technische universität
dortmund

LiDO3

First Contact

IT & Medien Centrum | CC HPC

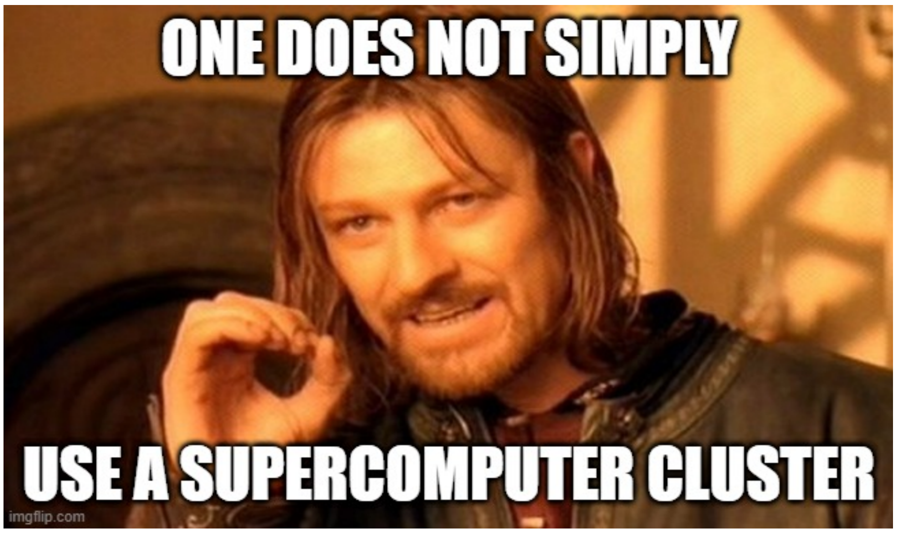
June 5, 2025

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, it is **not 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.

So, when bringing an application to LiDO3 that runs with a single thread on a processor (or, more precisely, on single compute core) you can not expect that the runtime of this application for a given set of input will be faster than on your local laptop or PC or server! In order to reach results faster, your application needs to be able to run in parallel. Either by means of shared-memory multiprocessing with threads[1] (using technologies like, e.g., OpenMP[2]) that allows an application to use more than one compute core in a compute node. Or by means of distributed-memory multiprocessing, built on top of the MPI library[3], that allows for an application to use several compute nodes. "Several" can be anything between two and several hundred compute nodes on LiDO3; on bigger sites (so-called Tier-2 and Tier-1 sites, read more on national HPC association NHR[4] and PRACE initiative[5]) the upper limit might be tens of thousands of compute nodes. For a serial, non-threaded application, on the other hand, the runtime on LiDO3 and any of those Tier-2/Tier-1 sites might even be slower than on your local compute hardware!

Given that LiDO3 is not a single big computer, but hundreds of mid-sized ones, the use of LiDO3 differs from the use of your laptop or PC or server:

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 resources (CPU/RAM) or, in the worst case, eat them away one another.

[1] https://en.wikipedia.org/wiki/Thread_(computing)#Threads_vs_processes
[2] https://en.wikipedia.org/wiki/Openmp
[3] https://en.wikipedia.org/wiki/Message_Passing_Interface
[4] https://de.wikipedia.org/wiki/Verbund_f%C3%BCr_Nationales_Hochleistungsrechnen
[5] https://de.wikipedia.org/wiki/Partnership_for_Advanced_Computing_in_Europe

This scheduler is called Slurm[6]. 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.

1.2 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[7] is not subject of this document. Neither is it a guide for structuring workload to scale on a HPC environment.

[6] https://en.wikipedia.org/wiki/Slurm
[7] 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 can be submitted by students and permanent employees of the *Technische Universität Dortmund*.

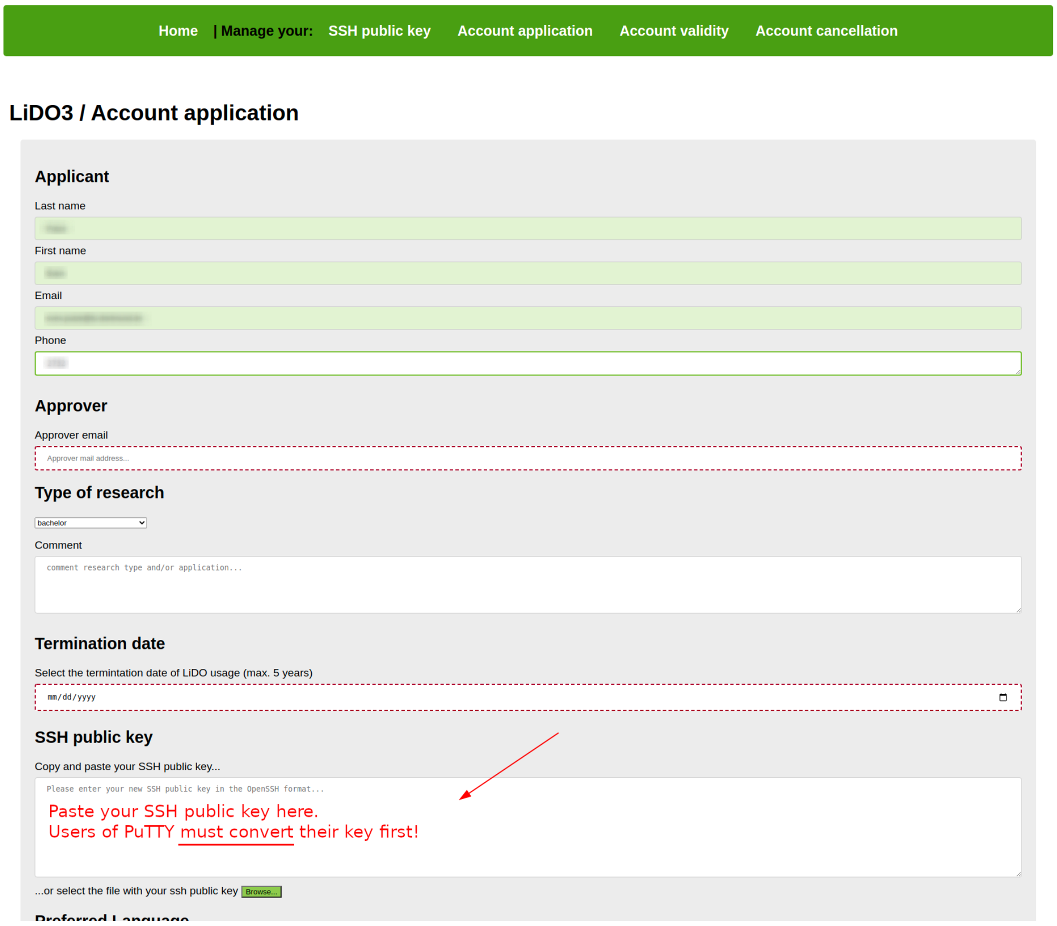
In most cases, students and employees of the *Technische Universität Dortmund* can use the LiDO3 usermanagement portal[1] to submit an application online.

In this application form, it is mandatory to provide information about the intended purpose of LiDO usage, termination date of LiDO usage, 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.

To minimize the attack surface for cyber attacks, the LiDO3 usermanagement portal is reachable from within the TU Dortmund University network only, i.e. your client machine must have an IP address assigned in the range between 129.217.0.1 and 129.217.255.255; if you connect from outside the university or are connected via Wifi network `eduroam`, first establish a VPN connection to `vpn.tu-dortmund.de`; single-sign-on login with uni account is mandatory).

[1] `https://l3umw.lido.tu-dortmund.de:8193/usermanagement/static/lido3-account-application-form.html`



Home | Manage your: SSH public key Account application Account validity Account cancellation

LiDO3 / Account application

Applicant

Last name

[redacted]

First name

[redacted]

Email

[redacted]

Phone

[redacted]

Approver

Approver email

Approver mail address...

Type of research

bachelor

Comment

```
comment research type and/or application...
```

Termination date

Select the termintation date of LiDO usage (max. 5 years)

mm/dd/yyyy

SSH public key

Copy and paste your SSH public key...

```
Please enter your new SSH public key in the OpenSSH format...
```

Paste your SSH public key here.
Users of PuTTY must convert their key first!

...or select the file with your ssh public key Browse...

Figure 2.1: Insert your generated public key into the *"SSH Public Key"* field to submit your public key.

Users of the PuTTY client must convert their key into the OpenSSH format first, see figure 2.4 on page 17.

To manage your existing LiDO3 user account, please use the web forms in the LiDO3 user management portal[2] To minimize the attack surface for cyber attacks, the LiDO3 usermanagement portal is reachable from within the TU Dortmund University network only, i.e. your client machine must have an IP address assigned in the range between 129.217.0.1 and 129.217.255.255; if you connect

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

from outside the university or are connected via Wifi network `eduroam`, first establish a VPN connection to `vpn.tu-dortmund.de`; single-sign-on login with uni account is mandatory).

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 open a ticket at the Service Desk[3]. Please note that this application can only be used by professors for their own projects, doctoral or post-doctoral research.

2.1.2 Approval

Upon submitting the application form[4], your approver (your supervising professor) will be informed 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, a ticket is generated in the ITMC ticket system that involves informing the LiDO team. If the approver does not react, a reminder will be sent every Monday after 7 days. If the approver still does not react, the LiDO team will notify you about the delay.

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 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 feasible within reasonable time – and on the other hand for user comfort.

[3]https://itmc.tu-dortmund.de/das-itmc/kontakt/service-desk/
[4]https://l3umw.lido.tu-dortmund.de:8193/usermanagement/static/lido3-account-application-form.html
[5]https://en.wikipedia.org/wiki/Public-key_cryptography

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 is entirely wrong:

- You are using a SSH client that does only support authentication via passwords, but maybe not via SSH keys. Or
- You used the SSH public key in your client instead of the SSH private key. Or
- The SSH public key entered in the LiDO3 account application web form (see 2.1) got scrambled (or its PuTTY file format representation was used instead of the canonical OpenSSH file format) such that your valid SSH private key does not match the scrambled SSH public key stored on LiDO3 any more, or
- The SSH private key you are using to connect to LiDO3 belongs to a SSH public key not stored (any more?) on LiDO3.

The internet is full of good tutorials that show *how to create and use a SSH key*. The approach is a bit different for Linux users[6] and for Windows/PuTTY users[7]. We will, hence, keep our tutorial short:

2.2.1 Create SSH key pair on Unix

Open a shell and enter

```
$ ssh-keygen -t rsa -b 4096 -C "comment helping you identify this
    ↪ key"
```

or

```
$ ssh-keygen -a 100 -t ed25519 -C "comment helping you identify
    ↪ this key"
```

If you already have other SSH key pairs, you can change the default filename in the following, otherwise just confirm the default by pressing the enter key.

[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

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

When prompted, type a secure passphrase to protect[8] your SSH private 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 SSH public key** into the user application form (see page 11) after the successful creation. Typically, the file containing the SSH public key is stored upon creation on your local system in a file with `.pub` file extension. Make sure to use the **SSH private key** when establishing a connection to the LiDO3 gateways with your SSH client software.

2.2.2 Create SSH key pair on Windows

2.2.2.1 OpenSSH client

With Windows Server 2022, Windows Server 2019, Windows 10 (build 1809 and later)[9] or newer) or Windows 11, an OpenSSH port is available in the command line prompt (in German-localized versions of Windows named 'Eingabeaufforderung'). This includes the program `ssh-keygen` described in the previous section 2.2.1. Its syntax is exactly the same. It will produce an SSH key pair in OpenSSH key format.

[8] If someone ever gains access to your computer (or to the files stored there, e.g. by accessing your backup drives), they also gain access to **every** system that uses your SSH key pair. To add an extra layer of security, you **should** use a passphrase to encrypt your SSH private key.

[9] You can check the version of Windows 10 by pressing `Win-key+R` and then invoke the command `winver`. An information dialog will pop up detailing the specific version of your Windows 10 installation, e.g. `Version 21H1 (Build 19043.2251)`.

2.2.2.2 PuTTY client (and derived software clients)

If you want to rely on a GUI-based solution, you can use the PuTTY Key Generator (`puttygen.exe`) from the PuTTY Software Suite.[10][11]

To create your SSH key pair for use on LiDO3 select either one of the four SSH key types "RSA", "DSA", "ECDSA", "ED25519". Please **do not use "SSH-1 (RSA)"**; the algorithm for this SSH key type is outdated and not supported any more on LiDO3 for security reasons.

To start the SSH key pair creation click on the button *Generate*.

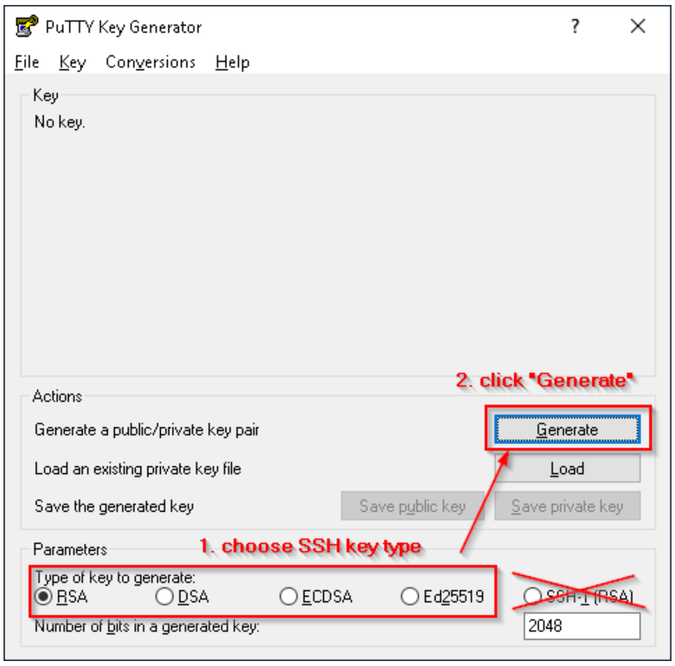


Figure 2.2: Choose SSH key type and click *Generate*.

For increased randomness in the generated SSH key, the user is required to move his mouse in random directions while the key is being generated. If you do not move your mouse around, the key generation will stall.

[10] https://www.chiark.greenend.org.uk/~sgtatham/putty/
[11] The MobaXterm SSH Key Generator is a direct clone of PuTTY Key Generator.

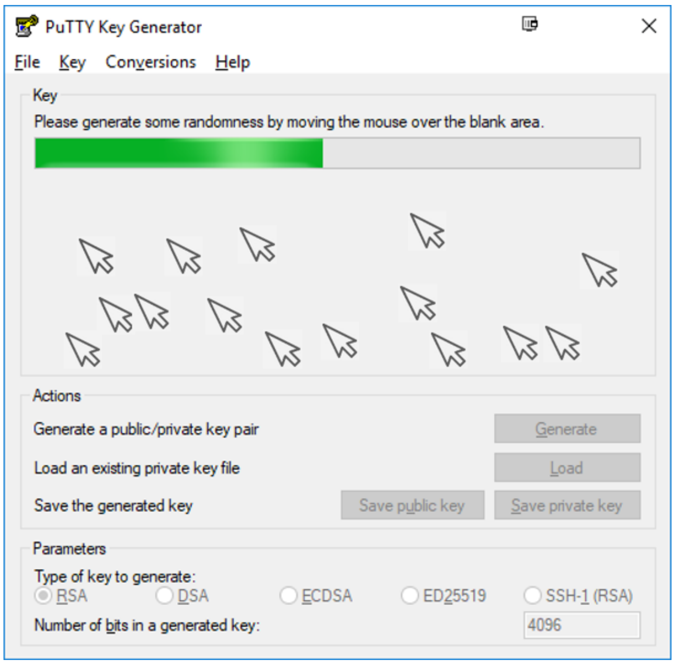
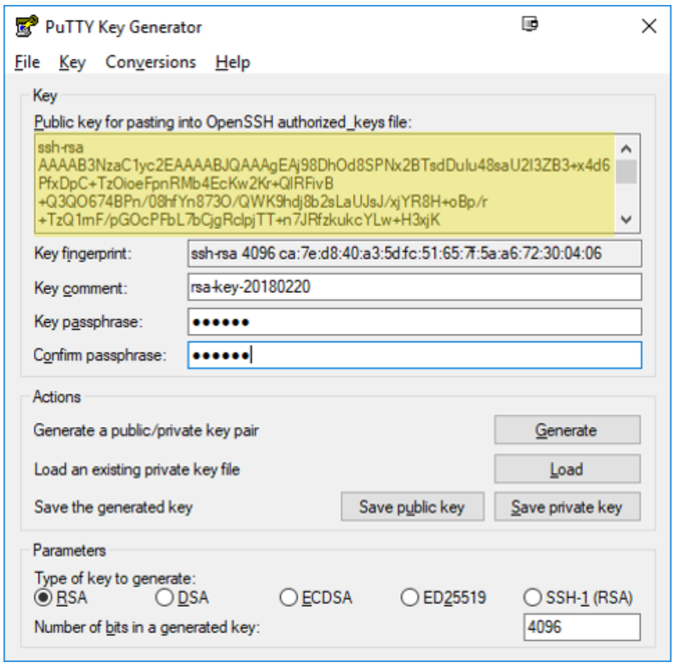


Figure 2.3: Random movements of the mouse pointer are used in order to create the key pair.

Once the SSH key pair has been generated, two steps remain: Firstly, store the SSH public key in a key file format that is understood by the OpenSSH server on LiDO3. Secondly, protect the SSH private key with a robust passphrase against abuse.

