -N` `scontrol show nodes`

continued on next page . . .

[47] Usually the login user has the same name as the Slurm account. Some factulties use a different slum account to submit jobs so that they can share the job management and the results.

[48] See also section *`scontrol, squeue, showq` - Query Job status* on page 61 for background informations.

. . . continued from previous page

Action	Slurm
Node details	`scontrol show node <nodename>`
Queue [49]	`sinfo` `sinfo -o "%P %l %c %D "`
Start job	`scontrol update JobId=<job_id> \` `StartTime=now`
Hold job	`scontrol update JobId=<job_id> \` `StartTime=now+30days`
Release hold job	`scontrol update JobId=<job_id> \` `StartTime=now`
Pending job	`scontrol requeue <job_id>`
Graphical Frontend	`sview`
set priority	`scontrol update JobId=<job_id> \` `-nice=-10000`
preempt job	`scontrol requeue <job_id>`
suspend job	`scontrol suspend <job_id>`
resume job	`scontrol resume <job_id>`
QoS details	`sacctmgr show QOS`
Performance metrics	`seff <job_id>`

[49] See also section *Format options for slurm commands* on page 83.

4.4.9 List of job states

Table 4.6: Job state.

Short	Long	Explanation
CA	CANCELLED	Job was explicitly cancelled by the user or system administrator. The job may or may not have been initiated.
CD	COMPLETED	Job has terminated all processes on all nodes.
CF	CONFIGURING	Job has been allocated resources, but are waiting for them to become ready for use (e.g. booting).
CG	COMPLETING	Job is in the process of completing. Some processes on some nodes may still be active.
F	FAILED	Job terminated with non-zero exit code or other failure condition.
NF	NODE_-FAIL	Job terminated due to failure of one or more allocated nodes.
PD	PENDING	Job is awaiting resource allocation.
PR	PREEMPTED	Job terminated due to preemption.
R	RUNNING	Job currently has an allocation.
S	SUSPENDED	Job has an allocation, but execution has been suspended.
TO	TIMEOUT	Job terminated upon reaching its time limit.

4.4.10 Format options for slurm commands

The available field specifications include:

Table 4.7: Field specifications.

Field	Explanation
%a	State/availability of a partition
%A	Number of nodes by state in the format "allocated/idle". Do not use this with a node state option ("%t" or "%T") or the different node states will be placed on separate lines.
%c	Number of CPUs per node
%d	Size of temporary disk space per node in megabytes
%D	Number of nodes
%f	Features associated with the nodes
%F	Number of nodes by state in the format "allocated/idle/other/total". Do not use this with a node state option ("%t" or "%T") or the different node states will be placed on separate lines.
%g	Groups which may use the nodes
%h	Jobs may share nodes, "yes", "no", or "force"'
%l	Maximum time for any job in the format "days-hours:minutes:seconds"
%m	Size of memory per node in megabytes
%N	List of node names

continued on next page ...

...continued from previous page

Field	Explanation
%P	Partition name
%r	Only user root may initiate jobs, "yes" or "no"
%R	The reason a node is unavailable (down, drained, or draining states)
%s	Maximum job size in nodes
%t	State of nodes, compact form
%T	State of nodes, extended form
%w	Scheduling weight of the nodes
%.<*>	right justification of the field
%<*>	size of field

4.4.11 Job variables

The available field specifications include:

Table 4.8: Job variables.

Environment	Slurm
Job ID	SLURM_JOB_ID / SLURM_JOBID
Job name	SLURM_JOB_NAME
Node list	SLURM_JOB_NODELIST / SLURM_NODELIST
Submit directory	SLURM_SUBMIT_DIR
Submit host	SLURM_SUBMIT_HOST
Job array index	SLURM_ARRAY_TASK_ID
User	SLURM_JOB_USER

4.5 Examples

4.5.1 Basic slurm script example

The following script asks for usage of 1 compute node with 20 cores for 10 minutes. See 'man sbatch' for details.

```
#!/bin/bash -l
#SBATCH --time=00:10:00
#SBATCH --nodes=1 --cpus-per-task=20 --constraint=cstd01
#SBATCH --partition=short
# Maximum 'mem' values depending on constraint (values in MB):
# cstd01/xeon_e52640v4/ib_1to3/cgpu01 AND
# cstd02/xeon_e52640v4/ib_1to1/nonblocking_comm: 62264
# cquad01: 255800
# cquad02: 1029944
#SBATCH --mem=60000

#SBATCH --mail-user=test.user@tu-dortmund.de
# Possible 'mail-type' values: NONE, BEGIN, END, FAIL, ALL
    ↪ (=BEGIN,END,FAIL)
#SBATCH --mail-type=ALL

cd /work/user/workdir
module purge
module load pgi/17.5
export OMP_NUM_THREADS=20
echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID
    ↪ $SLURM_TASK_PID) on $SLURMD_NODENAME"
echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"
echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"
srun ./myapp
```

4.5.2 Example using multiple GPU nodes

The following script asks for usage of 2 compute node with 20 cores each and 2 GPUs per node for 10 minutes. See 'man sbatch' for details.

```
#!/bin/bash -l
# for details.
#SBATCH --time=00:10:00
#SBATCH --nodes=2 --cpus-per-task=20 --constraint=cgpu01
    ↪ --gres=gpu:2
#SBATCH --partition=short
# Maximum 'mem' values depending on constraint (values in MB):
```

```
# cstd01/xeon_e52640v4/ib_1to3/cgpu01 AND
# cstd02/xeon_e52640v4/ib_1to1/nonblocking_comm: 62264
# cquad01: 255800
# cquad02: 1029944
#SBATCH --mem=60000

#SBATCH --mail-user=test.user@tu-dortmund.de
# Possible 'mail-type' values: NONE, BEGIN, END, FAIL, ALL
    ↪ (=BEGIN,END,FAIL)
#SBATCH --mail-type=ALL

cd /work/user/workdir
module purge
module load cuda
echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID
    ↪ $SLURM_TASK_PID) on $SLURMD_NODENAME"
echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"
echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"
nvidia-smi -a
```

4.5.3 Common software example: ANSYS CFX

To run ANSYS CFX on a **single compute node**, invoke the following script via

```
sbatch run_cfx_single_node_through_slurm.sh
```

It asks for 1 compute node with 20 cores for 90 minutes:

```
#!/bin/bash -l

#SBATCH --job-name phi1
#SBATCH --partition=short
#SBATCH --time 01:30:00
#SBATCH --exclusive
#SBATCH --nodes=1-1          # min 1 node, max 1 node
#SBATCH --ntasks-per-node=20
#SBATCH --cpus-per-task=1    # one cpu per job (hence, 20 cpus)
#SBATCH -o %N-%j.out         # STDOUT
#SBATCH -e %N-%j.err         # STDERR
# send mail when jobs starts, end,
# fails, gets requeued etc.
#SBATCH --mail-type=ALL
#SBATCH --mail-user=my.name@tu-dortmund.de
```

```
## change to directory where job file got submitted
cd $SLURM_SUBMIT_DIR

## show a number of interesting environment variables
echo "sbatch: START SLURM_JOB_ID ${SLURM_JOB_ID}"
echo "            (SLURM_TASK_PID ${SLURM_TASK_PID})"
echo "            on ${SLURMD_NODENAME}"
echo "sbatch: SLURM_JOB_NODELIST ${SLURM_JOB_NODELIST}"
echo "sbatch: SLURM_JOB_ACCOUNT ${SLURM_JOB_ACCOUNT}"
echo "sbatch: SLURM_NTASKS ${SLURM_NTASKS}"
echo "sbatch: SLURM_CPUS_ON_NODE ${SLURM_CPUS_ON_NODE}"
echo "sbatch: SLURM_JOB_NAME ${SLURM_JOB_NAME}"

## locate CFX
module load cfx/19.1

## run CFX
cfx5solve \
      -batch \
      -def Fluid_Flow_CFX.def  \
      -initial Start_Values.res \
      -start-method "Intel MPI Local Parallel" \
      -partition ${SLURM_NTASKS} \
      -double

### With newer CFX versions, you might want to try as well
#module load openmpi/mpi_thread_multiple/no_cuda/4.0.3 cfx/2019R3
#cfx5solve \
#      -batch \
#      -def Fluid_Flow_CFX.def  \
#      -initial Start_Values.res \
#      -start-method "Open MPI Local Parallel" \
#      -partition ${SLURM_NTASKS} \
#      -double
```

Listing 4.1: Contents of file 'run_cfx_single_node_through_slurm.sh'

The files

```
Fluid_Flow_CFX.def    Start_Values.res
```

should obviously be replaced with your own ANSYS CFX Solver Input File and ANSYS CFX Results File, respectively.