By default, PuTTY uses, unfortunately, its own propriatory key file formats, both for the SSH private key and the SSH public key. Hitting the button `Save public key` would store the SSH public key in PuTTY's propriatory key file format. For convenience, though, the SSH public key is also shown in OpenSSH key file format in the top section of the PuTTY Key Generator's main window, once it has been generated. Copy your freshly created SSH public key from the frame labeled `Public key ↪ for pasting into OpenSSH authorized_keys file` (see figure 2.4) and paste it into the LiDO3 account application web form (see figure 2.1 on page 11).



Public key for pasting into OpenSSH authorized_keys file:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAgEAj98DhOd8SPNx2BTsdDulu48saU2I3ZB3+x4d6
PfxDpC+TzOioeFpnRMb4EcKw2Kr+QIRFivB
+Q3QO674BPn/08hfYn873O/QWK9hdj8b2sLaUJsJ/xjYR8H+oBp/r
+TzQ1mF/pGOcPFbL7bCjgRclpjTT+n7JRfzkukcYLw+H3xjK
```

Key fingerprint: ssh-rsa 4096 ca:7e:d8:40:a3:5d:fc:51:65:7f:5a:a6:72:30:04:06

Key comment: rsa-key-20180220

Key passphrase: ••••••

Confirm passphrase: ••••••

Generate a public/private key pair

Load an existing private key file

Save the generated key

Type of key to generate:
[✓ RSA] DSA ECDSA ED25519 SSH-1 (RSA)

Number of bits in a generated key: 4096

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

A typical, valid SSH public keys (in OpenSSH key file format) starts with either one of the following strings. Please make sure you do not try to upload the SSH public key in PuTTY's key file format:

```
ssh-rsa AAAAB3NzaC1[...]
ssh-dss AAAAB3NzaC1[...]
ssh-ed25519 AAAAC3N[...]
```

The last step is to enter a passphrase which is later used to protect your SSH private key in the text field labeled `Key passphrase` and then hit the button `Save ↪ private key`. Select a directory to your liking to save the SSH private key and make sure nobody else has access to this directory. You will require this file in the future every time you intend to connect to the LiDO3 gateways.

If you intend to use your SSH private key with programs other than PuTTY, programs that use the OpenSSH key file format, e.g. with ThinLinc (see section 4.2.4), you are advised to save a copy of your SSH private key in OpenSSH key file format right away. Convert it via menu `Conversions`→`Export OpenSSH key` to a new file. (The conversion can be done at a later time by first loading your SSH private key in PuTTY key file format (file extension `*.ppk`) as well.)

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[12].To minimize the attack surface for cyber attacks, the LiDO3 usermanagement portal is reachable from within the TU Dortmund University network only, i.e. your client machine must have an IP address assigned in the range between 129.217.0.1 and 129.217.255.255; if you connect from outside the university or are connected via Wifi network `eduroam`, first establish a VPN connection to `vpn.tu-dortmund.de`; single-sign-on login with uni account is mandatory).

[12]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 partially 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 Deutsche
Forschungsgemeinschaft (DFG, German Research Foundation) 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.

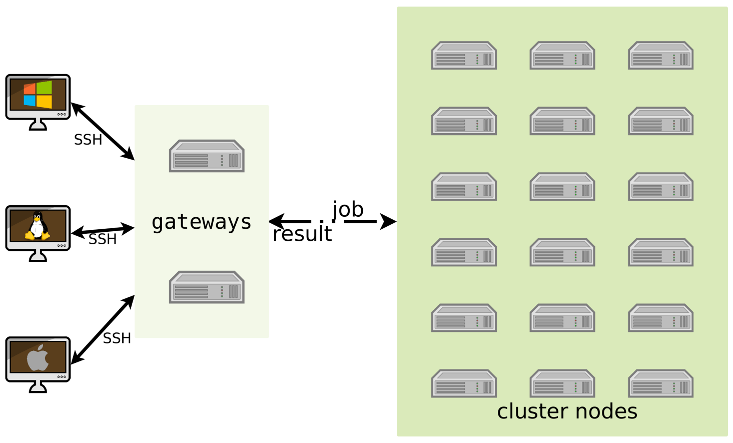


Figure 4.1: Clients connect to one of the gateway servers and transmit jobs.

[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

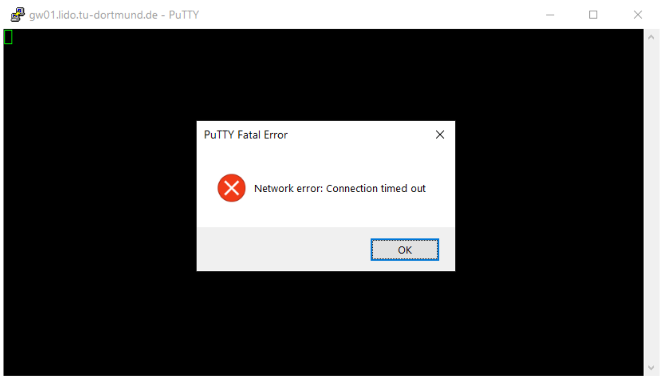
[2]http://en.wikipedia.org/wiki/Secure_Shell

[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.



Network error: Connection timed out

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 whether the key fingerprint of 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. Key fingerprints are long alphanumeric strings which are notoriously difficult to compare. For this, their – much shorter and for this easier to compare – hash value is calculated (typically using the cryptographic hash functions MD5 or SHA256, depending on your particular SSH client) and shown.

The correct key fingerprints are as follows:

For Gateway 1:

```
$ 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
2048 SHA256:rG0Cmye6DibyWvaqjHcma6vnwsvTfYATy1JM/O200Ns
    ↪ gw01.lido.tu-dortmund.de (RSA)
256 SHA256:SxL75DVFyNKVbSMkB1M/fPTy5qcPtWa5M9iHHe9OETU
    ↪ gw01.lido.tu-dortmund.de (ECDSA)
256 SHA256:lUQLD2VY/pTVpsSPwuUwvHA8jm/tNiGJ+GbaHP9sBPo
    ↪ gw01.lido.tu-dortmund.de (ED25519)
```

For Gateway 2:

```
$ 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
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
256 SHA256:sYjJMuRut7jSomxbluWOf0YKE1y5QE5esAQovRBveHo
    ↪ gw02.lido.tu-dortmund.de (ED25519)
```

When using an older version of PuTTY, the fingerprints may still be given in the MD5 format instead of the SHA256 format.

For Gateway 1:

```
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)
```

For Gateway 2:

```
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 came with no built-in SSH[11]-client-software, so you had to download and install a third-party tool:

- PuTTY (4.2.2.2), a well known, free and sufficient, but purely text-based SSH client
- Cendio Thinlinc (4.2.4), a remote desktop application (i.e. that allows running graphical applications) that encrypts all network traffic between client and server via SSH protocol
- MobaXterm (4.2.2.4), a fork based on PuTTY, with integrated X11 server that allows running graphical applications, tabs for multiple concurrent SSH connections and network tools.
- OpenSSH from the Cygwin project[12], another free, sufficient, text-based SSH client
- Bitvise SSH client[13], free for use of all types, including in organizations

[11] http://en.wikipedia.org/wiki/Secure_Shell
[12] http://cygwin.org/
[13] https://www.bitvise.com/

With Windows Server 2022, Windows Server 2019, Windows 10 (build 1809 and later)[14] or newer) or Windows 11, there are two additional options:

- These operating systems ship with a text-based SSH client[15] named `ssh.exe`. It is a port of OpenSSH described in the previous section 4.2.1. To use `ssh` first open a command prompt[16] and from the command prompt window, invoke `ssh`.
- Install a Linux subsystem[17] and use that to start a connection.

Since Windows users may not be used to connect to other computers via SSH[18], we will describe a few commonly used SSH clients in more detail here. Of course, you can use other SSH client software if that suits you better.

4.2.2.1 OpenSSH for Windows

With Windows Server 2022, Windows Server 2019, Windows 10 (build 1809 and later)[19] or newer) or Windows 11, ensure you are connected to the TU Dortmund network (directly or via VPN), open the command prompt (in German-localized versions of Windows named 'Eingabeaufforderung') and invoke

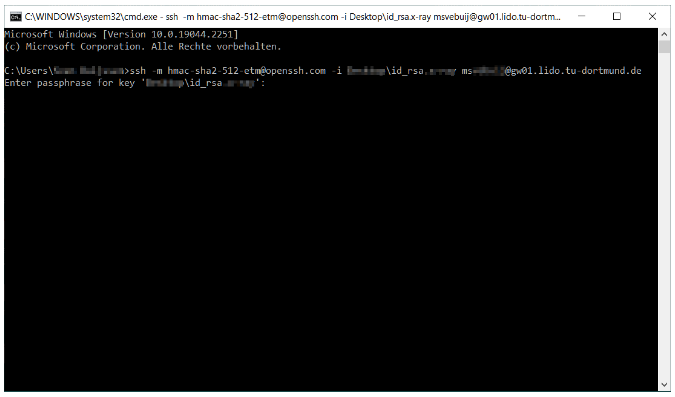
```
$ ssh -m hmac-sha2-512-etm@openssh.com -i
    ↪ \path\to\SSH\private\key\in\openssh\key\format
    ↪ <username>@gw01.lido.tu-dortmund.de
```

Listing 4.1: Command line to connect from Windows command prompt with OpenSSH_for_Windows to LiDO3 gateway server

but make sure to replace the correct path to your SSH private key and your username in listing 4.1.

If you secured your SSH private key with a passphrase (which is highly recommended!), you will be prompted for it now (see figure 4.3).

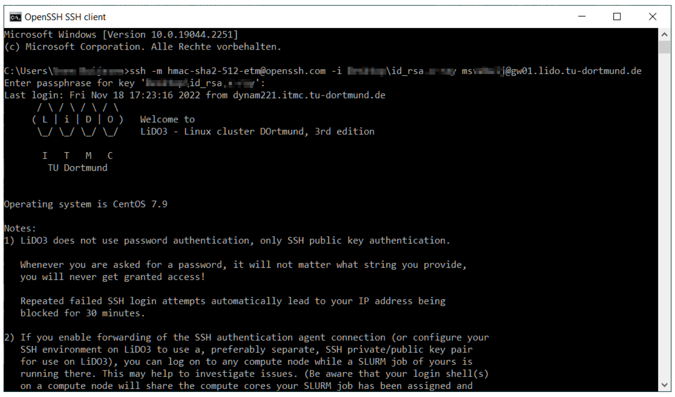
[14]You can check the version of Windows 10 by pressing `Win-key+R` and then invoke the command `winver`. An information dialog will pop up detailing the specific version of your Windows 10 installation, e.g. `Version 21H1 (Build 19043.2251)`.
[15]https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview
[16]Press `Win-key+R` and then invoke the command `cmd.exe`
[17]https://docs.microsoft.com/en-us/windows/wsl/install-win10
[18]http://en.wikipedia.org/wiki/Secure_Shell
[19]You can check the version of Windows 10 by pressing `Win-key+R` and then invoke the command `winver`. An information dialog will pop up detailing the specific version of your Windows 10 installation, e.g. `Version 21H1 (Build 19043.2251)`.



```
C:\WINDOWS\system32\cmd.exe - ssh  -m hmac-sha2-512-etm@openssh.com -i Desktop\id_rsa.x-ray msvebuij@gw01.lido.tu-dortm...
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\[redacted]>ssh -m hmac-sha2-512-etm@openssh.com -i [redacted]\id_rsa[redacted] ms[redacted]@gw01.lido.tu-dortmund.de
Enter passphrase for key '[redacted]\id_rsa[redacted]':
```

Figure 4.3: Create a new text file named "config" in directory `%userprofile%\.ssh`

If your SSH private key is accepted, you will be subsequently logged in to LiDO3 (see figure 4.4).



```
OpenSSH SSH client
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\[redacted]>ssh -m hmac-sha2-512-etm@openssh.com -i [redacted]\id_rsa[redacted] ms[redacted]j@gw01.lido.tu-dortmund.de
Enter passphrase for key '[redacted]id_rsa[redacted]':
Last login: Fri Nov 18 17:23:16 2022 from dynam221.itmc.tu-dortmund.de
      / \ / \ / \ / \
     ( L | i | D | O )   Welcome to
      \_/ \_/ \_/ \_/    LiDO3 - Linux cluster DOrtmund, 3rd edition

       I   T   M   C
        TU Dortmund


Operating system is CentOS 7.9

Notes:
1) LiDO3 does not use password authentication, only SSH public key authentication.

   Whenever you are asked for a password, it will not matter what string you provide,
   you will never get granted access!

   Repeated failed SSH login attempts automatically lead to your IP address being
   blocked for 30 minutes.

2) If you enable forwarding of the SSH authentication agent connection (or configure your
   SSH environment on LiDO3 to use a, preferably separate, SSH private/public key pair
   for use on LiDO3), you can log on to any compute node while a SLURM job of yours is
   running there. This may help to investigate issues. (Be aware that your login shell(s)
   on a compute node will share the compute cores your SLURM job has been assigned and
```

Figure 4.4: Create a new text file named "config" in directory `%userprofile%\.ssh`

You can reduce the lengthy command line from listing 4.1 by either creating a so-called batch file (a simple text file with the file extension `.bat`), e.g. `sshlido3.bat`. Make sure to insert the entire command line from listing 4.1, but replace the correct path to your SSH private key and your username. Subsequently, you only need to invoke this batch file to log in to LiDO3:

```
> \path\to\your\batch\file\sshlido3.bat
```

Or you create a SSH config file. The latter will allow you to log in from your command prompt via

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

Listing 4.2: Command line to connect from Windows command prompt with OpenSSH_for_Windows to LiDO3 gateway server, implicitly making use of a ssh config file

To create a SSH config file, point your Windows explorer to the directory `\%userprofile` and create a new text file there named `config` (see figure 4.5):

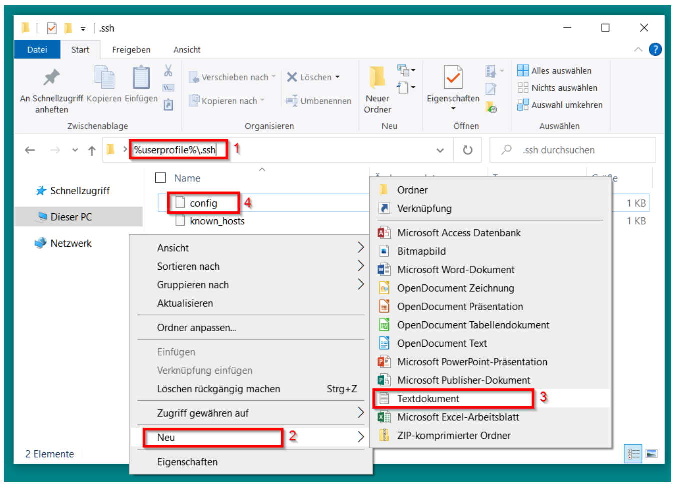
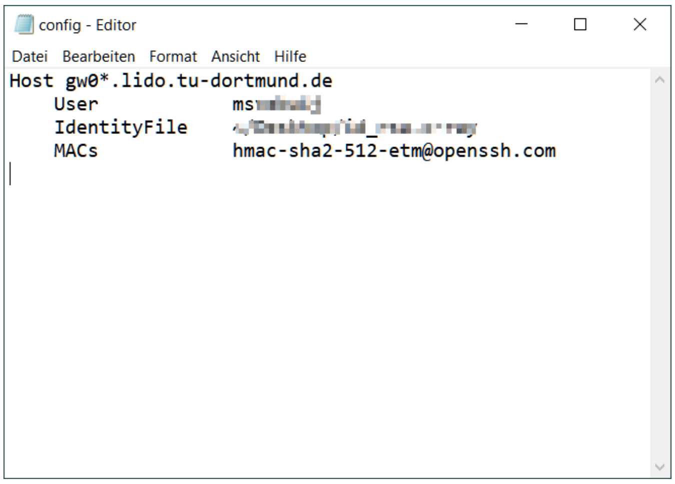


Figure 4.5: Create a new text file named "config" in directory `%userprofile%\.ssh`

Its content differs from that of the batch file, but is more flexible as it configures access for both LiDO3 gateway servers at the same time. Again replace in figure 4.6 the correct path to your SSH private key and your username.



```
Host gw0*.lido.tu-dortmund.de
    User            ms[redacted]
    IdentityFile    [redacted]
    MACs            hmac-sha2-512-etm@openssh.com
```

Figure 4.6: Example of a OpenSSH config file

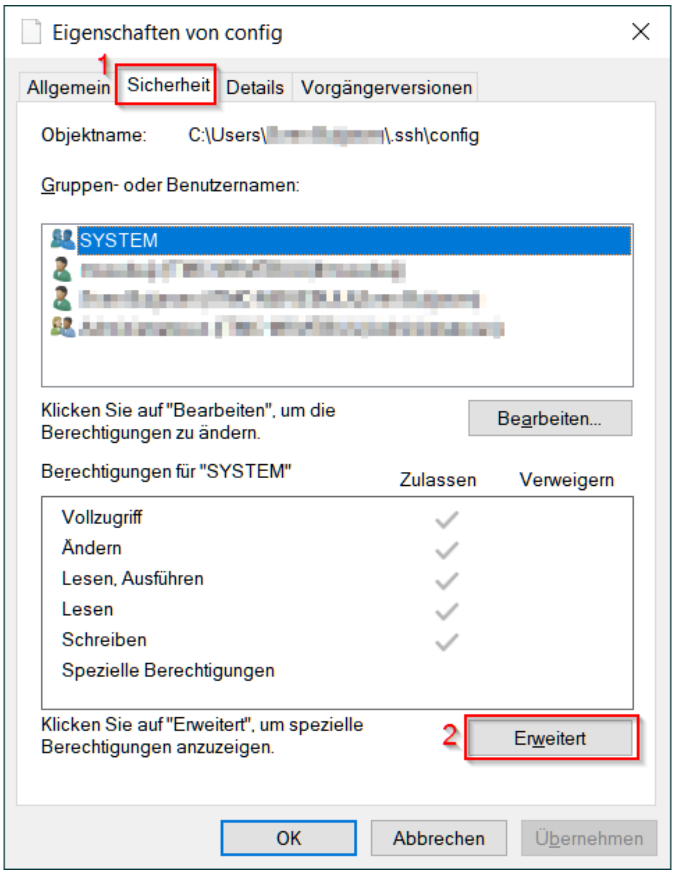
When you try to log in to LiDO3 for the first time with OpenSSH, `ssh` might complain about inadequate file permissions for your SSH private key, e.g.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions for '\path\to\SSH\private\key\in\openssh\key\format'
are too open. It is required that your private key files are NOT
accessible by others. This private key will be ignored.
Load key "\path\to\SSH\private\key\in\openssh\key\format": bad
   ↪ permissions
```

or the `config` file, e.g.