To use **multiple compute nodes** at once, one firstly has to pass a list of hosts to CFX. This is done by first assembling this list via

```
# Generate a comma-separated list of hostnames of compute nodes
    ↪ (plus multiplicity)
MYHOSTLIST=$( srun hostname | sort | uniq -c | awk '{print $2 "*"
    ↪ $1}' | paste -sd, )
echo $MYHOSTLIST
```

Later on, this list is passed to CFX with the additional parameter

```
cfx5solve -par-dist "$MYHOSTLIST"
```

Secondly, the Slurm job script needs to be slightly tweaked. The following listing shows a setup that uses 60 cores on 3 compute nodes:

```
#!/bin/bash -l

#SBATCH --job-name phi1
#SBATCH --partition=short
#SBATCH --time 01:30:00
#SBATCH --exclusive
#SBATCH --nodes=3-3          # min 3 nodes, max 3 nodes
#SBATCH --ntasks-per-node=20
#SBATCH --cpus-per-task=1    # one cpu per job (hence, 20 cpus)
#SBATCH -o %N-%j.out         # STDOUT
#SBATCH -e %N-%j.err         # STDERR
# send mail when jobs starts, end,
# fails, gets requeued etc.
#SBATCH --mail-type=ALL
#SBATCH --mail-user=my.name@tu-dortmund.de


## change to directory where job file got submitted
cd $SLURM_SUBMIT_DIR

## show a number of interesting environment variables
echo "sbatch: START SLURM_JOB_ID ${SLURM_JOB_ID}"
echo "            (SLURM_TASK_PID ${SLURM_TASK_PID})"
echo "            on ${SLURMD_NODENAME}"
echo "sbatch: SLURM_JOB_NODELIST ${SLURM_JOB_NODELIST}"
echo "sbatch: SLURM_JOB_ACCOUNT ${SLURM_JOB_ACCOUNT}"
echo "sbatch: SLURM_NTASKS ${SLURM_NTASKS}"
echo "sbatch: SLURM_CPUS_ON_NODE ${SLURM_CPUS_ON_NODE}"
echo "sbatch: SLURM_JOB_NAME ${SLURM_JOB_NAME}"

# Generate a comma-separated list of hostnames of compute nodes
```

```
# (plus multiplicity)
MYHOSTLIST=$( srun hostname | sort | uniq -c | \
                 awk '{print $2 "*" $1}' | paste -sd, )

## locate CFX
module load cfx/19.1

## run CFX
cfx5solve \
      -batch \
      -def Fluid_Flow_CFX.def  \
      -initial Start_Values.res \
      -parallel \
      -start-method "Intel MPI Distributed Parallel" \
      -par-dist "${MYHOSTLIST}" \
      -partition ${SLURM_NTASKS} \
      -double
```

Listing 4.2: Contents of file 'run_cfx_multiple_nodes_through_slurm.sh'

Thirdly, CFX uses SSH for the communication between nodes. Thus you need to setup inter-node SSH access (see section 4.2.3 on page 31), if you are using multiple nodes at once.

4.5.4 Common software example: ANSYS Fluent

Preliminary note: Fluent uses SSH for the communication between ranks. Thus you need to setup inter-node SSH access (see section 4.2.3 on page 31), even if you are only using multiple ranks on one single node.

The following script, when invoked via

```
sbatch run_fluent_trichter2D.sh
```

asks for 1 compute node with 20 cores for 60 minutes:

```
#!/bin/bash -l

#SBATCH --job-name trichter2D
#SBATCH --partition=short
#SBATCH --time 00:60:00
#SBATCH --exclusive
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
```

```
#SBATCH -e error_file.e
#SBATCH -o output_file.o
# send mail when jobs starts, end, fails, gets requeued etc.
#SBATCH --mail-type=ALL
#SBATCH --mail-user=my.name@tu-dortmund.de


## Gather the number of nodes and tasks
numnodes=$SLURM_JOB_NUM_NODES
numtasks=$SLURM_NTASKS
mpi_tasks_per_node=$(echo "$SLURM_TASKS_PER_NODE" | sed -e
    ↪ 's/^\([0-9][0-9]*\).*$/\1/')

## store hostname in txt file
srun hostname -s > slurmhosts.$SLURM_JOB_ID.txt

## calculate slurm task count
if [ "x$SLURM_NPROCS" = "x" ]; then
  if [ "x$SLURM_NTASKS_PER_NODE" = "x" ];then
    SLURM_NTASKS_PER_NODE=1
  fi
  SLURM_NPROCS=`expr $SLURM_JOB_NUM_NODES \*
    ↪ $SLURM_NTASKS_PER_NODE`
fi

export OMP_NUM_THREADS=1
export I_MPI_PIN_DOMAIN=omp:compact # Domains are
    ↪ $OMP_NUM_THREADS cores in size
export I_MPI_PIN_ORDER=scatter # Adjacent domains have minimal
    ↪ sharing of caches/sockets

# Number of MPI tasks to be started by the application per node
    ↪ and in total (do not change):
np=$[${numnodes}*${mpi_tasks_per_node}]

# load necessary modules
module purge
module add intel/mpi/2018.3
module add fluent/2019R1

# run the fluent simulation
fluent 2ddp -ssh -t$np -mpi=intel -pib
    ↪ -cnf=slurmhosts.$SLURM_JOB_ID.txt -g -i
    ↪ mycase_trichter2D.jou

# delete temp file
```

```
rm slurmhosts.$SLURM_JOB_ID.txt
```

Listing 4.3: Contents of file 'run_fluent_trichter2D.sh'

The file

```
mycase_trichter2D.jou
```

should obviously be replaced with your own ANSYS Fluent problem description file.

4.5.5 Common software example: Matlab

The following script, when invoked via

```
sbatch StartMatlabBatchJobViaSLURM.sh
```

asks for 1 compute node with 10 cores for 90 minutes:

```
#!/bin/bash -l
#SBATCH --job-name=MatlabSimulation
# run at most for 0 days, 1 hour, 30 minutes and 15 seconds
#SBATCH --time=0-01:30:15
#SBATCH --partition=short
# ask for ten compute cores on one compute node
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=10
# memory requirement per core in megabytes
#SBATCH --mem-per-cpu=1536
#SBATCH --output=/work/myusername/tmp/slurm_job
#SBATCH --error=/work/myusername/tmp/slurm_job
# send mail when jobs starts, end, fails, gets requeued etc.
#SBATCH --mail-type=ALL
#SBATCH --mail-user=my.name@tu-dortmund.de

cd /work/myusername/tmp
module purge
module load matlab/r2019b

# Run the Matlab simulation, stored in
    ↪ /work/myusername/tmp/matlab_main.m
srun matlab -nodisplay -nosplash -r 'matlab_main; quit;'
```

Listing 4.4: Contents of file 'StartMatlabBatchJobViaSLURM.sh'

Once Slurm grants these resources, Matlab's command line interface get invoked on the assigned compute node, spawns as many Matlab worker processes as cores requested in the Slurm job script in order to calculate in parallel an estimate for the value of π:

```
pc = parcluster('local')

% use a fixed number of Matlab worker processes, e.g. 5
%% parpool(pc, 5)

% automatically choose as many Matlab worker processes as cores
   ↪ requested in
% Slurm job file
parpool(pc, str2num(getenv('SLURM_CPUS_ON_NODE')))

% run Matlab simulation that uses commands like 'parfor',
   ↪ 'parfeval',
% 'parfevalOnAll', 'spmd' or 'distributed' to have Matlab
   ↪ automatically
% distribute the workload on multiple worker processes
EstimatePi
```

Listing 4.5: Contents of file 'matlab_main.m'

relying on the helper script

```
% Calculate the value of Pi using a Monte Carlo simulation
itmax=1e9;
n=0;

tic;
parfor i = 1:itmax
  x=rand;
  y=rand;
  if (x^2 + y^2 < 1.0)
    n=n+1;
  end
end
elapsedTime = toc;

pi = 4.0 * n / itmax;
fprintf("Calculating pi = %.10f took %s seconds\n", pi,
   ↪ elapsedTime);
```

Listing 4.6: Contents of file 'EstimatePi.m'

4.5.6 Common software example: R

A user can build additional R modules and store them in his home directory. That is the preferred way over having to install them systemwide to avoid conflict situations where user A needs a set of R modules in a certain version and user B needing them in older or newer versions. Installing them to a user's home directory allows the user to quickly check whether up- or downgrading R modules resolves issues he is having with them.

Given that the home directory is writable only on both gateway servers, a user should not try to build R modules on any of the compute nodes (unless he redefines `HOME` to point to some writable location inside a Slurm job). The following convenience script facilitates downloading optional R packages:

```
#!/bin/bash -l

extra=""
if test -z "$@"; then
    extra="-O index.html"
fi
file=$( grep ">"$1_ index.html | awk -F'>' '{ print $7 }' | sed
    ↪ 's|</a||' | tail -n 1)
wget http://cran.r-project.org/src/contrib/$file $extra
```

Copy this content to a new file named `load_r_module`, set the executable bit for the script

```
chmod 755 load_r_module
```

When invoked without arguments, the script downloads the index of the directory `https://cran.r-project.org/src/contrib/` and stores it as `index.html` in the local directory:

```
rm -f index.html*
./load_r_module
```

When invoked with an argument, the script queries the cache file `index.html` in the local directory for that given argument string and tries to download the tarball if a R module is found that matches this string. Example:

```
./load_r_module digest
```

will download the most recent version of the R module `digest`, at the time of writing `digest_0.6.23.tar.gz`.

Subsequently, the user can compile and install this R module as follows:

```
module purge
module load R/<version of your liking>

for package in <list of desired R modules>; do ./load_r_module
    ↪ ${package} && R CMD INSTALL
    ↪ --configure-args=--with-mpi=${OMPI_HOME}
    ↪ ${package}_*.tar.gz || ( echo "ERROR"; read junk ); done
```

Make sure to replace the strings `<version of your liking>` and `<list of desired modules>` in the instructions above appropriately. In case the R module(s) you want to install have unfulfilled dependencies, the R install command will fail, reporting the name of the missing dependency:

```
for package in spdep ; do ./load_r_module ${package} && R CMD
    ↪ INSTALL --configure-args=--with-mpi=${OMPI_HOME}
    ↪ ${package}_*.tar.gz || ( echo "ERROR. Press any key to
    ↪ continue"; read junk ); done
--2020-04-11 15:21:32--
    ↪ http://cran.r-project.org/src/contrib/sp_1.4-1.tar.gz
Resolving cran.r-project.org (cran.r-project.org)... 137.208.57.37
Connecting to cran.r-project.org
    ↪ (cran.r-project.org)|137.208.57.37|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1698902 (1.6M) [application/x-gzip]
Saving to: 'sp_1.4-1.tar.gz'

100%[=============================>] 1,698,902   --.-K/s   in
    ↪ 0.1s

2020-04-11 15:21:32 (12.7 MB/s) - 'sp_1.4-1.tar.gz' saved
    ↪ [1132945/1132945]

ERROR: dependencies 'sp', 'spData', 'sf', 'deldir', 'LearnBayes',
    ↪ 'coda', 'expm', 'gmodels' are not available for package
    ↪ 'spdep'
* removing
    ↪ '/home/msvebuij/R/x86_64-pc-linux-gnu-library/3.6.1/rgdal'
```

```
ERROR. Press any key to continue
```

In this particular example, where one wanted to install the R module `spdep`, we needed to first compile and install `sp` and then a lot of others which had dependencies of their own. Prepend iteratively all the missing R modules to `<list of desired modules>` in ascending order of dependency and start over until the compilations succeeds. This approach easily requires a dozen or so iterations, depending on the particular R module a user wants to use. For this example, the complete instruction would look like:

```
for package in sp raster spData e1071 classInt DBI units sf
    ↪ deldir LearnBayes coda expm gmodels spdep ; do ./load_r_mod
    ↪ ${package} && R CMD INSTALL
    ↪ --configure-args=--with-mpi=${OMPI_HOME}
    ↪ ${package}_*.tar.gz || ( echo "ERROR"; read junk ); done
```

It may turn out, however, that some system software needs to installed as well upon which an R module relies. In that case, please inform the LiDO team what system software is required and, better yet, additionally what R module you are trying to compile.

The following script, when invoked via

```
 sbatch StartRBatchJobViaSLURM.sh
```

asks for 1 compute node with 1 cores for 90 minutes and 15 seconds:

```
#!/bin/bash -l
#SBATCH --job-name=Ranalysis
# run at most for 0 days, 1 hour, 30 minutes and 15 seconds
#SBATCH --time=0-01:30:15
#SBATCH --partition=short
# ask for a single compute core on one compute node
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=1
# memory requirement per CPU in megabytes
#SBATCH --mem-per-cpu=1536
#SBATCH --output=/work/myusername/tmp/slurm_job
#SBATCH --error=/work/myusername/tmp/slurm_job
# send mail when jobs starts, end, fails, gets requeued etc.
#SBATCH --mail-type=ALL
#SBATCH --mail-user=my.name@tu-dortmund.de
```

```
cd /work/myusername/tmp
module purge
module load R/3.6.1-gcc73-base
    ↪ openmpi/mpi_thread_multiple/no_cuda/4.0.1

# Run the R analysis, use an external script for the R
    ↪ instructions
Rscript my_script.R
```

4.5.6.1 Using multiple versions of R along with additional R modules

When building additional R modules yourself, please be aware that R requires that all R modules are built by the very same R version and that this R version is the one you invoke. Given that on LiDO3 multiple R versions are available, it might happen that you compiled an additional R module with `R/3.6.3-gcc93-base`, but tried to invoke `R` after loading the modulefile `R/4.0.0-gcc93-base` some time later. In these cases of conflicting R versions involved, `R` will bail out.