```
> ssh gw01.lido.tu-dortmund.de
%% Bad owner or permissions on /path/to/user/profile/.ssh/config
```

If it does, you will not be able to log in. The remedy for both files is to disable inheritance of file permissions for each of these files and to remove read permissions for all system users and administrators but yourself. The procedure is to do this is as follows: from the context menu of your SSH private key file and/or the SSH config file select the last menu item `Properties` (in German-localized versions of Windows named 'Eigenschaften'). A dialog window will open. Select the second tab labeled `Security` and click the button `Advanced` (see figure 4.7).



Eigenschaften von config

1 Allgemein | Sicherheit | Details | Vorgängerversionen

Objektname: C:\Users\[redacted]\.ssh\config

Gruppen- oder Benutzernamen:

SYSTEM
[redacted]
[redacted]
[redacted]

Klicken Sie auf "Bearbeiten", um die Berechtigungen zu ändern. — Bearbeiten...

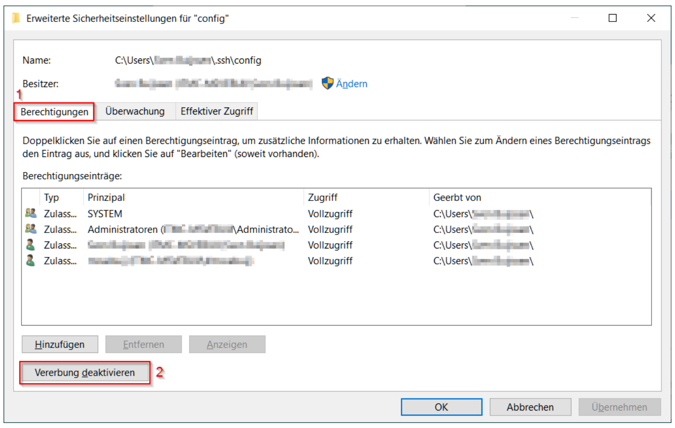
Berechtigungen für "SYSTEM"	Zulassen	Verweigern
Vollzugriff	✓	
Ändern	✓	
Lesen, Ausführen	✓	
Lesen	✓	
Schreiben	✓	
Spezielle Berechtigungen		

Klicken Sie auf "Erweitert", um spezielle Berechtigungen anzuzeigen. — 2 Erweitert

OK | Abbrechen | Übernehmen

Figure 4.7: Impose sufficiently restrictive file permissions for OpenSSH (step 1)

Another dialog window will open. In its first tab click the button `Disable inheritance` (see figure 4.8). You will subsequently be asked whether you want to `Replace all ↪ child object permissions entries with inheritable permission ↪ entries from this object`. Confirm this choice.


Erweiterte Sicherheitseinstellungen für "config"

Name: C:\Users\[redacted]\.ssh\config

Besitzer: [redacted] Ändern

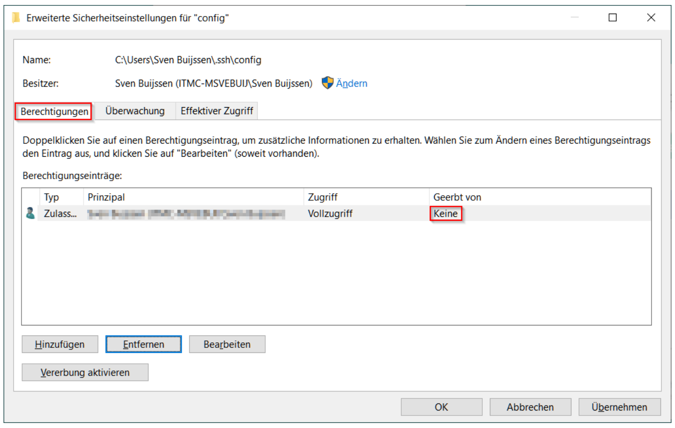
Doppelklicken Sie auf einen Berechtigungseintrag, um zusätzliche Informationen zu erhalten. Wählen Sie zum Ändern eines Berechtigungseintrags den Eintrag aus, und klicken Sie auf "Bearbeiten" (soweit vorhanden).

Berechtigungseinträge:

Typ	Prinzipal	Zugriff	Geerbt von
Zulass...	SYSTEM	Vollzugriff	C:\Users\[redacted]\
Zulass...	Administratoren ([redacted]\Administrato...	Vollzugriff	C:\Users\[redacted]\
Zulass...	[redacted]	Vollzugriff	C:\Users\[redacted]\
Zulass...	[redacted]	Vollzugriff	C:\Users\[redacted]\

Figure 4.8: Impose sufficiently restrictive file permissions for OpenSSH (step 2)

Subsequently, remove all permission entries except for one that refers to your Windows username. In other words, remove all permissions for system accounts, administrator roles and other accounts but your own account (see figure 4.9. Save the changes by click the `OK` button of this dialog window and again the `OK` button of the file properties dialog window.


Erweiterte Sicherheitseinstellungen für "config"

Name: C:\Users\Sven Buijssen\.ssh\config

Besitzer: Sven Buijssen (ITMC-MSVEBUIJ\Sven Buijssen) Ändern

Doppelklicken Sie auf einen Berechtigungseintrag, um zusätzliche Informationen zu erhalten. Wählen Sie zum Ändern eines Berechtigungseintrags den Eintrag aus, und klicken Sie auf "Bearbeiten" (soweit vorhanden).

Berechtigungseinträge:

Typ	Prinzipal	Zugriff	Geerbt von
Zulass...	[redacted]	Vollzugriff	Keine

Figure 4.9: Impose sufficiently restrictive file permissions for OpenSSH (step 3)

Now, you will be able to log in as shown in listing 4.2.

4.2.2.2 PuTTY

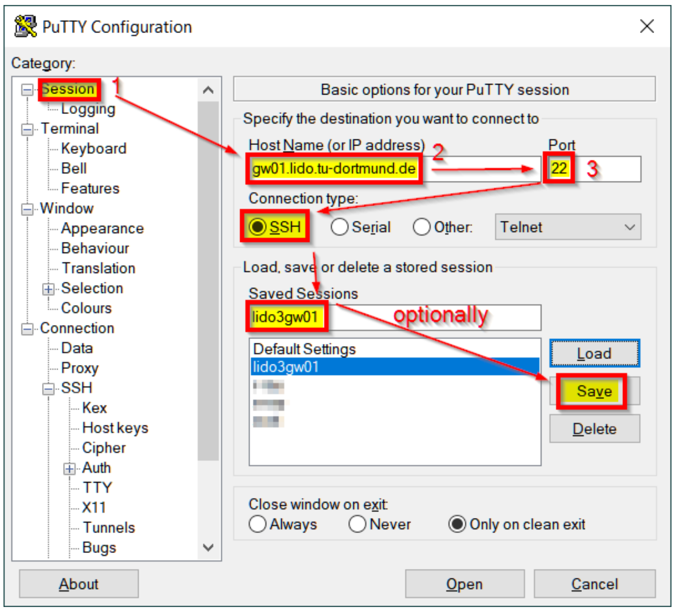
Download the latest version of PuTTY[20] and install it on your Windows client.

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

When you fire up PuTTY for the first time in order to connect to the LiDO3 cluster, make the following changes to its configuration screen:

- In category `Session` specify as hostname one of the gateway servers, i.e. `gw01.lido.tu-dortmund.de` or `gw02.lido.tu-dortmund.de` (see figure 4.10), use port 22 and connection type `SSH`.

  To avoid having to reconfigure PuTTY every time you are about to connect to the LiDO3 cluster, provide a name in the text field `Saved Sessions` and save your settings by hitting the `Save` button. (Do not forget to return here once you have made other changes to the configuration in order to save the updated settings!)



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): gw01.lido.tu-dortmund.de

Port: 22

Connection type: [✓ SSH] Serial Other: Telnet

Load, save or delete a stored session

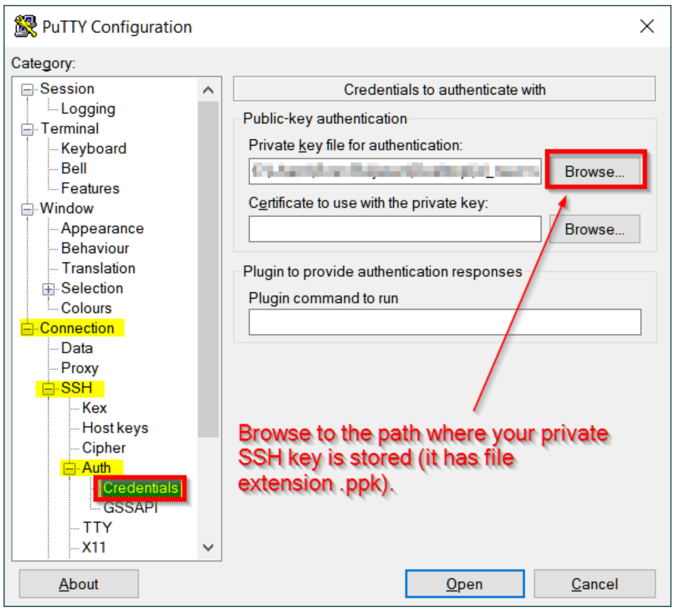
Saved Sessions: lido3gw01

Default Settings
lido3gw01

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

Figure 4.10: Mandatory: Enter the gateway name, path to your private SSH key

- In category `Connection→SSH→Auth→Credentials` enter the path to your private *SSH Key* (see page 14) (see figure 4.11).



Private key file for authentication: [redacted]

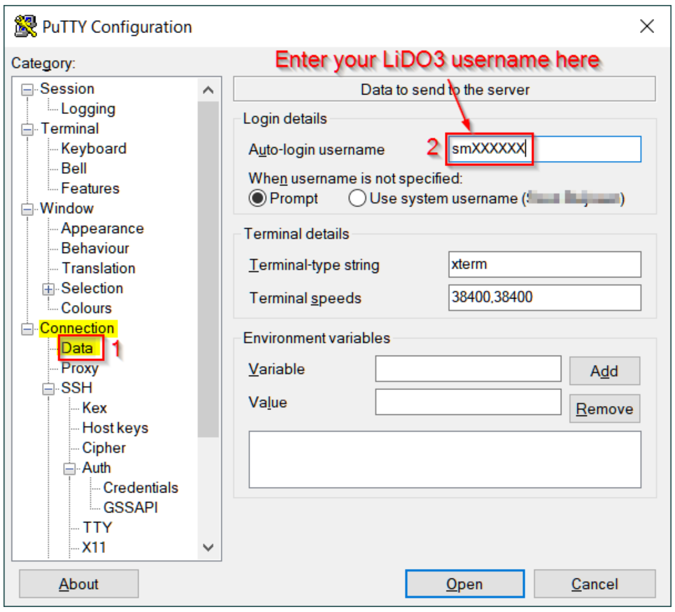
Certificate to use with the private key:

Plugin command to run:

Browse to the path where your private SSH key is stored (it has file extension .ppk).

Figure 4.11: Mandatory: Enter the path and filename to your personal private SSH-key

- In category `Connection→Data` enter your LiDO3 username (see figure 4.12). If you do not provide it here, you will be asked for your username every time your log in (see figure 4.13).



Enter your LiDO3 username here

Auto-login username: smXXXXXX

When username is not specified:
[✓ Prompt] Use system username ([redacted])

Terminal-type string: xterm

Terminal speeds: 38400,38400

Variable:

Value:

Figure 4.12: Recommended: hardcode your LiDO3 username

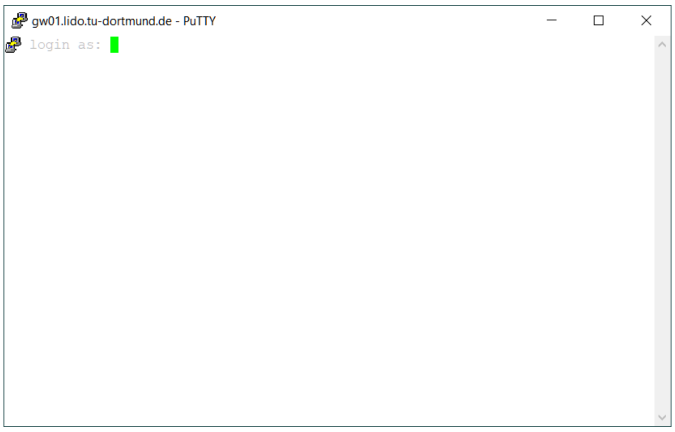
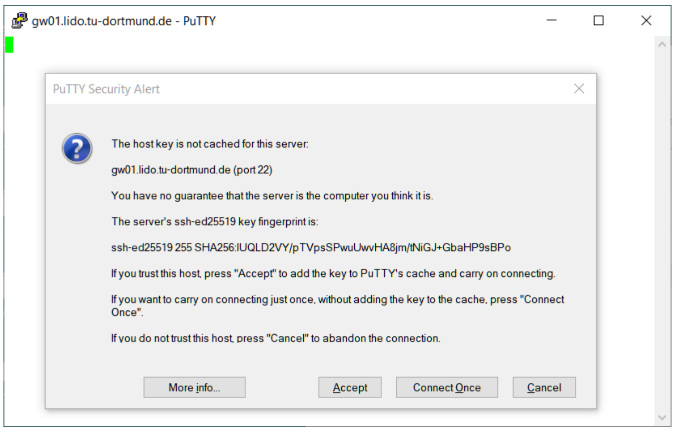


Figure 4.13: PuTTY will ask for your LiDO3 username if auto-login username in figure 4.12 is left unconfigured

Once you have made and saved all these changes in a session profile (see figure 4.10), you can now establish your first encrypted login session to the LiDO3 cluster by clicking the `Open` button of the PuTTY configuration dialog.



```
The host key is not cached for this server:

gw01.lido.tu-dortmund.de (port 22)

You have no guarantee that the server is the computer you think it is.

The server's ssh-ed25519 key fingerprint is:

ssh-ed25519 255 SHA256:lUQLD2VY/pTVpsSPwuUwvHA8jm/tNiGJ+GbaHP9sBPo

If you trust this host, press "Accept" to add the key to PuTTY's cache and carry on connecting.

If you want to carry on connecting just once, without adding the key to the cache, press "Connect Once".

If you do not trust this host, press "Cancel" to abandon the connection.
```

Figure 4.14: PuTTY will report an unknown identity when connecting for the first time to a new host and ask for confirmation before proceeding.

If you connect to one of the gateway servers for the first time, you will be asked whether the key fingerprint of 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. Key fingerprints are long alphanumeric strings which are notoriously difficult to compare. For this, their – much shorter and for this easier to compare – hash value is calculated (typically using the cryptographic hash functions MD5 or SHA256, depending on your particular SSH client) and shown.

The correct key fingerprints are as follows:

For Gateway 1:

```
$ 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
2048 SHA256:rG0Cmye6DibyWvaqjHcma6vnwsvTfYATy1JM/O200Ns
    ↪ gw01.lido.tu-dortmund.de (RSA)
256 SHA256:SxL75DVFyNKVbSMkB1M/fPTy5qcPtWa5M9iHHe9OETU
    ↪ gw01.lido.tu-dortmund.de (ECDSA)
256 SHA256:lUQLD2VY/pTVpsSPwuUwvHA8jm/tNiGJ+GbaHP9sBPo
    ↪ gw01.lido.tu-dortmund.de (ED25519)
```

For Gateway 2:

```
$ 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
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
256 SHA256:sYjJMuRut7jSomxbluWOf0YKE1y5QE5esAQovRBveHo
    ↪ gw02.lido.tu-dortmund.de (ED25519)
```

When using an older version of PuTTY, the fingerprints may still be given in the MD5 format instead of the SHA256 format.

For Gateway 1:

```
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)
```

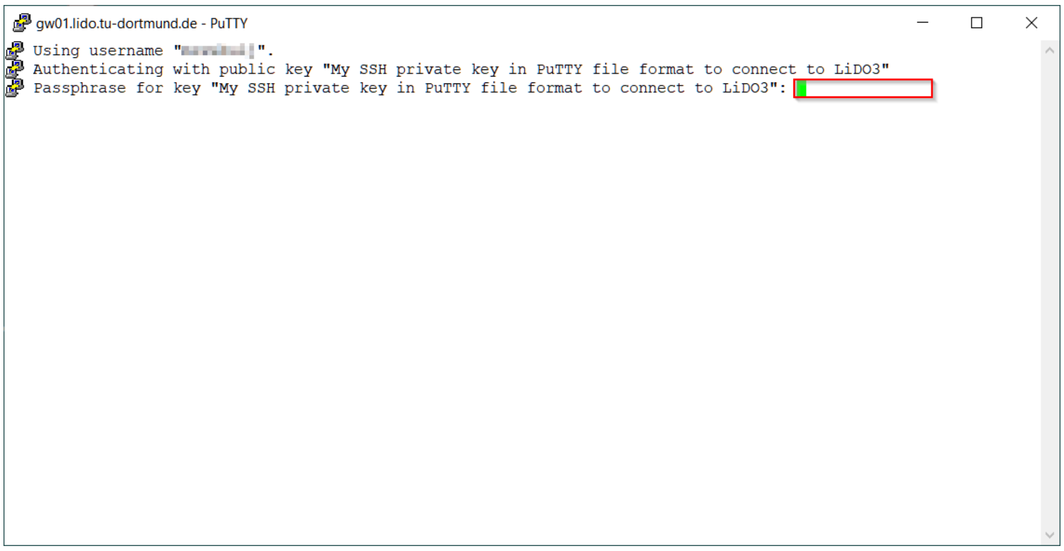
For Gateway 2:

```
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)
```

If the key fingerprints that PuTTY reports do not match those listed in this document, do not proceed! Instead contact the LiDO team over the phone, via email or open a support ticket with the ITMC service desk.

If you have confirmed the correct identity, accept the key with a click on the button `Accept` (see figure 4.14).

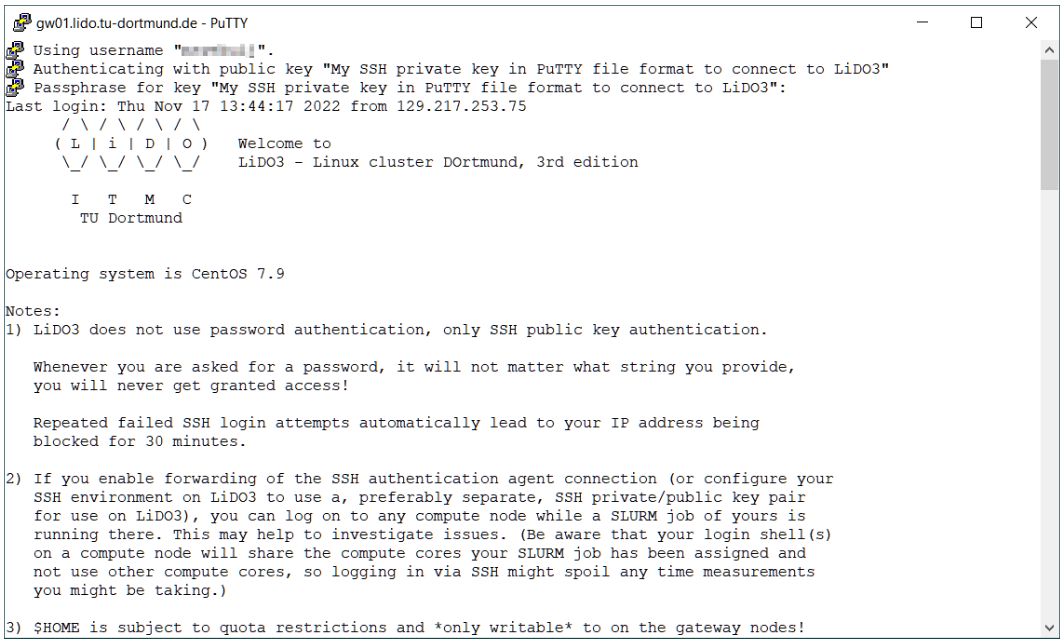
If you have encrypted your SSH private key with a passphrase, you will now be asked to enter it (see figure 4.15). There will be no visual feedback on how many and what keys you type to enter your passphrase. Press the `[Enter]` key once your have entered the passphrase.



```
Using username "[redacted]".
Authenticating with public key "My SSH private key in PuTTY file format to connect to LiDO3"
Passphrase for key "My SSH private key in PuTTY file format to connect to LiDO3":
```

Figure 4.15: SSH private key, if you chose to encrypt it (recommended!), needs to be unlocked during login to LiDO3

Now you are logged in to LiDO3. Welcome to the world of high performance computing.



gw01.lido.tu-dortmund.de - PuTTY

```
Using username "[redacted]".
Authenticating with public key "My SSH private key in PuTTY file format to connect to LiDO3"
Passphrase for key "My SSH private key in PuTTY file format to connect to LiDO3":
Last login: Thu Nov 17 13:44:17 2022 from 129.217.253.75
      / \ / \ / \ / \
     ( L | i | D | O )    Welcome to
      \_/ \_/ \_/ \_/     LiDO3 - Linux cluster DOrtmund, 3rd edition

       I   T   M   C
        TU Dortmund


Operating system is CentOS 7.9

Notes:
1) LiDO3 does not use password authentication, only SSH public key authentication.

   Whenever you are asked for a password, it will not matter what string you provide,
   you will never get granted access!

   Repeated failed SSH login attempts automatically lead to your IP address being
   blocked for 30 minutes.

2) If you enable forwarding of the SSH authentication agent connection (or configure your
   SSH environment on LiDO3 to use a, preferably separate, SSH private/public key pair
   for use on LiDO3), you can log on to any compute node while a SLURM job of yours is
   running there. This may help to investigate issues. (Be aware that your login shell(s)
   on a compute node will share the compute cores your SLURM job has been assigned and
   not use other compute cores, so logging in via SSH might spoil any time measurements
   you might be taking.)