To avoid this, use a slightly more complicated `~/.Rprofile` than the default one. Instead of

```
.libPaths("/work/<user>/R")
```

Listing 4.7: Default contents of file '.Rprofile', problematic when using multiple R versions

use

```
# Source: https://stackoverflow.com/a/54555489

# Set version specific local libraries
## get current R version (in semantic format)
version <- paste0(R.Version()$major,".",R.Version()$minor)
## get username on Unix
## (note: use USERNAME under Microsoft Windows)
uname <- Sys.getenv("USER")
## generate R library path for parent directory
libPath <- paste0("/work/", uname, "/R/")

setLibs <- function(libPath, ver) {
        # combine parent and version for full path
        libfull <- paste0(libPath, ver)
        # create a new directory for this R version
        # if it does not exist
```

```
        if(!dir.exists(libPath)) {
                dir.create(libPath)
        }
        if(!dir.exists(libfull)) {
                # Warn user (the necessity of creating
                # a new library may indicate an inadvertant
                # choice of the wrong R version)
                warning(paste0("Library for R version '", ver, "'
    ↪ does not exist; it will be created at: ", libfull ))
                dir.create(libfull)
        }
        .libPaths(c(libfull, .libPaths()))
}

setLibs(libPath, version)
```

Listing 4.8: Contents of file '.Rprofile', compatible with using multiple R versions

Note that you need to compile additional R modules for every R version you intend to use. The compiled R libraries will end up in subdirectories of `/work/<user>/R/<R version>/`.

4.5.7 Third-party node usage example

In this case, the partition is related to the nodes itself and no additional constraint is needed to identify the nodes to be used.

```
#!/bin/bash -l
#SBATCH --time=00:10:00
#SBATCH --nodes=1 --cpus-per-task=20
#SBATCH --partition=ext_vwl_prio
#SBATCH --mem=250000

#SBATCH --mail-user=test.user@tu-dortmund.de
# Possible 'mail-type' values: NONE, BEGIN, END, FAIL, ALL
    ↪ (=BEGIN,END,FAIL)
#SBATCH --mail-type=ALL

cd /work/user/workdir
module purge
module load pgi/17.5
export OMP_NUM_THREADS=20
echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID
    ↪ $SLURM_TASK_PID) on $SLURMD_NODENAME"
echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"
echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"
```

```
srun ./myapp
```

4.5.8 Have a job automatically clean up when risking to exceed the configured walltime

By default, *Slurm* (up to version 20.02) sends the signal `SIGQUIT` and, after waiting for the amount of time defined by the Slurm configuration parameter `KillWait` which on LiDO3 is to 30 s –, the signal `SIGTERM` to a job that exceeded its requested walltime at the same time. If one needs more time between these two signals, i.e. the first signal that indicates that action needs to be taken and the second signal that definitely pulls the plug on your simulation, for instance, to move result files from the `/scratch` file system to the parallel file system and to clean up any remaining temporary files, the user needs to set up three things in the Slurm job script:

1. A `SBATCH` instruction when to send what kind of signal. This can be done by including the following lines (only the #SBATCH instruction is the actual workhorse, the preceding lines are mere comments for the reader) in the header section of the *Slurm* job script

   ```
   # When a job is within 120 seconds of its end time,
   # send it the signal SIGQUIT.
   # Note 1: due to the resolution of event handling
   #         by Slurm, the signal may be sent up to 60
   #         seconds earlier than specified.
   # Note 2: only *one* signal can be defined. Later
   #         Slurm signal definition override earlier
   #         definitions.
   #SBATCH --signal=B:SIGQUIT@120
   ```

   which sends approximately 2 minutes before exceeding the wall time the signal `SIGQUIT`.

2. A shell `trap` trying to catch the signal and defining an action to undertake upon receiving it. Example:

   ```
   trap -- 'echo \"Got SIGQUIT at $(date). Starting cleanup\";
       ↪ test -d /scratch/${USER}/${SLURM_JOB_ID} && rm -rf
       ↪ /scratch/${USER}/${SLURM_JOB_ID}' SIGQUIT;
   ```

   This oneliner is hard to read such that some users may prefer the alternative of a custom shell function defining the actions near job end:

```
cleanup_before_exiting() {
    echo -n 'Got SIGQUIT at $(date),';
    echo -n 'roughly 2 minutes before exceeding the';
    echo 'walltime. Starting clean up.';
    test -d /scratch/${USER}/${SLURM_JOB_ID} && \
        rm -rf /scratch/${USER}/${SLURM_JOB_ID}
    exit 0;
}
trap -- 'cleanup_before_exiting' SIGQUIT
```

3. Finally, it is absolutely mandatory to send any of the long-running processes your Slurm job will execute immediately to the background by adding a trailing ampersand to that process' command and to subsequently add a 'wait' shell command that causes the Slurm job file to wait for the completion of the long-running process before continuing. Example:

```
# Start the actual worker process (a simple 'sleep'
# in this example).
# Note: It is absolutely mandatory to immediately
#       send the job to the background with the
#       trailing ampersand and then use the 'wait'
#       shell command to wait for the completion of
#       the worker process. Otherwise the Slurm
#       signal is *not* caught by this Slurm job
#       script and the configured action to run
#       shortly before exceeding the requested
#       walltime will *not* run!
sleep 600 &
wait
```

A complete *Slurm* job file example is given below:

```
#!/bin/sh -l

#SBATCH --time=00:04:00
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=1
#SBATCH --partition=short
#SBATCH --mail-type=NONE
# When a job is within 120 seconds of its end time,
# send it the signal SIGQUIT.
# Note 1: due to the resolution of event handling
#         by Slurm, the signal may be sent up to 60
#         seconds earlier than specified.
# Note 2: only *one* signal can be defined. Later
```

```
#         Slurm signal definition override earlier
#         definitions.
#SBATCH --signal=B:SIGQUIT@120

useTrapVariant=2

if test ${useTrapVariant} = 1; then
  ##################################
  # Example 1: Simple signal handling with a
  # one-liner: print a message, then start
  # cleaning up in /scratch before job exceeds
  # requested walltime.
  trap -- 'echo \"Got SIGQUIT at $(date). Starting
   ↪ cleanup\"; \
  test -d /scratch/${USER}/${SLURM_JOB_ID} && \
  rm -rf /scratch/${USER}/${SLURM_JOB_ID}' SIGQUIT;
else
  ##################################
  # Example 2: Same prupose, but more readable
  # with a user function; print a message, then
  # start cleaning up in /scratch before job
  # exceeds requested walltime.
  cleanup_before_exiting() {
    echo -n 'Got SIGQUIT at $(date),';
    echo -n 'roughly 2 minutes before exceeding the';
    echo 'walltime. Starting clean up.';
    test -d /scratch/${USER}/${SLURM_JOB_ID} && \
      rm -rf /scratch/${USER}/${SLURM_JOB_ID}
    exit 0;
  }
  trap -- 'cleanup_before_exiting' SIGQUIT
fi


# Start the actual worker process (a simple 'sleep'
# in this example).
# Note: It is absolutely mandatory to immediately
#       send the job to the background with the
#       trailing ampersand and then use the 'wait'
#       shell command to wait for the completion of
#       the worker process. Otherwise the Slurm
#       signal is *not* caught by this Slurm job
#       script and the configured action to run
#       shortly before exceeding the requested
#       walltime will *not* run!
sleep 600 &
wait
```

4.5.9 Example for job steps

A job consists of

- one or more steps,
- each step executing one or more tasks,
- each task using one or more CPU.

Typically jobs are created with the `sbatch` command, containing steps that are created with the `srun` command.

Tasks are requested (at the job level or the step level) with `--ntasks` and CPUs[50] are requested per task with `--cpus-per-task`.