3) $HOME is subject to quota restrictions and *only writable* to on the gateway nodes!
```

Figure 4.16: Succesful login to one of the LiDO3 gateways.

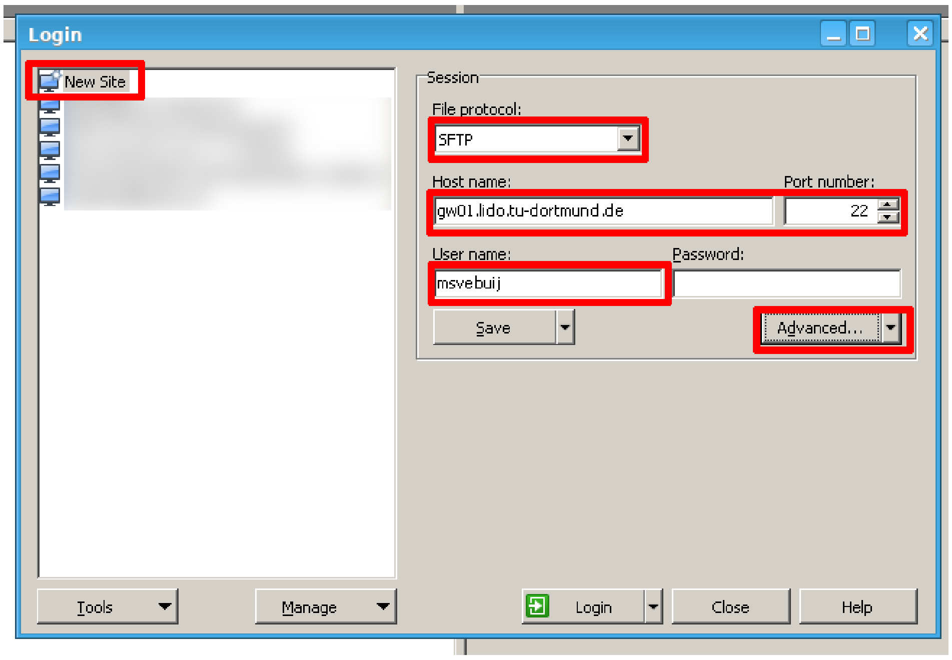
You can end the PuTTY session with the command `exit`.

4.2.2.3 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[21], i.e. an encrypted file transfer over the network. A widely used `scp` GUI for Windows is called WinSCP[22]. 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').

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


Login

New Site

Session

File protocol: SFTP

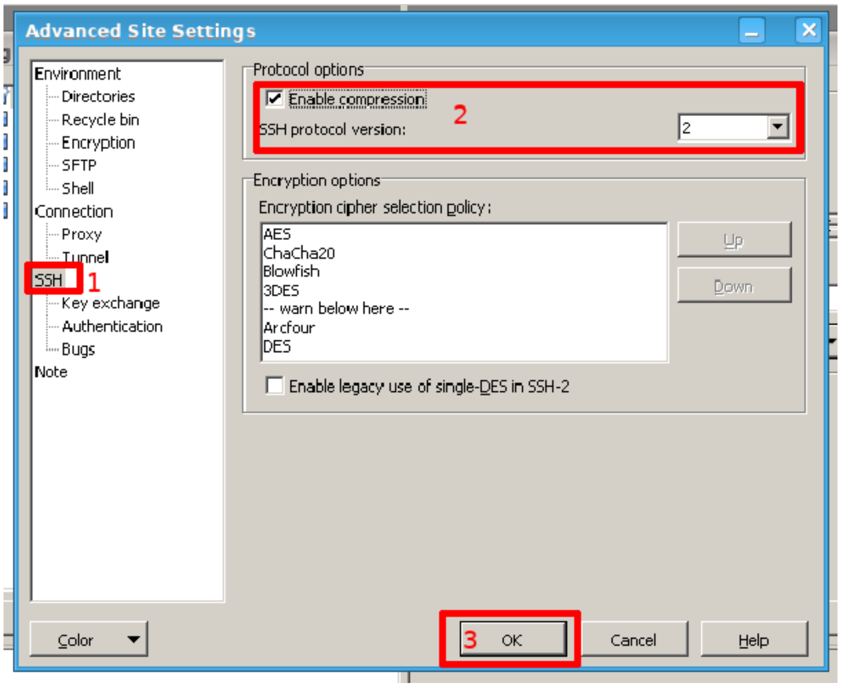
Host name: gw01.lido.tu-dortmund.de

Port number: 22

User name: msvebuij

Password:

Figure 4.17: Setting up protocol, server and username and opening advanced settings


Advanced Site Settings

Protocol options

☑ Enable compression

SSH protocol version: 2

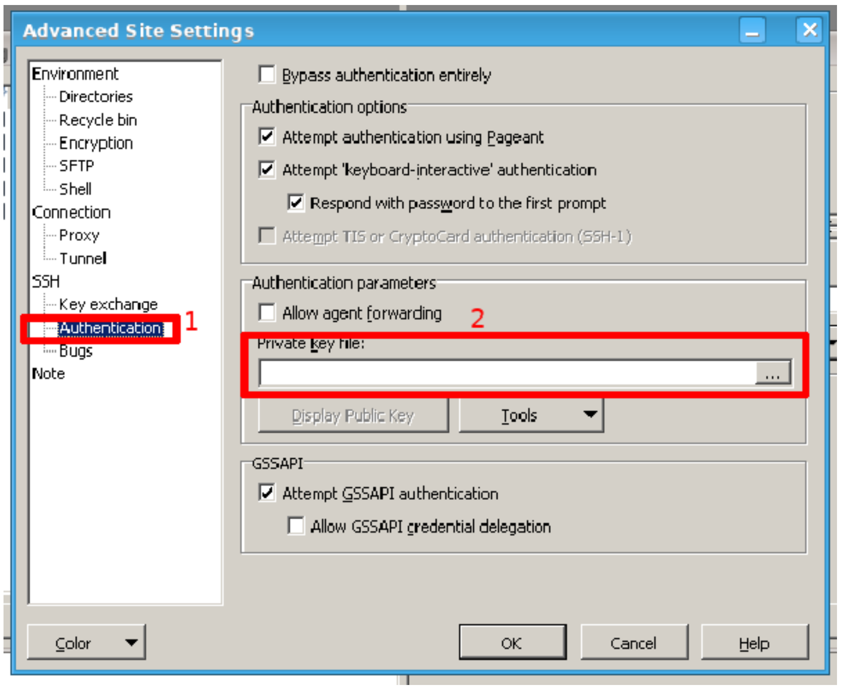
Encryption options

Encryption cipher selection policy:

AES
ChaCha20
Blowfish
3DES
-- warn below here --
Arcfour
DES

☐ Enable legacy use of single-DES in SSH-2

Figure 4.18: Enable SSH compression.


Advanced Site Settings

Environment, Directories, Recycle bin, Encryption, SFTP, Shell, Connection, Proxy, Tunnel, SSH, Key exchange, Authentication, Bugs, Note

1

- [ ] Bypass authentication entirely

Authentication options
- [x] Attempt authentication using Pageant
- [x] Attempt 'keyboard-interactive' authentication
- [x] Respond with password to the first prompt
- [ ] Attempt TIS or CryptoCard authentication (SSH-1)

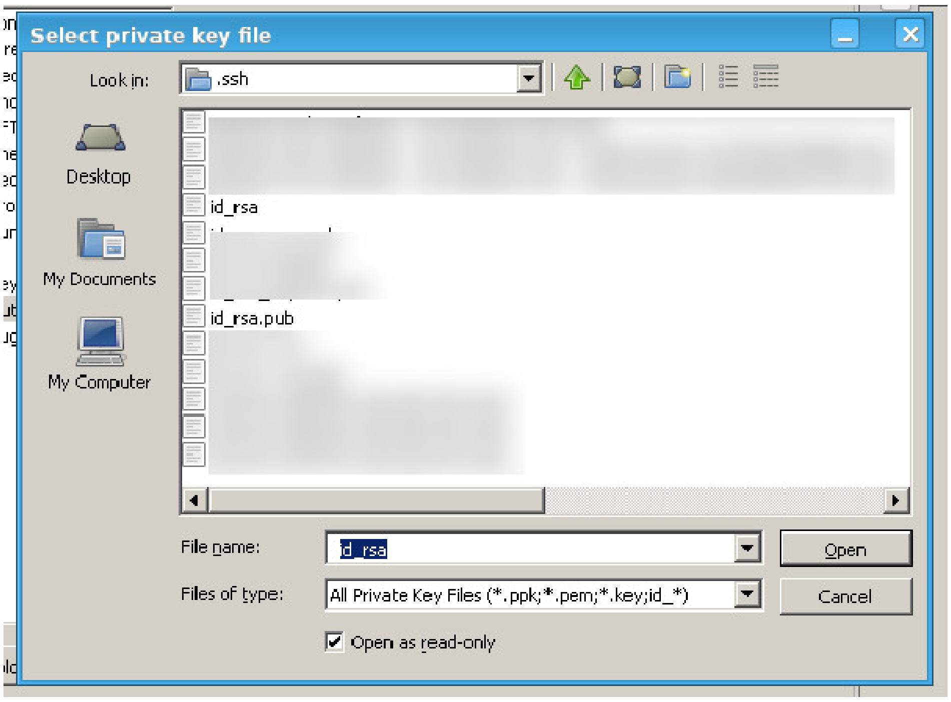
Authentication parameters
- [ ] Allow agent forwarding

2

Private key file:

GSSAPI
- [x] Attempt GSSAPI authentication
- [ ] Allow GSSAPI credential delegation

Figure 4.19: Open private key selection.


Select private key file

Look in: .ssh

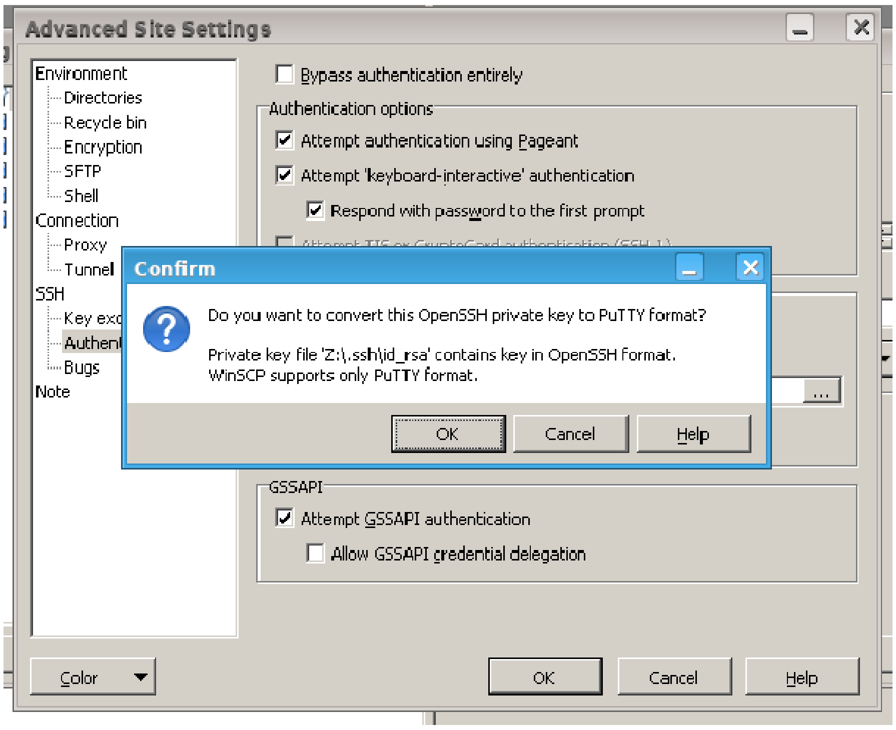
id_rsa
id_rsa.pub

File name: id_rsa

Files of type: All Private Key Files (*.ppk;*.pem;*.key;id_*)

- [x] Open as read-only

Figure 4.20: Select private key.


Advanced Site Settings
Environment
Directories
Recycle bin
Encryption
SFTP
Shell
Connection
Proxy
Tunnel
SSH
Key exc
Authent
Bugs
Note
Bypass authentication entirely
Authentication options
Attempt authentication using Pageant
Attempt 'keyboard-interactive' authentication
Respond with password to the first prompt
GSSAPI
Attempt GSSAPI authentication
Allow GSSAPI credential delegation
Color
OK
Cancel
Help
Confirm
OK
Cancel
Help

Do you want to convert this OpenSSH private key to PuTTY format?

Private key file 'Z:\.ssh\id_rsa' contains key in OpenSSH format.
WinSCP supports only PuTTY format.

Figure 4.21: Confirm private key conversion.

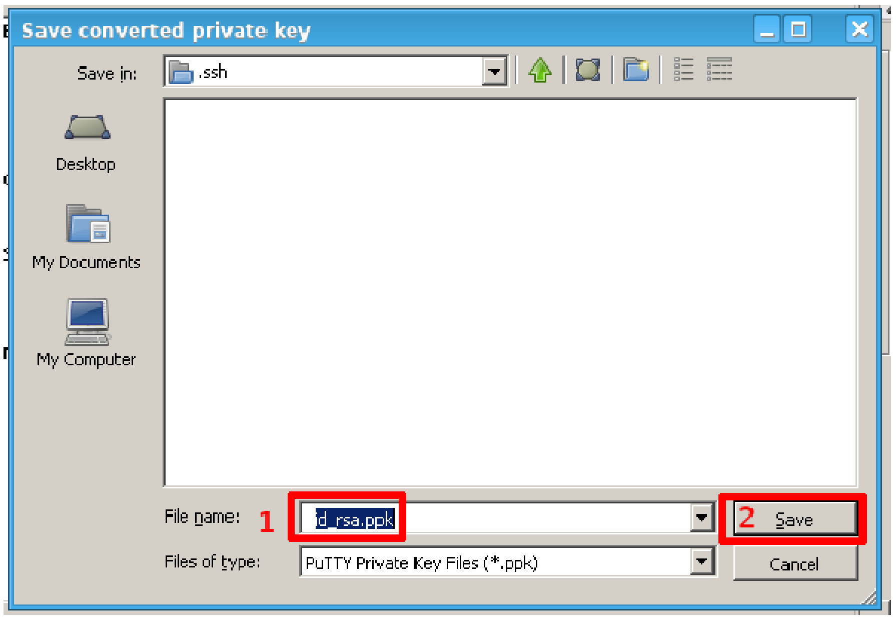
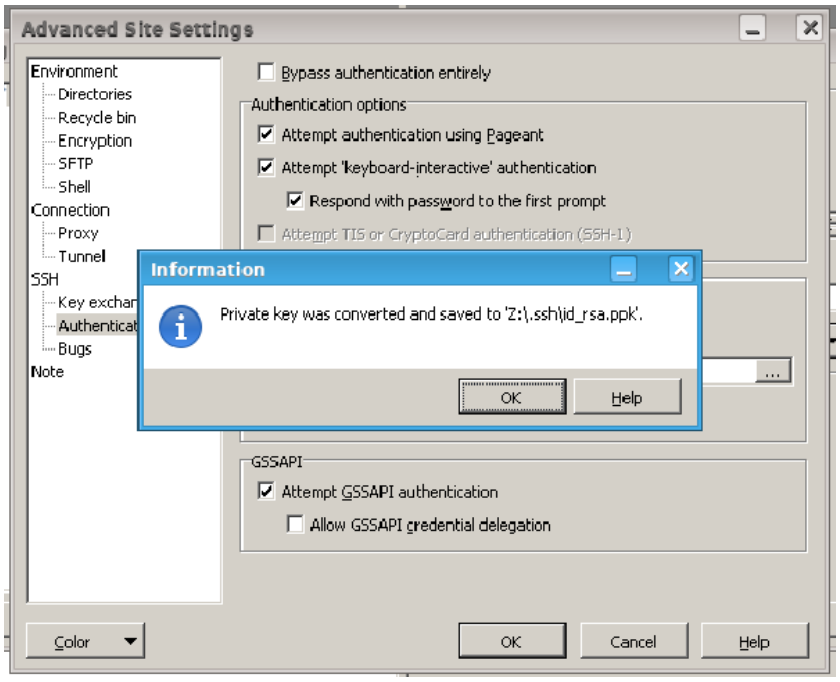
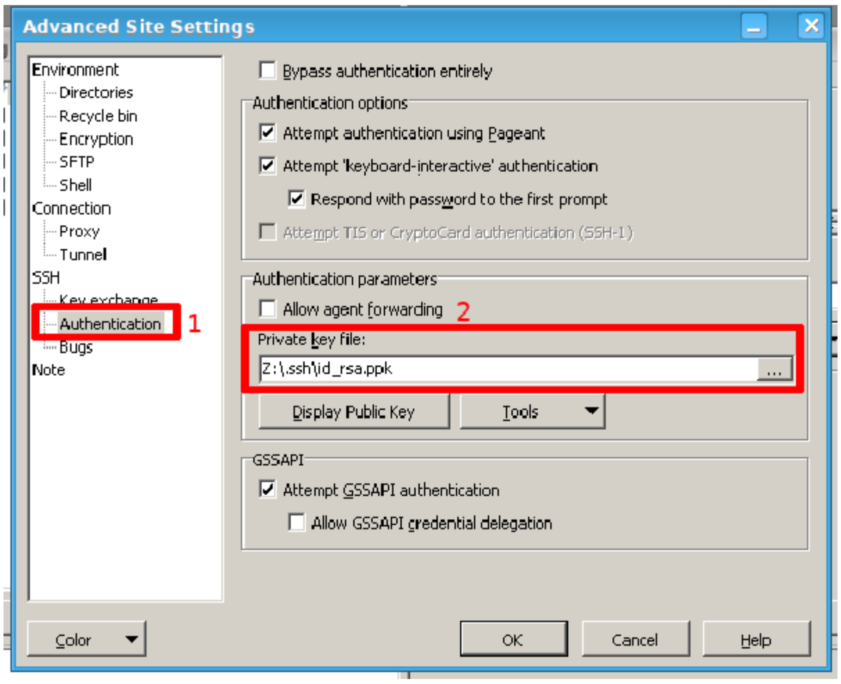


Figure 4.22: Save converted key file.


Advanced Site Settings
Environment
Directories
Recycle bin
Encryption
SFTP
Shell
Connection
Proxy
Tunnel
SSH
Key exchan
Authenticat
Bugs
Note
Bypass authentication entirely
Authentication options
Attempt authentication using Pageant
Attempt 'keyboard-interactive' authentication
Respond with password to the first prompt
Attempt TIS or CryptoCard authentication (SSH-1)
Information
OK
Help
GSSAPI
Attempt GSSAPI authentication
Allow GSSAPI credential delegation
Color
OK
Cancel
Help

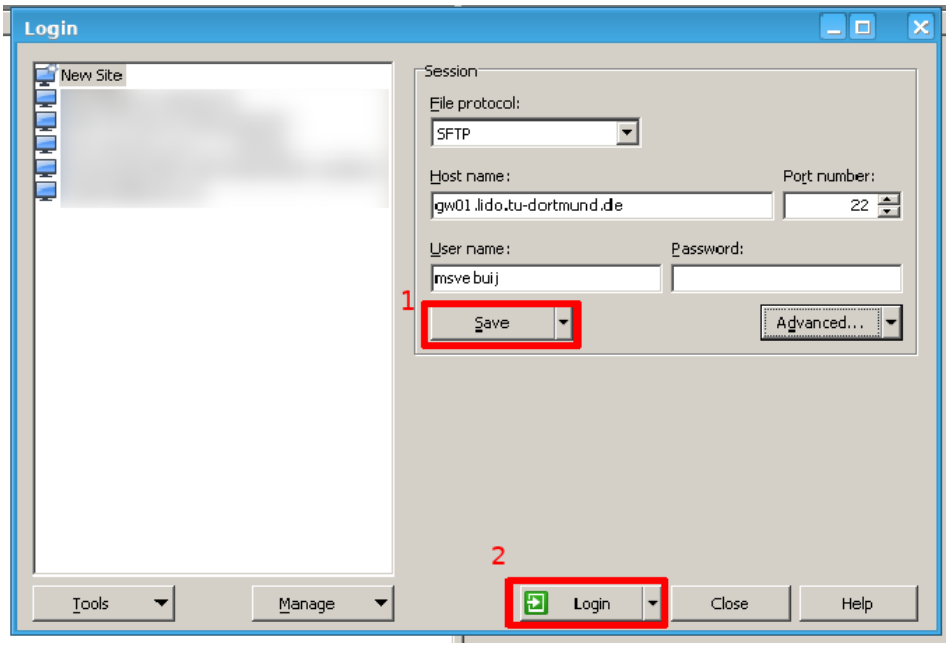
Private key was converted and saved to 'Z:\.ssh\id_rsa.ppk'.

Figure 4.23: Acknowledge success information.


Advanced Site Settings
Environment
Directories
Recycle bin
Encryption
SFTP
Shell
Connection
Proxy
Tunnel
SSH
Key exchange
Authentication
1
Bugs
Note
Bypass authentication entirely
Authentication options
Attempt authentication using Pageant
Attempt 'keyboard-interactive' authentication
Respond with password to the first prompt
Attempt TIS or CryptoCard authentication (SSH-1)
Authentication parameters
Allow agent forwarding
2
Display Public Key
Tools
GSSAPI
Attempt GSSAPI authentication
Allow GSSAPI credential delegation
Color
OK
Cancel
Help

Private key file: Z:\.ssh\id_rsa.ppk

Figure 4.24: Confirm key selection.



File protocol: SFTP
Host name: gw01.lido.tu-dortmund.de
Port number: 22
User name: msvebuij
Password:

Figure 4.25: Save connection setup and open connection.

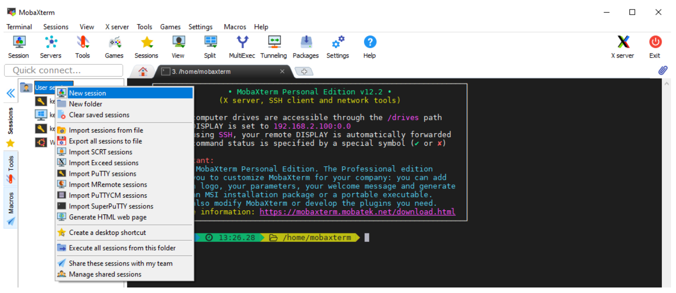
After you are connected, you can copy files around by drag-drop moving them from one window to the other or using menu entries and keyboard shortcuts, respectively. You can move (i.e. up-/download, then delete at source location) files to/from your local client, move them on the server side to different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

4.2.2.4 MobaXterm

The MobaXterm Client from Mobatek[23] offers a native shell environment for Windows with a build-in X11 server which allows for a login to the LiDO3 gateways and to start GUIs remotely.

The steps to connect to LiDO3 via MobaXterm are as follows:
After downloading and installing the MobaXterm client software on your own desktop, make sure that you are connected to the TU Dortmund University network (either by an active VPN connection or simply because you are physically at the university).
Next, you start MobaXterm and the first thing you want to do is to create a new user session.

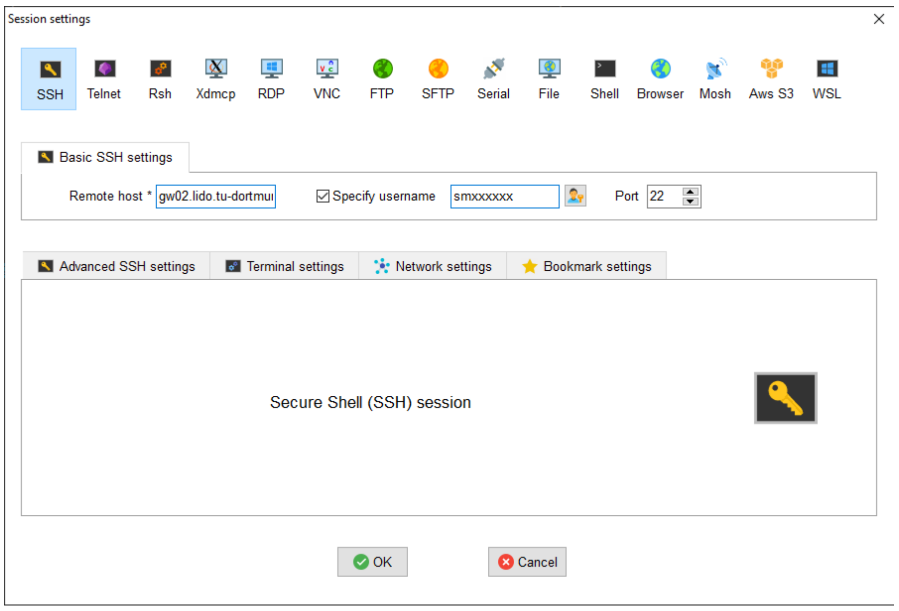
[23] https://mobaxterm.mobatek.net/



```
        • MobaXterm Personal Edition v12.2 •
      (X server, SSH client and network tools)

omputer drives are accessible through the /drives path
ISPLAY is set to 192.168.2.100:0.0
sing SSH, your remote DISPLAY is automatically forwarded
ommand status is specified by a special symbol (✓ or ✗)

ant:
MobaXterm Personal Edition. The Professional edition
ou to customize MobaXterm for your company: you can add
 logo, your parameters, your welcome message and generate
n MSI installation package or a portable executable.
lso modify MobaXterm or develop the plugins you need.
 information: https://mobaxterm.mobatek.net/download.html

 13:26.28  /home/mobaxterm
```

Figure 4.26: Mandatory: After starting MobaXterm right click on 'User Sessions' and select 'New session'.

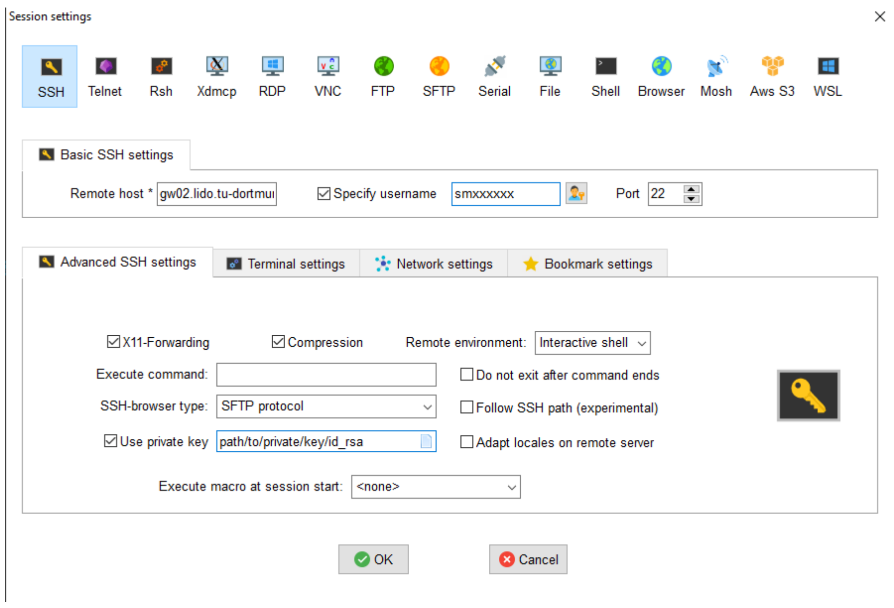
In the following screen you can select which type of connection you want for your session. To connect to LiDO3 you want to select 'SSH'. Now you want to declare your 'Remote host', i.e. gw02.lido.tu-dortmund.de to connect to gateway02 server, followed by your username, which is usually your university account (smxxxxxx or mxxxxxxx).



Remote host * gw02.lido.tu-dortmui ☑ Specify username smxxxxxx Port 22

Figure 4.27: Select SSH, fill in the LiDO3 gateway hostname and your login name.

As you need a SSH private key to connect to LiDO3, you can now select 'Advanced SSH settings' and check 'Use private key' and fill in the path to your SSH private key for which a public key is present on LiDO3. Make sure the 'X11-Forwarding' is active to use the build-in X11 server MobaXterm is shipped with.

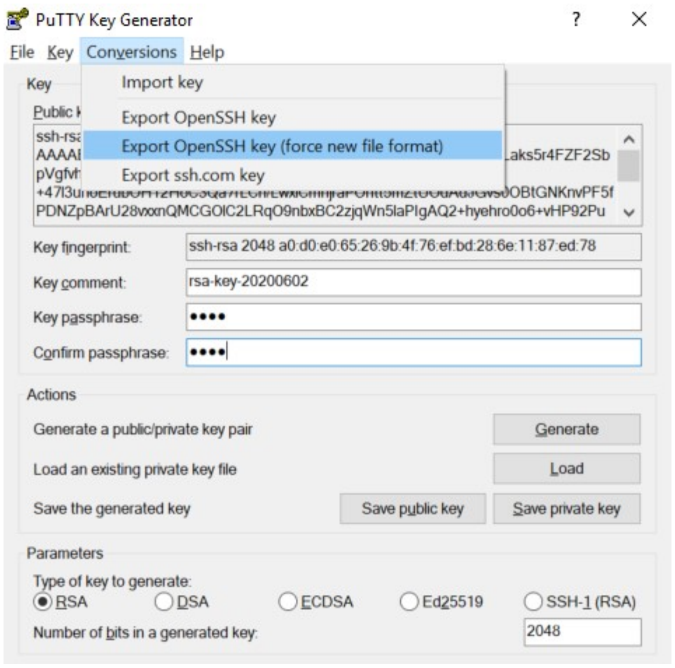


Remote host * gw02.lido.tu-dortmu
☑ Specify username smxxxxxx
Port 22

☑ X11-Forwarding
☑ Compression
Remote environment: Interactive shell
Execute command:
☐ Do not exit after command ends
SSH-browser type: SFTP protocol
☐ Follow SSH path (experimental)
☑ Use private key path/to/private/key/id_rsa
☐ Adapt locales on remote server
Execute macro at session start: <none>

Figure 4.28: Add the path to the SSH private key, check X11-Forwarding.

The provided key must be in the OpenSSH key file format. The propriatory key file format used by PuTTY (see 30), stored in files with `ppk` file extension, is not supported. If you get the error message "no authorization" or "invalid key file type", you may need to convert your SSH private key first to the OpenSSH key file format. This is done with the PuTTY tool `puttygen` or its direct clone, the MobaXterm SSH Key Generator. Load your existing SSH private key in PuTTY key file format (‘*.ppk‘) in `puttygen` and convert it via menu `Conversions`→`Export OpenSSH key` to a new file.



```
ssh-rsa
AAAA                                                  aks5r4FZF2Sb
pVgfv
+47I3u                                               0BtGNKnvPF5f
PDNZpBArU28vxnQMCGOlC2LRqO9nbxBC2zjqWn5laPIgAQ2+hyehro0o6+vHP92Pu
```

Key fingerprint: ssh-rsa 2048 a0:d0:e0:65:26:9b:4f:76:ef:bd:28:6e:11:87:ed:78

Key comment: rsa-key-20200602

Key passphrase: ••••

Confirm passphrase: ••••

Generate a public/private key pair

Load an existing private key file

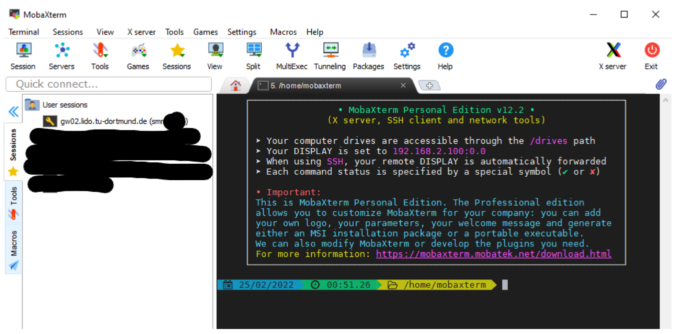
Save the generated key

Type of key to generate:
(● RSA) ○ DSA ○ ECDSA ○ Ed25519 ○ SSH-1 (RSA)

Number of bits in a generated key: 2048

Figure 4.29: Converting the private key from ppk to OpenSSH key format.

After you successfully configured your session, you can now select it from your left hand side menu and double click on it.

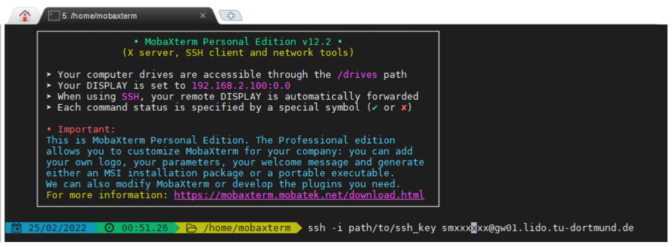

MobaXterm
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Quick connect...
5. /home/mobaxterm
Sessions
Tools
Macros

User sessions
gw02.lido.tu-dortmund.de (smm[redacted])
[redacted]

```
              • MobaXterm Personal Edition v12.2 •
            (X server, SSH client and network tools)

 ➤ Your computer drives are accessible through the /drives path
 ➤ Your DISPLAY is set to 192.168.2.100:0.0
 ➤ When using SSH, your remote DISPLAY is automatically forwarded
 ➤ Each command status is specified by a special symbol (✓ or ✗)

 • Important:
 This is MobaXterm Personal Edition. The Professional edition
 allows you to customize MobaXterm for your company: you can add
 your own logo, your parameters, your welcome message and generate
 either an MSI installation package or a portable executable.
 We can also modify MobaXterm or develop the plugins you need.
 For more information: https://mobaxterm.mobatek.net/download.html

 25/02/2022  00:51.26  /home/mobaxterm
```

Figure 4.30: Select your just created user session via double-clicking. A new prompt will open and you will be asked for your passphrase.

Alternatively, you can simply start a local session and enter your ssh command into the command line, as you would do if you were using a Unix OS.



```
             • MobaXterm Personal Edition v12.2 •
            (X server, SSH client and network tools)

 ➤ Your computer drives are accessible through the /drives path
 ➤ Your DISPLAY is set to 192.168.2.100:0.0
 ➤ When using SSH, your remote DISPLAY is automatically forwarded
 ➤ Each command status is specified by a special symbol (✓ or ✗)

 • Important:
 This is MobaXterm Personal Edition. The Professional edition
 allows you to customize MobaXterm for your company: you can add
 your own logo, your parameters, your welcome message and generate
 either an MSI installation package or a portable executable.
 We can also modify MobaXterm or develop the plugins you need.
 For more information: https://mobaxterm.mobatek.net/download.html