Note that jobs submitted with `sbatch` have one implicit step — the Bash script itself.

```
#!/bin/bash -l
#SBATCH --nodes 7
#SBATCH --tasks-per-node 6
#SBATCH --cpus-per-task 1
# The job requests 42 CPUs, on 7 nodes, every task will use 1 cpu.

# STEP 01:
# request 7 nodes,
# sub-allocate 7 tasks (one per node) to create a directory in
    ↪ /scratch.
# Must run on every node but only only one task per node needed.
srun --nodes 7 --tasks 7 mkdir -p /scratch/${USER}_${SLURM_JOBID}

# STEP 02:
# No explicit allocation, hence use all 64 tasks to run an MPI
    ↪ program
# on some data to produce some output.
srun mpi_process.mpi <input.dat > output.txt &

# STEP 03:
# sub-allocate of 24 tasks for a not well scaling program.
srun --ntasks 24 --nodes 4 --exclusive reduce_mpi_data <
    ↪ output.txt > result.txt &

# STEP 04:
# sub-allocate a single node.
# The gzip cannot run on separate nodes to compress output.txt.
# Thanks to the ampersand `&` this step runs at the same time as
    ↪ the
```

[50] CPU cores to be more precise.

```
# previous step
OMP_NUM_THREAD=10; srun --ntasks 10 --nodes 1 --exclusive gzip
    ↪ output.txt &

# wait for the steps to finish
wait
```

4.5.10 Example for parallel debugging with TotalView

TotalView [51] is a HPC debugging software for parallel debugging of C/C++, Fortran and mixed-language python applications. It is a available as a module [53].

The TotalView remote debugging setup consists of three elements:

- The GUI visualisation on the user's computer, received from the gateway
- The TotalView master running on the gateway/frontend
- The tvconnect debugger client, running on the compute nodes

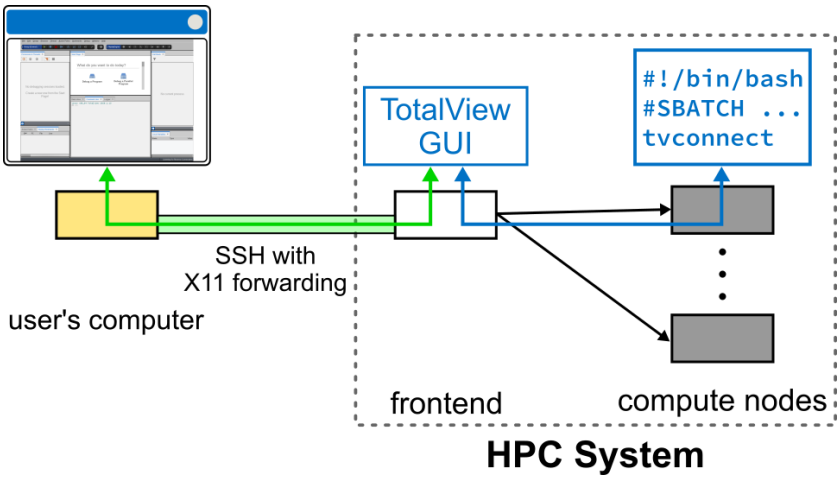


Figure 4.22: TotalView debugging overview

To start the debugging process, you need to make shure that you are able to start a GUI application on LiDO3. If you happen to have an X server on your side of the connection, you may simply use ssh with the `-X` parameter to tunnel the applications

[51] TotalView[52] website
[53] see section *Modules in job scripts* on page 46

rendering to your workstation or you may use a ThinLinc connection to use a Desktop on a LiDO3 gateway server.

TotalView organises the communication between the debugger on the compute node (`tvconnect`) and the GUI on the gateway server (`totalview`) via a shared directory, that needs to adhere to special file permissions [54]. The easiest way to ensure these constrains is to let TotalView itself create the directory on the `/work/$USER` directory. Thus one needs to define the shell variable `TV_REVERSE_CONNECT_DIR` whenever calling any TotalView binary. We will use `/work/$USER/.totalview` in the reminder of this section. This can be easily achieved by adding

```
export TV_REVERSE_CONNECT_DIR=/work/$USER/.totalview
```

to your shell rc file, e.g. `.bashrc` or `.cshrc`.

The next step is to preprend the usual `mpirun` or `srun` call in your *Slurm* job script with `tvconnect`.

```
tvconnect mpirun -n 80 ./helloworld
```

Obviously, your program should be compiled with debugging symbols and maybe without any optimisations. For example with GCC that would mean using the flags `-O0 -g`.

To start the actual debugging, you must make sure, that the `totalview` GUI application is running on the gateway server (i.e. shows a windows on your screen) and that your job to be debugged is executing with the aforementioned changes. In this case, a dialog will be presented in the GUI whether you want to start debugging your application.

[54] see the documentation[55] for a detailed description

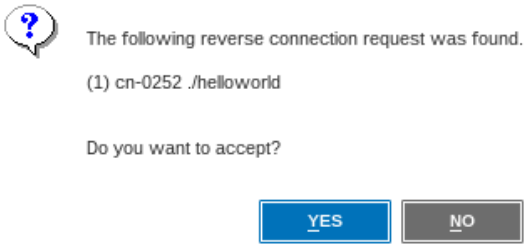


Figure 4.23: TotalView dialog for incoming debugging process

Note that your *slurm* job will be on hold until you start debugging in the GUI or the maximum walltime is reached. That is it, now the TotalView debugger is hooked to your program and you can begin the actual debugging process.

4.6 System overview

- name: LiDO3
- architecture: Distributed Memory
- vendor: Megware
- installation: 2017

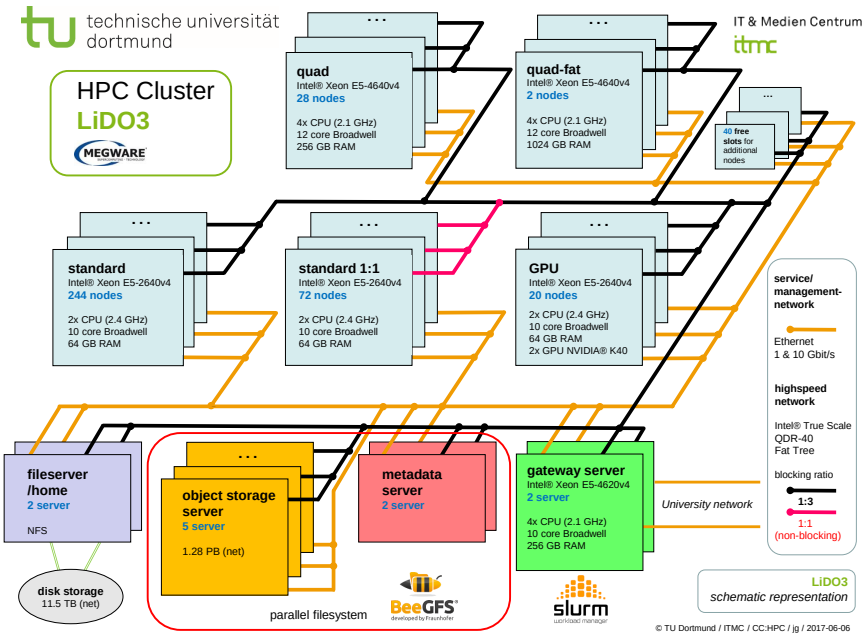


Figure 4.24: Schematic representation of the LiDO3 architecture.

4.7 Dictionary

4.7.1 Walltime

Walltime, or *Wall-Clock Time* is the passage of time from the moment a job is assigned one or multiple compute nodes and started until it ends, seen from the human perspective. In other words, if the job is started but some necessary resource is missing or becomes unavailable while the job is still running (e.g., filesystem, network, results from a previous computation as input data), walltime increases. In this case, whether or not CPU time increases depends on whether the processes started by the job perform a busy-wait or put the CPU to sleep while waiting for the necessary resource to become available again. So, if a requested CPU waits for seven hours for resources and intermittendly uses the CPU for one hour, walltime is 8 hours, CPU time is 1 hour. When using multiple cores, the CPU time is accordingly scaled - walltime is not, obviously.