 25/02/2022  00:51.26  /home/mobaxterm  ssh -i path/to/ssh_key smxxxxxx@gw01.lido.tu-dortmund.de
```

Figure 4.31: Alternatively to creating a user session you can use MobaXterm like a common terminal you would be using on a Unix OS and enter the ssh command manually.

4.2.2.5 SSH Cryptonaut

The GUI client software Cryptonaut[24] is deprecated and discontinued, but still in use by some users. The latest beta version 7.0.0.1483 does, until now, support the current SSH protocol version and is thus still usable, despite dating back to 2017.

If you are not already using this software, please strongly consider using one of the other SSH clients, that are still maintained and receive security updates.

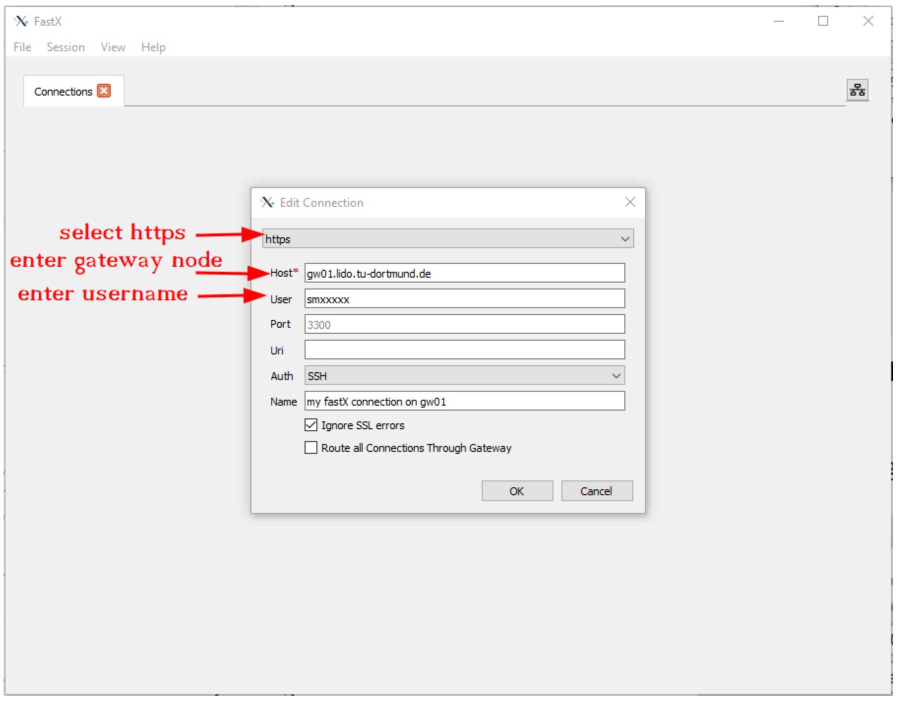
4.2.3 Starnet FastX

The FastX Client (available for Linux, Windows and MacOS) from Starnet[25] offers a remote graphical login to the LiDO3 gateways. It is also possible to connect via web-interface.

4.2.3.1 Using the FastX Client

The steps to connect to LiDO3 via FastX are as follows:
After downloading and installing the FastX client software on your own desktop, you need to make sure that you are inside the University network by VPN or simply by beeing at the university.
Next you start FastX and click the '+'-Button in the connections tab.

[24] https://web.archive.org/web/20171104034936/https://www.ssh.com/products/ssh-cryptonaut
[25] https://www.starnet.com/download/fastx-client



https
Host* gw01.lido.tu-dortmund.de
User smxxxxx
Port 3300
Uri
Auth SSH
Name my fastX connection on gw01
☑ Ignore SSL errors
☐ Route all Connections Through Gateway

Figure 4.32: Mandatory: Click the '+'-Button in order to enter a new connection. The host has to be one of our gateway nodes.

After clicking on *ok*, the connection you have just set up should now appear in the overview. Make sure that your key agent is active with your ssh-key.

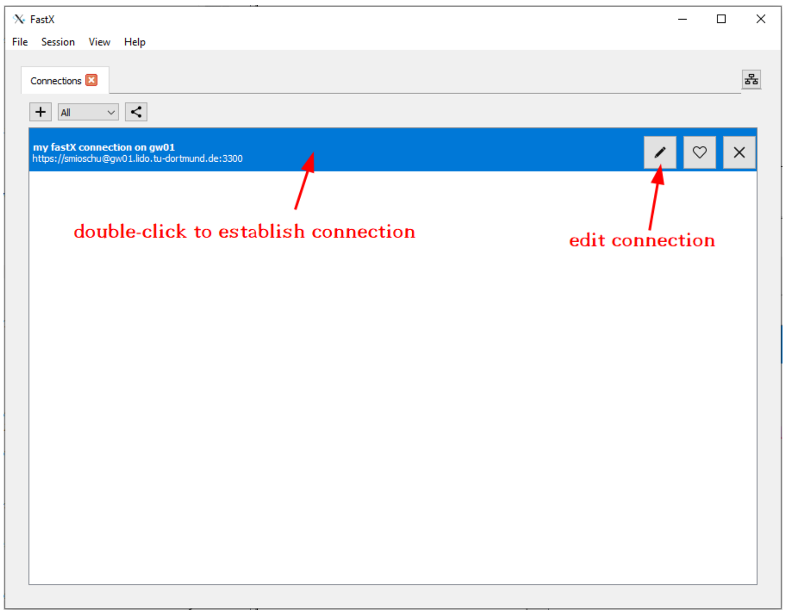
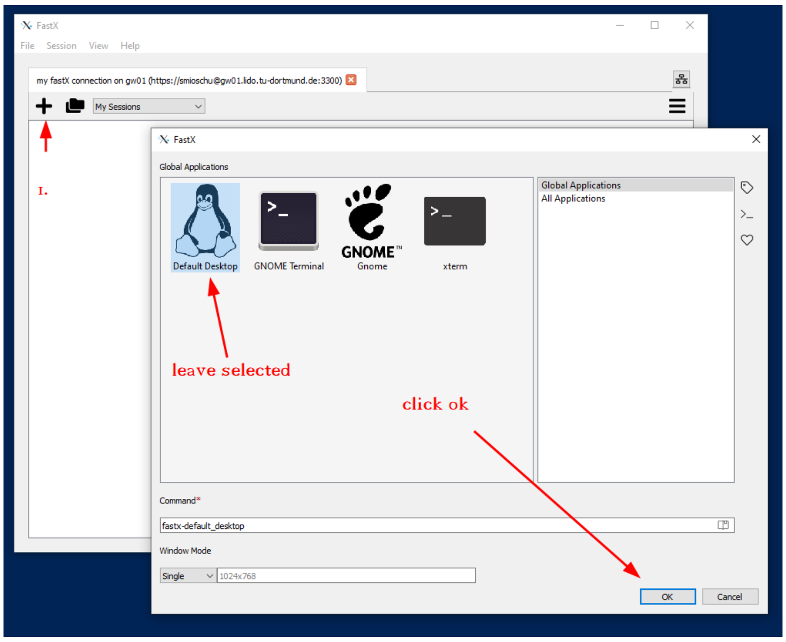

FastX
File Session View Help
Connections
All
my fastX connection on gw01
https://smioschu@gw01.lido.tu-dortmund.de:3300
double-click to establish connection
edit connection

Figure 4.33: Select the new connection-option and establish the connection.

In the connection overview, double-click on the line with the connection you want to establish. If the connection is active, a new window appears. Here you can select one of your persistent sessions. If the list is blank, you can start a new session pressing the '+'-Button. If there is already a session, then double-click this entry.



my fastX connection on gw01 (https://smioschu@gw01.lido.tu-dortmund.de:3300)

Command*: fastx-default_desktop

Window Mode: Single 1024x768

Figure 4.34: Begin of a new session.

After starting or enter an existing session, you get your window with the remote session

The terminal and other applications can be reached via the *Applications* menu in the top left corner of your screen.

In order to get a context menu to control the FastX session, you have to move the mouse in the upper middle area of the window. There you can pause your session (the window closes but the session remains active - you can reconnect later from the same or different computer with FastX).

If you want to terminate the session, you have to click on a small 'x' on the session icon in the session overview. You will be asked to terminate the session.

Another feature of FastX is cooperative working, i.e. you can specify other users who may participate in your session. Furthermore you can also specify whether other users must first wait to be admitted or whether they are allowed to take control of the session. This coop feature will described in Section 4.2.3.3.

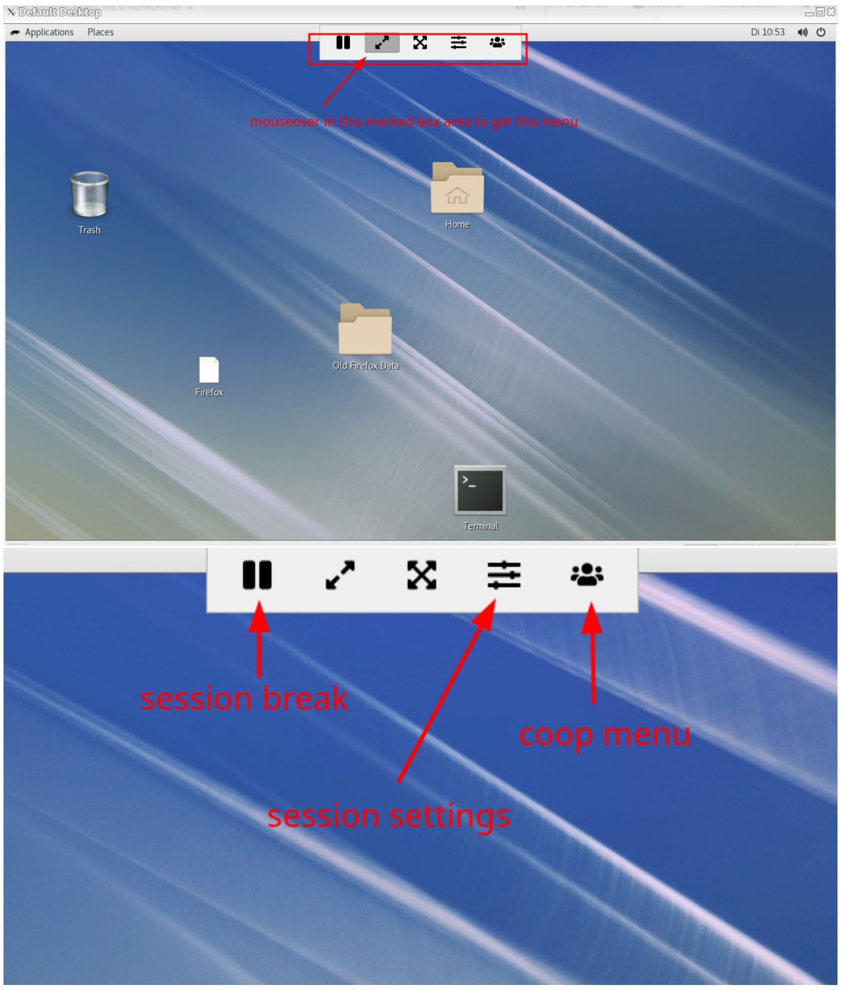


Figure 4.35: To get the context menu of your session, move the mouse to the area, marked with a red box. The context menu appears.

4.2.3.2 Using FastX without Client in a Browser

You can connect to LiDO3 with FastX with your favorite browser. The disadvantage of using your browser in contrast using a client is, that your browser does not support any key-agent and you have to enter the location and the passphrase of your ssh-key.

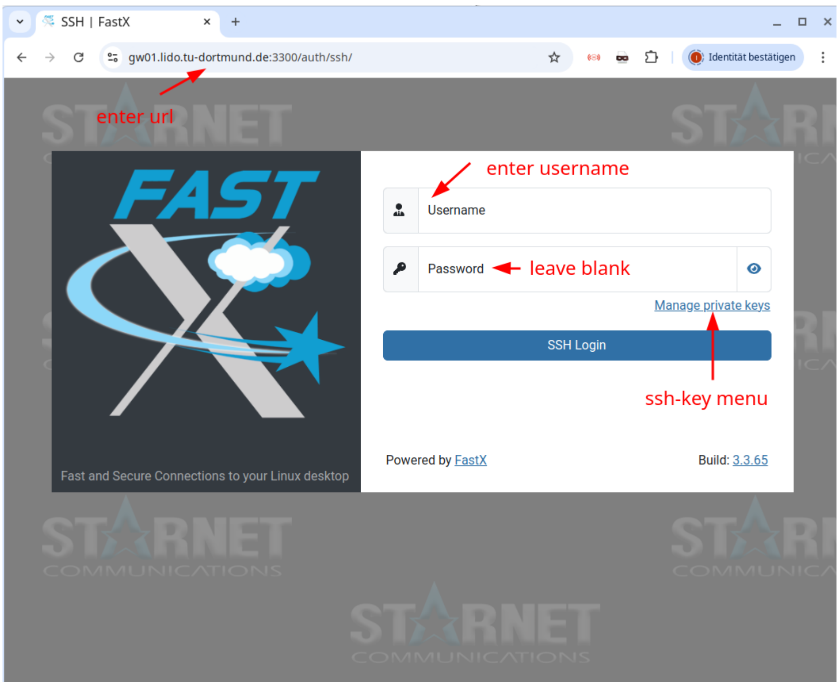


Figure 4.36: Enter the url gw01.lido.tu-dortmund.de:3300 to get this screen. The prefix gw02 instead of gw01 is possible.

In this menu you have to provide your ssh-key. Click 'manage private keys'. The next menu appears.

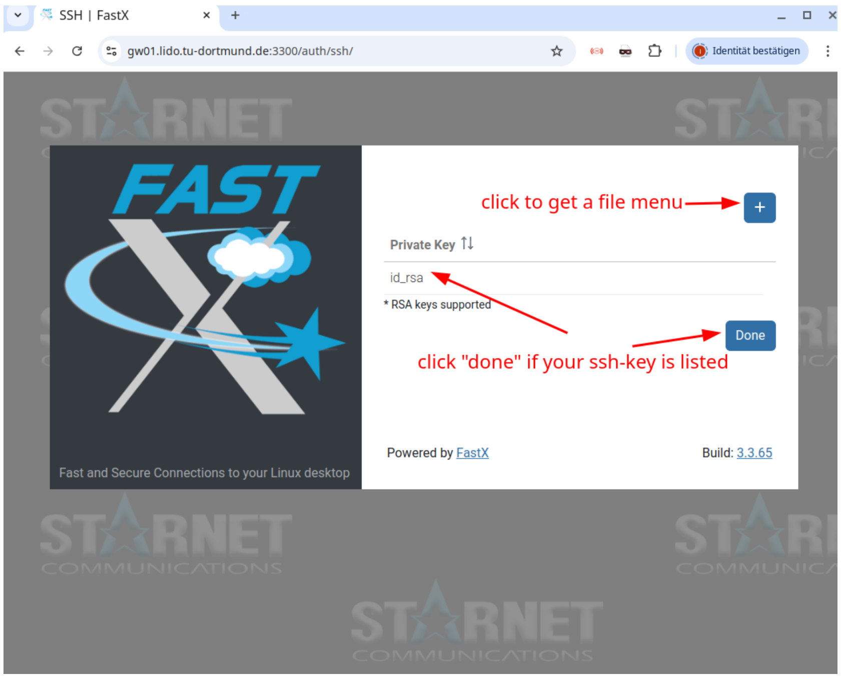


Figure 4.37: Managing your private key.

To provide a new private key, you have to click the '+'-Button. A file menu appears where you can select your key. Your selected key should now appear in the list of private keys. After that you can close this menu by clicking 'Done'.

Then you return to the first menu, where you can enter your username. The password field remains blank. By clicking 'SSH Login', a prompt to enter the passphrase appears.

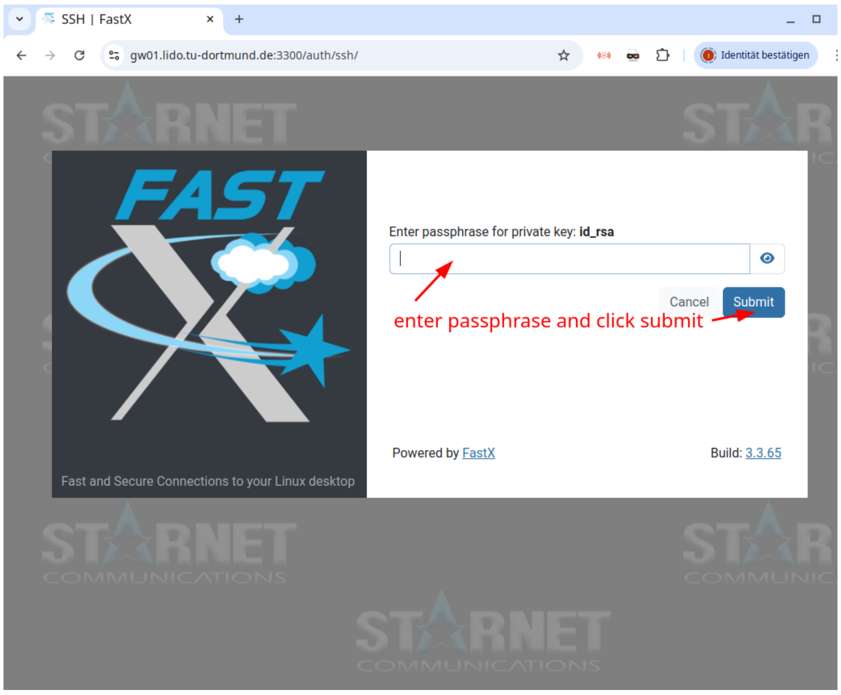

SSH | FastX
gw01.lido.tu-dortmund.de:3300/auth/ssh/
Identität bestätigen
FAST X
Fast and Secure Connections to your Linux desktop
Enter passphrase for private key: id_rsa
Cancel
Submit
enter passphrase and click submit
Powered by FastX
Build: 3.3.65

Figure 4.38: Enter passphrase of your private key.

After that, you can start a new session or you can connect to a paused session.

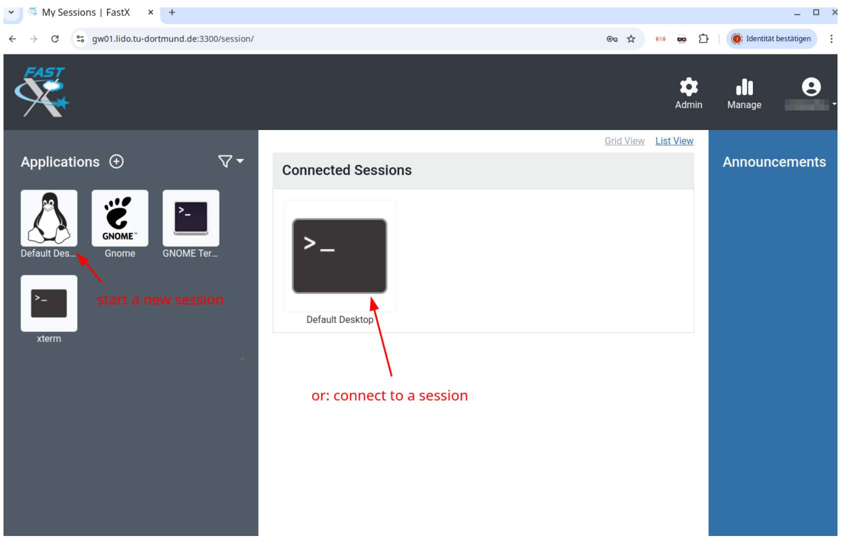


Figure 4.39: Select a session.

After a session is opened or reopened, you get a new browser tab with your connected session.

There is a session control menu just like in the client version. Move the mouse to the upper center area of the tab to get this menu. (Compare Fig. 4.35). To leave the session without closing, simply close the tab. To enter additional user, which should participate, select the coop-menu. To terminate the session, you have to select the first tab, where your session is listed. There you can click on a small 'x' on the session icon and you will be asked to terminate the session.

4.2.3.3 Cooperative working with FastX

If you want to show your work to a collegue or you have a problem and you want to show it to an admin, you can tell FastX which users are allowed to view your session. Just click on the coop button of your session menu.

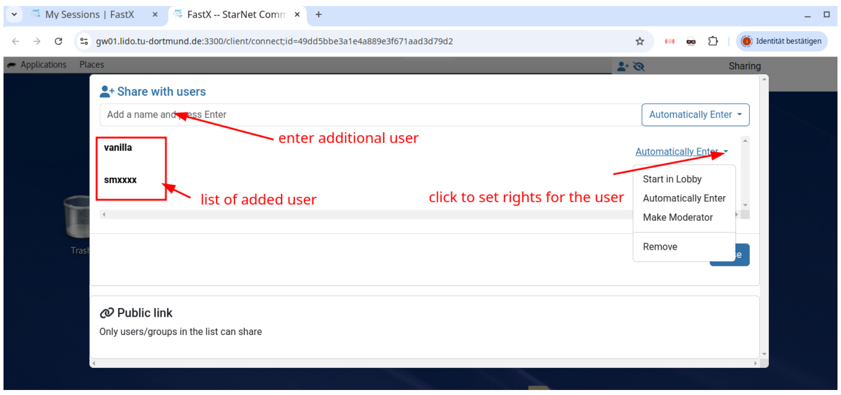


Figure 4.40: Select a session.

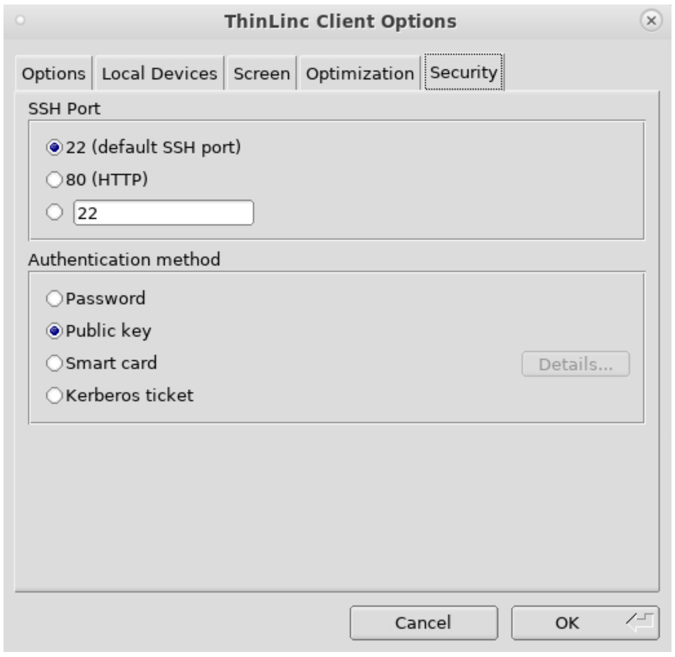
The added user can select this session in his own session menu in the ‘Session shared with me’ area. In the client, the added user has to select ‘Session shared with me’ instead of 'my sessions' in order to view and select this session.

4.2.4 Cendio ThinLinc

The ThinLinc Client (available for Windows and Linux) from Cendio[26] offers a remote graphical login to the LiDO3 gateways.

[26] https://www.cendio.com

The steps to connect to LiDO3 via Thinlinc are as follows:
After downloading and installing the ThinLinc client software on your own desktop, you need to make sure that you are inside the University network by VPN or simply by beeing at the university.
Next you start ThinLinc and change the authentification method in the options menu.

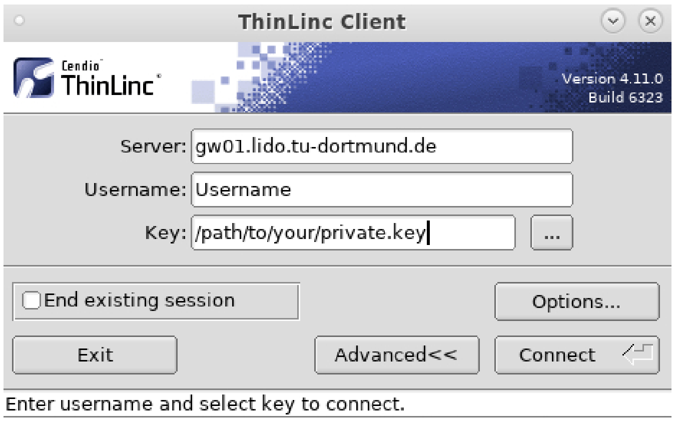


SSH Port
- [✓ 22 (default SSH port)]
- 80 (HTTP)
- 22

Authentication method
- Password
- [✓ Public key]
- Smart card
- Kerberos ticket

Figure 4.41: Mandatory: Open the Options menu, switch to the Security tab and change the Authentication method from Password to Public key.

Afterwards you can type in the login credentials and click on *Connect*.

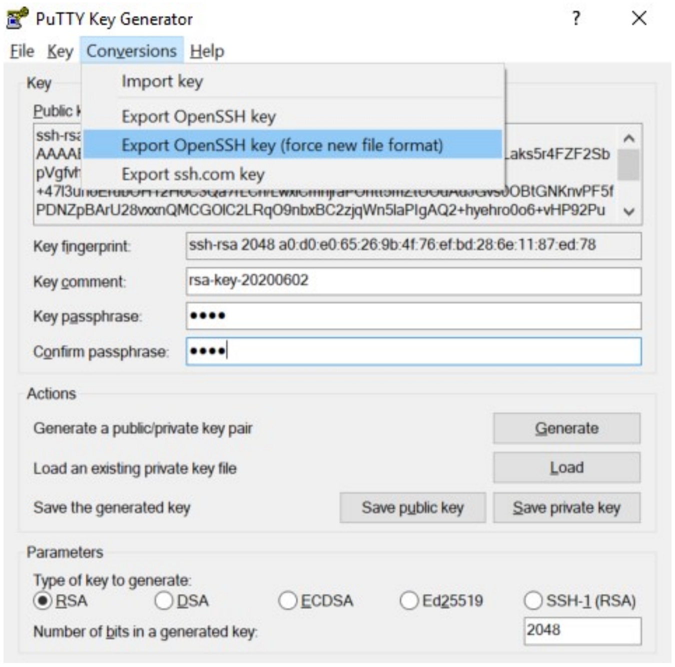


Server: gw01.lido.tu-dortmund.de
Username: Username
Key: /path/to/your/private.key

Enter username and select key to connect.

Figure 4.42: Fill in the LiDO3 gateway hostname, your login credentials and the path to your private key.

The provided key must be in the OpenSSH key file format. The ppk PuTTY file format is not supported. If you get the error message "no authorization" or "invalid key file type", you may need to convert your private key first. This is done with the PuTTY tool `puttygen`. Load your existing private SSH key in PuTTY key format (‘*.ppk‘) in `puttygen` and convert it via menu `Conversions`→`Export OpenSSH key` to a new file.



Key fingerprint: ssh-rsa 2048 a0:d0:e0:65:26:9b:4f:76:ef:bd:28:6e:11:87:ed:78
Key comment: rsa-key-20200602
Key passphrase: ••••
Confirm passphrase: ••••
Generate a public/private key pair
Load an existing private key file
Save the generated key
Type of key to generate: [✓ RSA]
Number of bits in a generated key: 2048

Figure 4.43: Converting the private key from ppk to OpenSSH key format.

After your sucessfull login, you need to accept the probably uncached server key mentioned before and type in your ssh key password.
Click *forward* in the welcome screen and select *Gnome Desktop Classic* in the GUI selection screen. Afterwards confirm the selection by clicking *OK*

The Terminal and other applications can be reached via the *Applications* menu in the top left corner of your screen.

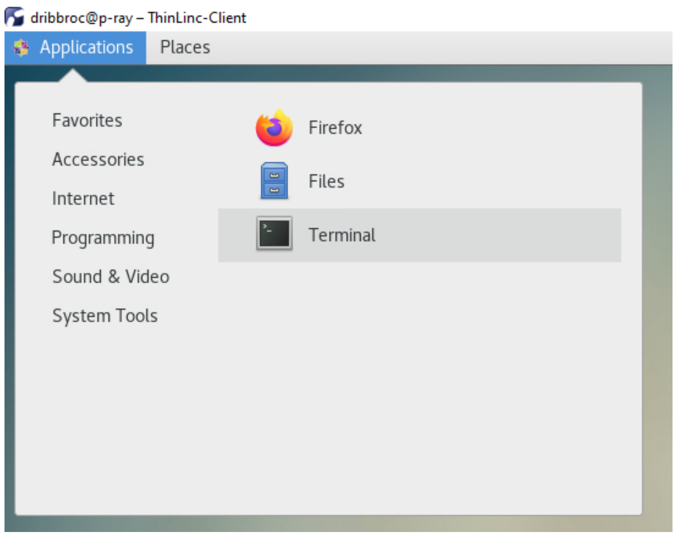


Figure 4.44: The Terminal and other applications can be reached via the *Applications* menu in the top left corner of your screen.

To logout correctly, it is not sufficient to simply close the ThinLinc windows but you need to click on the power-switch icon in the top right corner, on your Username and on *Log Out*.

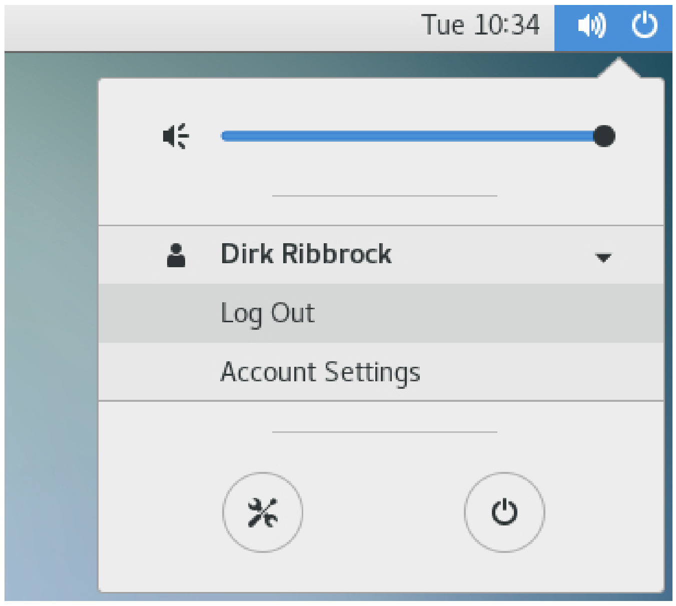


Figure 4.45: To log out correctly, it is not sufficient to simply close the ThinLinc windows but you need to click on the power-switch icon in the top right corner, on your Username and on *Log Out*.

Inside a ThinLinc windows, a menu can be opened with the key `F8`. This menu allows to switch between fullscreen and windowed mode and some other things.

4.2.5 Logins to compute nodes and inter-node connections

Once your account application has been approved and your account created, you can log in to the gateway servers. In general, you can not log in to the compute nodes. But you can log in to any compute node if and if only you have a Slurm job running there.[27] You can log in from a gateway server to a compute node and from a compute node to another compute node. Once your Slurm job finishes, your interactive SSH session there will end, too. Note that your SSH session on a compute node will use the very same compute cores that your Slurm job got assigned. So, CPU intensive tasks performed in your SSH session will slow down your application and mess with any time measurements you might be running as part of your Slurm job.

The first case, SSH from gateway to a compute node, is typically useful to monitor a simulation and check whether it behaves in the way you expect it to. For example, you can inspect on a compute node memory usage of your program or whether more threads are being used by your simulation than the number of compute cores asked for. The resulting significant amount of CPU time spent in kernel space (due to a high number of context switches) can significantly reduce the CPU efficiency of your simulation.

The second case, inter-node connections, are required by some applications that span multiple compute nodes. They use SSH for the initial communication setup during program startup. The application started on the first assigned compute node creates SSH connections to the remaining assigned compute nodes. Some Ansys applications operate this way.

In order to log in from a gateway to a compute node or from a compute node to another, public key authentication is required. If you are asked for a password (not the passphrase possibly securing your private key), you most probably did not provide a private key or the wrong private key. In this case, SSH automatically skipped forward to the next authentification method, i.e. password authentification. But on LiDO3, password authentification is disabled and thus this login method will fail, regardless of the password you provide.

[27] Check the column `NODELIST` in the output of the command `squeue --user=$USER ↪ --state=running` for the names of the compute nodes assigned to your currently running Slurm jobs.

There are two ways to enable public key authentication on the compute nodes. Either you configure the SSH client you use to log in to the gateway servers to enable a feature called "Agent Forwarding" (with OpenSSH 7.x this can be done by adding the command line option `-o ForwardAgent=yes`). Or you can generate a SSH key pair consisting of a private and a public key for **use on LiDO3 only**. I.e. a key pair you do not use anywhere else. (Re-using the key pair by configuring a remote computer such that the private key stored on LiDO3 allows to log in from LiDO3 to said remote computer entails the risk of compromising the remote computer if ever LiDO3 would get compromised.)

In order to generate an additional SSH key pair that allows password-less SSH logins between LiDO3 gateway servers and compute nodes or between compute nodes, proceed as follows:[28]

Step 1)

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