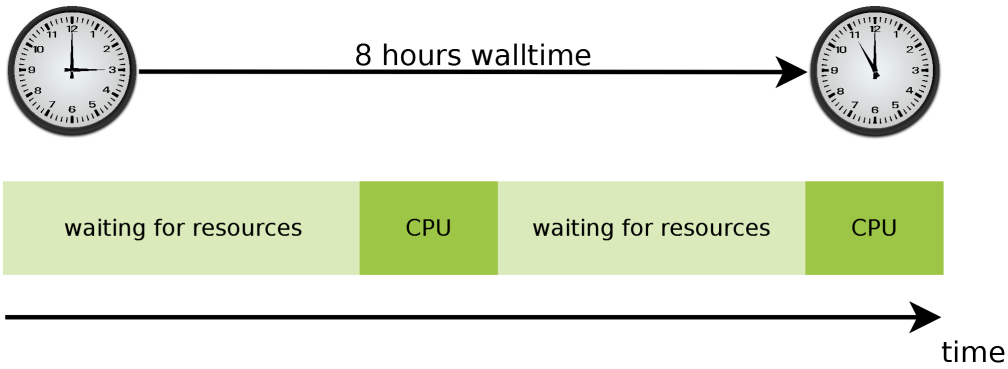


Figure 4.25: A job waiting more than utilizing the CPU uses eight hours walltime total.

4.7.2 Backfilling

Backfilling is a mechanism that allows starting a job with lower priority before a job with higher priority in the queue without delaying the job with the higher priority. By doing this *Backfilling* helps to maximize cluster utilization and throughput.

Let *Job A* be a job that just has started. *Job B* needs the nodes that are currently used by *Job A* and some extra nodes. Thus it can only start after *Job A* has been finished.

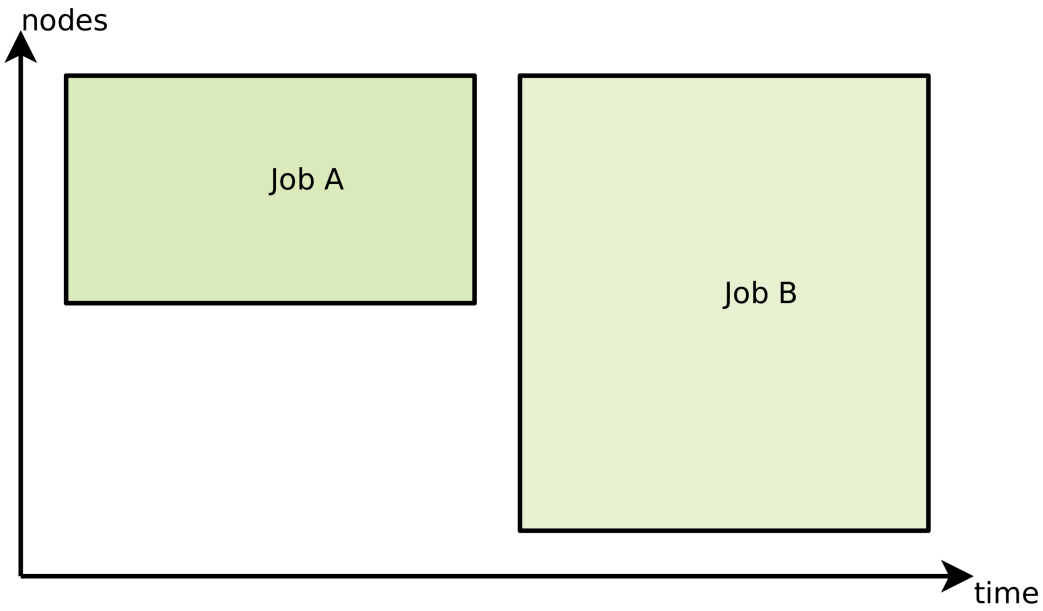


Figure 4.26: *Job B* is waiting for nodes used by *Job A*.

Job C is smaller than *Job A* - it will use less *Walltime*. And it does not depend on nodes that are used by *Job A*. This means that *Job C* can be started before *Job B* without delaying *Job B*.

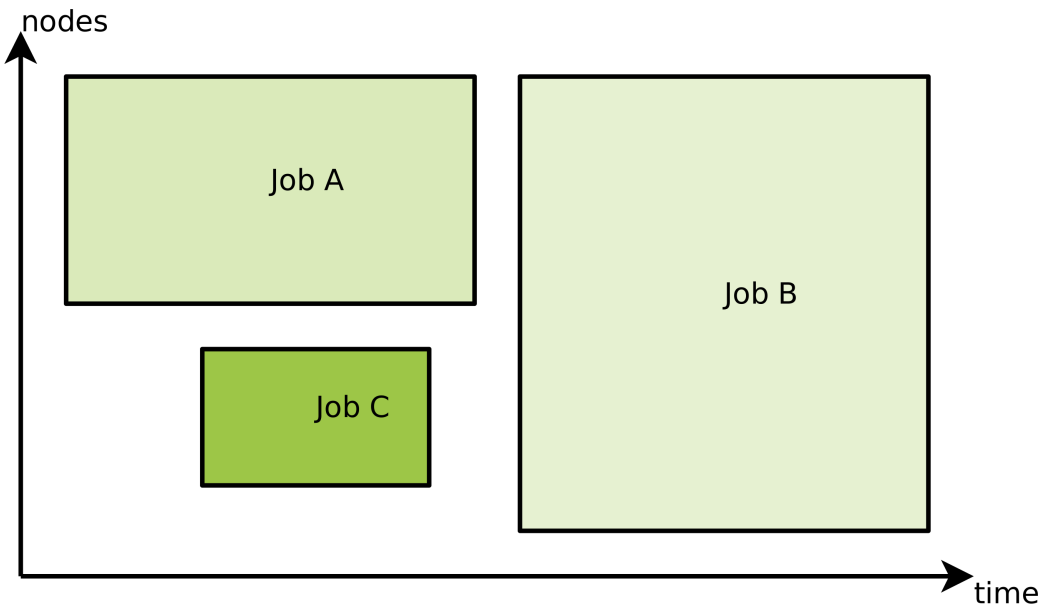


Figure 4.27: *Job C* is started before *Job B* because it will be finished before *Job B* can start.

Filling those gaps in the execution plan is called *Backfilling*.

4.8 Get support

For support and further assistance, please write an email to
the LiDO team mailing list[56]
(lido-team.itmc@lists.tu-dortmund.de).

4.9 Frequently asked questions

4.9.1 My Slurm job exits with `can't open /dev/ipath, ↪ network down (err=26)`

Your job encountered a race conditions, described in detail at bugs.schedmd.com[57].
This sometimes happens if multiple users try to use MPI on the same node independently.
Until this is fixed by the MPI vendors, a common work-around is to use the nodes all alone by adding

```
##SBATCH --exclusive
```

to your Slurm job script.
If you happen to use Intel MPI, another solution may be to define a certain environment variable by adding

```
export I_MPI_HYDRA_UUID=`uuidgen`
```

to your job script and thus inhibit the race condition.

4.9.2 No GPU is visible on a GPU node

In order to actively use a GPU, you need to add

```
#SBATCH --gres=gpu:n
```

[56] mailto:lido-team.itmc@lists.tu-dortmund.de?subject=LiDO3:%20support%20needed
[57] https://bugs.schedmd.com/show_bug.cgi?id=5956

to your Slurm job script, where `n` denotes the number of GPUs you want to use.

4.9.3 How can i use more than one CPU socket on a GPU node?

Every CPU socket is bound to one GPU. Thus if you want to use more than one CPU socket (i.e. more than 10 cores), you need to allocate both GPUs with

```
#SBATCH --gres=gpu:2
```

4.10 Appendix

4.10.1 Symbolic links for non-writable home directory

Here is an example of some software that needs to write in the home directory during runtime. `${NEWUSER}` contains the name of the user that is affected.

```
# Software like 'matplotlib' (standalone or inside ParaView)
   ↪ tries to write a
# lock file to
   ↪ $HOME/.cache/matplotlib/tex.cache/.matplotlib_lock-*.
   ↪ Without
# this symbolic link, matplotlib would fail when run on compute
   ↪ nodes.
$ ssh gw01
$ mkdir /work/${NEWUSER}/.allinea
$ ln -s /work/${NEWUSER}/.allinea /home/${NEWUSER}/.allinea
$ mkdir /work/${NEWUSER}/.ansys
$ ln -s /work/${NEWUSER}/.ansys /home/${NEWUSER}/.ansys
$ mkdir /work/${NEWUSER}/.cache
$ ln -s /work/${NEWUSER}/.cache /home/${NEWUSER}/.cache
$ mkdir -p /work/${NEWUSER}/.ccache
$ ln -s /work/${NEWUSER}/.ccache /home/${NEWUSER}/.ccache
$ mkdir -p /work/${NEWUSER}/.cmake/packages
   ↪ /home/${NEWUSER}/.cmake
$ mkdir /work/${NEWUSER}/.cfx
$ ln -s /work/${NEWUSER}/.cfx /home/${NEWUSER}/.cfx
$ ln -s /work/${NEWUSER}/.cmake/packages
   ↪ /home/${NEWUSER}/.cmake/packages
$ mkdir /work/${NEWUSER}/.config
$ ln -s /work/${NEWUSER}/.config /home/${NEWUSER}/.config
$ mkdir /work/${NEWUSER}/.felix
$ ln -s /work/${NEWUSER}/.felix /home/${NEWUSER}/.felix
$ mkdir /work/${NEWUSER}/felix-cache
$ ln -s /work/${NEWUSER}/felix-cache /home/${NEWUSER}/felix-cache
$ mkdir /work/${NEWUSER}/.java
$ ln -s /work/${NEWUSER}/.java /home/${NEWUSER}/.java
$ touch /work/${NEWUSER}/.lesshst
$ ln -s /work/${NEWUSER}/.lesshst /home/${NEWUSER}/.lesshst
$ mkdir /work/${NEWUSER}/.matlab
$ ln -s /work/${NEWUSER}/.matlab /home/${NEWUSER}/.matlab
$ mkdir /work/${NEWUSER}/.oracle_jre_usage
$ ln -s /work/${NEWUSER}/.oracle_jre_usage
   ↪ /home/${NEWUSER}/.oracle_jre_usage
$ mkdir -p /work/${NEWUSER}/.ssh
$ ln -s /work/${NEWUSER}/.ssh /home/${NEWUSER}/.ssh
$ mkdir /work/${NEWUSER}/.subversion
```

```
$ ln -s /work/${NEWUSER}/.subversion /home/${NEWUSER}/.subversion
#does not work   $ touch /work/${NEWUSER}/.viminfo
#does not work   $ ln -s /work/${NEWUSER}/.viminfo
    ↪ /home/${NEWUSER}/.viminfo
$ touch /work/${NEWUSER}/.Xauthority
$ ln -s /work/${NEWUSER}/.Xauthority /home/${NEWUSER}/.Xauthority
```

4.10.2 Migrating your Slurm scripts to full node usage

The following approaches have proven to work for a wide variety of use cases. Each of them assumes that your current calculation executed by a single program call is not able to utilize a complete LiDO3 node. It further assumes that you want to execute this program multiple times, possibly for differing input data. It is up to you (for example inside a short benchmarking session) to know or figure out how many program calls can be done in parallel on a single node to utilize – but not overutilize – the available resources (e.g. CPU cores or amount of memory or memory bandwidth).

Obviously, this guide cannot provide any solution for cases where you only want to execute one serial, single-threaded program call at a time – this usage model is not suited for a compute cluster at all.

4.10.2.1 Executing several processes concurrently in the background

If your programs are not compiled with MPI support at all, you can exploit a common shell feature: every command is executed in the background if followed by the ampersand character, `&`. In other words, the command is run, but – unlike when run in foreground mode – control is immediately passed back to the shell such that one can interactively enter and invoke other commands. Or have another program start in non-interactive, i.e. batch, mode.

To explicitly wait for all programs started by your Slurm script and that are being executed in the background to finish, before control is passed back to the shell (i.e. the shell is ready to execute a new command), issue the command `wait`.

As there is no Slurm logic involved in the program startup at all, this approach does only work on a single node. If you want to allocate multiple nodes at once, this approach won't work for you because the Slurm script is only executed on the first of those compute nodes.

As there is now only one Slurm script executing multiple programs at once, it might be a good idea to redirect `stdout` and `stderr` to disjunct files for improved clarity and a reasonable chance to debug any arising issue. The syntax `&> filename` means to catch both `stdout` and `stderr` in a single file named `filename`. The alternative syntax catches them in separate files, with `&1> outputfile` catching ordinary terminal output and `&2> errorfile` catching any error output.

Listing 4.9: List every command individually

```
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --time=02:00:00
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
#SBATCH --constraint=cstd01
#SBATCH --exclusive

cd project_folder
call-to-single-threaded-program-a parameter1_1 parameter2_1 &>
    ↪ out-and-err.1  &
call-to-single-threaded-program-a parameter1_2 parameter2_2 &>
    ↪ out-and-err.2  &
call-to-single-threaded-program-b parameter1_3 parameter2_3 1>
    ↪ out.3 2> err.3 &
wait
```

Obviously, you are not restricted to calling the *same* program over and over again.

It is, however, advised to group your program calls by similar execution time per node to avoid that a compute node is partially idle and gets underutilized once the first programs finish.

If your parameters follow some sort of scheme or logic, you might want to use a simple `for` loop to start all calculations with fewer lines of code.

Listing 4.10: Use a for loop to invoke commands

```
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
#SBATCH --time=02:00:00
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
#SBATCH --constraint=cstd01
#SBATCH --exclusive

cd project_folder
for (( i=0; i < $SLURM_NTASKS ; ++i)); do
  call-to-single-threaded-program $i &> out-and-err.$i &
done
wait
```

In this example, we start 20 programs and pass a running number between 0 and 19 to each program. Here we simply assume that the program will then decide on its own how to react: which input parameters to use based on the number passed as command line argument.

4.10.2.2 Slurm's `srun --multi-prog` option

The `--multi-prog` option of `srun` allows to start multiple programs (or the same program multiple times) with different sets of parameters as long as the additional parameters, e.g. `--cpus-per-task`, are identical.

For this purpose, `srun` parses a configuration file one needs to provide and that steers the actual program execution.

The following configuration file `srun.conf` mimicks the commands run in example 4.9:

Listing 4.11: Example for srun.conf

```
0 call-to-single-threaded-program-a parameter1_1 parameter2_1 &>
    ↪ out-and-err.1
1 call-to-single-threaded-program-b parameter1_2 parameter2_2 &>
    ↪ out-and-err.2
2 call-to-single-threaded-program-b parameter1_3 parameter2_3 1>
    ↪ out.3 2> err.3
```

It tells `srun` to invoke `call-to-single-threaded-program-a` as the first task, `call-to-single-threaded-program-b` as the second task and third task.
The executable arguments may be augmented by expression `%t` which gets replaced by the task number, and `%o` which gets replaced with task's offset within this range.
If a line should be executed more than once, you can list multiple task ranks per line. Multiple values may be comma separated. Ranges may be indicated with two numbers separated with a '-' with the smaller number first (e.g. "0-4" and not "4-0"). To indicate all tasks, specify a rank of '*' (in which case you probably should not be using this option). If an attempt is made to initiate a task for which no executable program is defined, the following error message will be produced "No executable program specified for this task".

Listing 4.12: Example for srun.conf with 6 tasks in total

```
0,5 call-to-single-threaded-program-a parameter1_1 parameter2_1
    ↪ &> out-and-err.%t
1-3 call-to-single-threaded-program-b parameter1_2 parameter2_2
    ↪ &> out-and-err.%t
4 call-to-single-threaded-program-b parameter1_3 parameter2_3 1>
    ↪ out.%t 2> err.%t
```

As is common in multiple programming environments, `0` references the first task and `$SLURM_NTASKS - 1` references the last task.
The corresponding Slurm script would look like this:

```
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=3
#SBATCH --time=02:00:00
```

```
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
#SBATCH --constraint=cstd01
#SBATCH --exclusive

cd project_folder
srun --multi-prog ./srun.conf
```

Note that with `srun` and its ability to spread jobs across multiple allocated compute nodes, we could ask for more than a single compute node for this Slurm job, i.e. increase the node count in the line `#SBATCH --nodes=x` to more than `1`. Obviously, we would then need to add many more lines to `srun.conf` to cater for a higher workload.

4.10.2.3 GNU Parallel

GNU Parallel overcomes the disadvantage of the former approaches and relieves the user from the burden of providing a matching number of program calls and matching the execution times. In the simplest use case, one provides a file with one arbitrary program execution per line. The amount of lines does not need to match the amount of cores, allocated by your Slurm job scripts. GNU Parallel will process the next open line, if any previously processed line finishes.

It is, however, advised to put those lines in front of all others that trigger a long running simulation such that such a line will not get executed as one of the last.

Let us say we have a file similar to the above `srun` example:

Listing 4.13: Example commands.txt

```
call-to-single-threaded-program-a parameter1_1 parameter2_1 &>
    ↪ out-and-err.1
call-to-single-threaded-program-b parameter1_2 parameter2_2 &>
    ↪ out-and-err.2
call-to-single-threaded-program-b parameter1_3 parameter2_3 1>
    ↪ out.3 2> err.3
```

Then this commands can be processed via

```
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=1
```

```
#SBATCH --ntasks-per-node=2
#SBATCH --time=02:00:00
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
#SBATCH --constraint=cstd01
#SBATCH --exclusive

cd project_folder
parallel < commands.txt
```

By default, GNU Parallel detects the number of cores of a node and starts one command per core. You can use the parameter `--jobs` to specify the number of concurrent commands explicitly.

```
parallel --jobs ${SLURM_NTASKS_PER_NODE} < commands.txt
```

If you want to use GNU Parallel with multiple nodes at once, you can provide a nodelist via `--sshloginfile`. Note, that `--jobs` now controls the number of concurrent programm calls **per** node.

```
scontrol show hostnames $SLURM_JOB_NODELIST > node_list
parallel --sshloginfile node_list --jobs ${SLURM_NTASKS_PER_NODE}
    ↪ < commands.txt
```

You may need to set up a proper inter-node SSH connections (see section 4.2.3 on page 31) to make this work.

Note that GNU Parallel does not load any module environment on the remote site. You might simply want to ensure this in the `commands.txt` or by using the `env_parallel` bash function.

4.10.3 Slurm for Torque/PBS users

A *Torque queue* is a *Slurm partition*.

Table 4.9: Job control.

Action	Slurm	Torque/PBS	Maui
Job information	squeue <job_id> scontrol show job <job_id>	qstat <job_id> qstat -f <job_id>	checkjob
Job information (all)	squeue -al scontrol show job	qstat -f	
Job information (user)	squeue -u $USER	qstat -u $USER	
Queue information	squeue	qstat	showq
Delete a job	scancel <job_id>	qdel	
Clean up leftover job		momctl -c <job_id>	
Submit a job	srun <jobfile> sbatch <jobfile> salloc <jobfile>	qusb <jobfile>	msub
Interactive job	salloc -N <minnodes[-maxnodes]> \ -p <partition> sh	qsub -I	
Free processors	srun -test-only -p <partition> \ -n 1 -t <time limit> sh		showbf
Expected start time	squeue --start -j <job_id>		showstart <job_id>
Blocked jobs	squeue --start		mdiag -b showq -b
Queues/partitions	scontrol show partition	qstat -Qf	mdiag -c
Node list	sinfo -N scontrol show nodes	pbsnode -l	
Node details	scontrol show node <nodename>	pbsnode <nodename>	
Queue [58]	sinfo sinfo -o "%P %l %c %D "	qstat -q	
Start job	scontrol update JobId=<job_id> \ StartTime=now	qrun	runjob
Hold job	scontrol update JobId=<job_id> \ StartTime=now+30days	qhold <job_id>	sethold
Release hold job	scontrol update JobId=<job_id> \ StartTime=now	qrls <job_id>	releasehold
Pending job	scontrol requeue <job_id>		
Graphical Frontend	sview	xpbs	
set priority	scontrol update JobId=<job_id> \ -nice=-10000		setspri 10000 \ <job_id>
preempt job	scontrol requeue <job_id>		mjobctl -R <job_id>
suspend job	scontrol suspend <job_id>		mjobctl -s <job_id>
resume job	scontrol resume <job_id>		mjobctl -r <job_id>
QoS details	sacctmgr show QOS	mdiag -q	

[58] See also section *Format options for slurm commands* on page 83.

4.10.3.1 Job variables in Slurm and Torque

The available field specifications include:

Table 4.10: Job variables.

Environment	Torque/PBS	Slurm
Job ID	`PBS_JOBID`	`SLURM_JOB_ID` / `SLURM_JOBID`
Job name	`PBS_JOBNAME`	`SLURM_JOB_NAME`
Node list	▲ **`PBS_NODELIST`** `PBS_NODEFILE`	`SLURM_JOB_NODELIST` / `SLURM_NODELIST`
Submit directory	`PBS_O_WORKDIR`	`SLURM_SUBMIT_DIR`
Submit host	`PBS_O_HOST`	`SLURM_SUBMIT_HOST`
Job array index	`PBS_PBS_ARRAY_INDEX`	`SLURM_ARRAY_TASK_ID`
User	`PBS_USER`	`SLURM_JOB_USER`

4.10.4 Picture credits

- Windows Logo - Wiki Commons[59]
- Apple Logo - Wiki Commons[60]
- Tux Logo - Wiki Commons[61]
- Computer shape - Openclipart[62]
- Server shape Openclipart[63]
- Light bulb - Openclipart[64]
- Warning triangle - Openclipart[65]
- Clock - Openclipart[66]
- TU Dortmund ITMC - itmc.tu-dortmund.de[67]
- Mordor Meme generated with imgflip[68]
- TotalView pictures - PC2 TotalView HowTo[69]
- Remaining screeshots and figures - created by the LiDO Team

[59] http://commons.wikimedia.org/wiki/Category:Microsoft_Windows_logos
[60] http://commons.wikimedia.org/wiki/File:Apple_logo_black.svg?uselang=de
[61] http://commons.wikimedia.org/wiki/Tux#/media/File:Tux.svg
[62] https://openclipart.org/detail/17391/computer
[63] https://openclipart.org/detail/171414/router
[64] https://openclipart.org/detail/211389/lightbulb
[65] https://openclipart.org/detail/14428/h0us3s-Signs-Hazard-Warning-9-by-h0us3s
[66] https://openclipart.org/detail/217065/3-oclock
[67] https://www.itmc.tu-dortmund.de/cms/de/home/anfahrt/anfahrt-hauptgebaeude/index.html
[68] https://imgflip.com/memegenerator/One-Does-Not-Simply
[69] https://wikis.uni-paderborn.de/pc2doc/Noctua-Software-TotalView