```
$ 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. While it is in general strongly advised to protect a SSH private key with a passphrase, please do not set a passphrase for this SSH key pair for use on LiDO3 only! Why? Software like Ansys Mechanical APDL, Ansys Fluent or CFX uses SSH internally during startup and they will not prompt you for the passphrase. As a result, their initialisation of the communication network via SSH will fail and these simulation packages will abort. So, for them a passphrase-protected private key imposes a problem. So, omit applying a passphrase for the SSH private key by just pressing the enter key when prompted for the passphrase.[29] The command will generate, besides an output along the following lines

[28]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.

[29]Why is there no security concern when omitting the passphrase for the SSH key pair used on LiDO3 internally only? Someone that would somehow gain access to your LiDO3 account can already log in to compute nodes using Agent Forwarding provided that a Slurm job of yours is running there and has no advantage of using the not passphrase-protected SSH private key stored on LiDO3. Someone with root privileges can log in to the compute nodes anyway such that this person would not benefit from being able to access and use your SSH private key stored on LiDO3. But both statements only hold for as long as the SSH key pair is not used anywhere else. It is, hence, strongly advised to passphrase-protect SSH private keys that you use to log in to remote computers, regardless whether they are stored on your local computer or on LiDO3.

```
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.

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

```
$ mkdir ~/.ssh
$ 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 you will get the error message

```
Bad owner or permissions on /home/[...]/.ssh/config
```

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 - optional)

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.

4.2.6 Troubleshooting

4.2.6.1 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)"
```

4.2.6.2 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.

4.2.6.3 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[30] before, you can find more than one tutorial[31] 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[32]
- emacs[33]
- gedit[34]
- nedit[35]
- nano[36]
- pico[37]

[30] https://en.wikipedia.org/wiki/Bash_%28Unix_shell%29
[31] http://tldp.org/LDP/Bash-Beginners-Guide/html/
[32] https://en.wikipedia.org/wiki/Vi
[33] https://en.wikipedia.org/wiki/Emacs
[34] https://en.wikipedia.org/wiki/Gedit
[35] https://en.wikipedia.org/wiki/NEdit
[36] https://en.wikipedia.org/wiki/GNU_nano
[37] https://en.wikipedia.org/wiki/Pico_(text_editor)

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.

See section *4.9.4 Can I have Visual Studio Code on LiDO3?* for the reasons you can't.

Just keep in mind that the newline[38] 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
  ```

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

[38]https://en.wikipedia.org/wiki/Newline

`/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 206 for some examples of already existing software.

- `/work` has different characteristics: it has a default quota of 2 TiB[39] 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.

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

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

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[40].

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 32 GiB 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`.
- 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`

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

- 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
```

4.3.2.3.2 du output and the parallel file system BeeGFS

`du` (abbreviated from *d*isk *u*sage) is a standard Unix program used to estimate file space usage, i.e. space used under a particular directory or files on a file system. `du`, however, does not work well on the parallel file system BeeGFS: `du` tends to report too low values for the files stored under `$WORK`. In particular for those subtrees in `$WORK` that have been created months or years ago and not been accessed for a long time. The output `du` reports may change if you were to copy the subdirectory to gauge to a different file system or a remote location prior and run the very same `du` command again.

Instead of `du -hs`, `du -hs --apparent-size` or similar, use `lido3-quota` or `beegfs-ctl --getquota --uid <your-lido3-user-id>` to determine how much file space you are using in `$WORK`.

For a more fine-grained analysis, have `find` and `awk` calculate how much space is being used by directory subtrees. Example:

```
find /work/my/dir1 /work/my/dir2 -print0 | xargs -0 stat
    ↪ --format=%s | awk '{ s+=$1; } END { print s / 1024**3 "
    ↪ GiB"; }'
```

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.

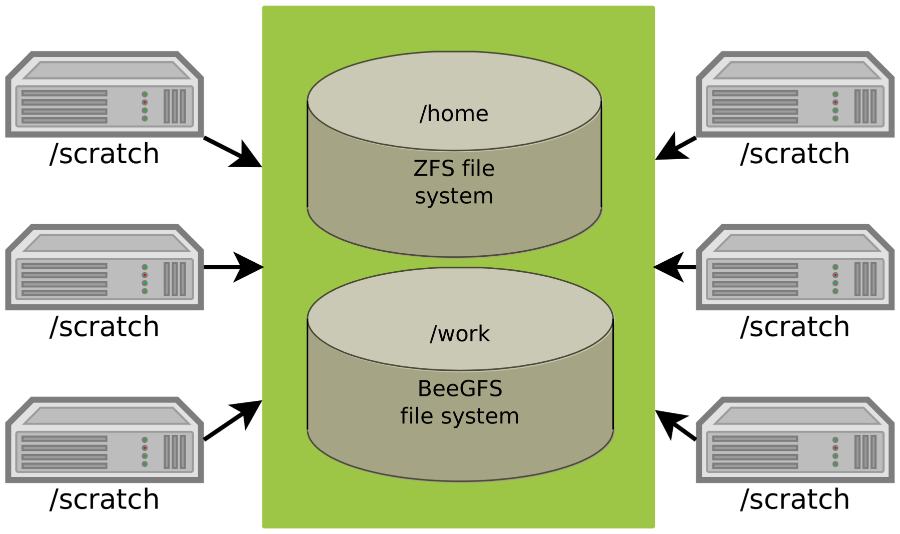


Figure 4.46: */home* and */work* can be accessed from any node, */scratch* is only a local file system.

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

In case your Slurm jobs do not allocate complete compute nodes (e.g. only a few cores or even only a single one), the Slurm scheduler may start several of your jobs on the same compute node. They might get started on the same compute node simultaneously or the run times overlap. If their input data is (partially) identical and of a considerable size, it might not be possible to copy them to a job-individual subdirectory in `/scratch`. In this case, you will face the challenge of ensuring that only one of your jobs copies the files. If several Slurm jobs would copy the same files to the same destination, this could result in data corruption in the copied files. The following code snippet uses a lock file mechanism (in compute science often called semaphore[41]) to ensure just that: only one of the potentially many Slurm jobs started on a single compute node does the actual file transfer, all others wait for the transfer to finish:

```
#!/bin/bash
#SBATCH --time=[...]
#SBATCH --nodes=[...]
#SBATCH --ntasks-per-node=[...] --cpus-per-task=[...]
#SBATCH --partition=[...]
#SBATCH [...]

##################################################################
# Cache input files on local file system. (To avoid straining
# the object storage servers of the parallel file system BeeGFS
# with repeated random access patterns for these input files and
# to maximize file reading speed while your application
# runs. (You may even consider creating a temporary ramdisk for
# the duration of your Slurm job in case your application does
# heavy I/O on small files.)
#
# Use a semaphore to allow only one job script at a time to
# do the actual file transfer (avoiding data corruption in the
# destination directory from otherwise potential concurrent
# file transfers).
LOCK_FILE=/scratch/${USER}/input_file_transfer_lock
SOURCE_DIR=/work/${USER}/path/to/your/input/data
DEST_DIR=/scratch/$USER/common_input_data
if [ ! -d "$( dirname ${LOCK_FILE} )" ]; then
    mkdir -p "$( dirname ${LOCK_FILE} )"
fi
trap "rm ${LOCK_FILE} 2>/dev/null" INT TERM EXIT
# Try (repeatedly if necessary) to obtain an exclusive lock
exec {NAMED_FILE_DESCRIPTOR}> ${LOCK_FILE}
flock -x -n ${NAMED_FILE_DESCRIPTOR};
while test $? -ne 0; do
    printf "%s: %s" "$( date )" "Waiting for lock on ${LOCK_FILE}
    ↪ to get released."
    for i in $( seq 10 ); do echo -n "."; sleep 1; done; echo
    flock -x -n ${NAMED_FILE_DESCRIPTOR};
done
# Exclusive lock obtained
if [ ! -d "${DEST_DIR}" -o $( find ${DEST_DIR}/ 2>/dev/null | wc
    ↪ -l ) -ne $( find ${SOURCE_DIR} | wc -l ) ]; then
```

[41] https://en.wikipedia.org/wiki/Semaphore_(programming)

```
    printf "%s: %s\n" "$( date )" "Starting to copy files from
    ↪ ${SOURCE_DIR} to ${DEST_DIR}."
    # input files are not present in local file system yet.
    mkdir -p ${DEST_DIR}
    cp -af ${SOURCE_DIR} $( dirname ${DEST_DIR} )
    # Update time stamp of the just copied files to prevent that
    # system cleanup scripts on LiDO3 automatically remove
    # them. (They will remove any file in /scratch not modified
    # within the last two days unless the owner of these files
    # still runs a Slurm job on this particular compute node.)
    find ${DEST_DIR}/ -print0 | xargs -0 touch
    printf "%s: %s\n" "$( date )" "Copying finished."
fi
# Release lock
flock -u -n ${NAMED_FILE_DESCRIPTOR};

###### End of input file transfer to /scratch
##################################################################
```

Listing 4.3: File transfer to `/scratch` with lock file safeguard mechanism

⚠ The `/scratch` partions on all nodes are regularly purged and all files that have not been modified in the last two days after your job finished are deleted.

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[42] clients for transfering the files back and forth from your Windows machine, e.g. FileZilla[43] and WinSCP[44]. For both programs, the respective websites explain how to set up SSH public key authentication[45],[46].

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[47] 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)

[42] https://en.wikipedia.org/wiki/Graphical_user_interface
[43] https://filezilla-project.org/
[44] http://winscp.net/
[45] https://wiki.filezilla-project.org/Howto
[46] https://winscp.net/eng/docs/guide_public_key
[47] https://en.wikipedia.org/wiki/Setuid

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
```

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[48] 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
```

Setting the setgid bit via

```
$ chmod g+s /home/sma/shared-work
```

[48] https://en.wikipedia.org/wiki/File-system_permissions

does not require root privileges, it can be added by any user. As stated on the previous page, the setgid bit triggers that all newly created files and directories will be owned by the very same unix group `/home/sma/shared-work` belonged to at the time those extra files and directories got created. Once the setgid bit has been set additionally, an *s* will be listed instead of an *x* for the group x-bit:

```
$ ls -lad /home/sma/shared-work
drwxr-s--- 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* whould 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[49] and its current values.

```
$ 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.

[49] https://en.wikipedia.org/wiki/Umask

4.3.5 Software modules, environment modules

The software and tools needed for development and job execution are organized as *environment modules*, commonly abbreviated to *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[50] 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 ---
dot  module-git  module-info modules  null  use.own

---- /cluster/sfw/modulefiles ---
abaqus/2022-hotfix2  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>`:

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

```
$ 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[51].

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:

```
# 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, environment modules*).

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

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 :[52] 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
```

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.

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

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 (see section 4.8) for additional support.

4.3.6.1 configure-make-install

Most classic Unix/Linux software packages use GNU Autotools[53] (`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

```
./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.

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

For cmake-based[54] 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, venv, virtualenv & conda

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

```
python3 -m pip install --user <package_name>
pip3 install --user <package_name>
```

Note that this will install to `${HOME}/.local/lib/python<pythonversion>`. On LiDO3, a user's home directory gets set up such that `${HOME}/.local` is in fact a symbolic link to `${WORK}/.local`. So, the Python packages seem to have been installed to a user's home directory, but in fact they are stored in the parallel file system. This has the benefit that a user can install additional Python packages as part of a Slurm job[55] Additionally, the performance accessing these files from several dozen or hundred concurrently running Slurm jobs is a lot better than if they were stored in a user's `${HOME}` directory (which is served by two NFS servers instead of a high-performant parallel file system). There is a drawback, however: Python packages getting in fact installed to the parallel file system means that they will not be backed up automatically.

[54] https://cmake.org/

[55] Remember that only `${WORK}` can be written to from the compute nodes while `${HOME}` can only be written to from the gateway servers; see section 4.3.2.1.

You may want to organize your Python modules required for different projects in so-called virtual environments. Read up on venv[56] and virtualenv[57]. These external sites describe their use much better than we ever could. There are a number of video tutorials on why and how to use virtual environments in Python available on sites like, e.g., YouTube, too.

When installing packages, `pip` installs dependencies in a recursive, serial loop. No effort is made to ensure that the dependencies of all packages are fulfilled simultaneously. This can lead to environments that are broken in subtle ways, if packages installed earlier in the order have incompatible dependency versions relative to packages installed later in the order. `venv` and `virtualenv` serve to a good end to avoid this. But it is still up to you as a user to ensure that your virtual environment does not get messed up by updating or installing a Python package.

In contrast, `conda` uses a satisfiability (SAT) solver to verify that all requirements of all packages installed in an environment are met. This check can take extra time but helps prevent the creation of broken environments. As long as package metadata about dependencies is correct, conda will predictably produce working environments.[58] In order to use `conda` with Python versions newer than 3.7, make sure to load on LiDO3 an environment modulefile (see 4.3.5) matching the pattern `python/3*conda*`. Install a package like, e.g., `requests` by replacing `<package_name>` with `requests` and name your new conda environment in a distinguishable manner:

```
conda create --no-default-packages --name <customname>
    ↪ <package_name>
```

`conda` installs to `${HOME}/.conda` by default. On LiDO3, similiar to `pip`, a user's home directory gets set up such that `${HOME}/.conda` is in fact a symbolic link to `${WORK}/.conda`. This has the same advantages and the disadvantage mentioned in the previous paragraph for `pip`.

[56] https://docs.python.org/3/library/venv.html
[57] https://virtualenv.pypa.io/en/latest/
[58] For more details that `pip` and `conda` have in common and where they differ, see the official documentation[59].

4.4 Resource management

LiDO3 uses the *Slurm Workload Manager*[60] 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. It is not possible to start processes directly on individual compute nodes, except as part of interactive Slurm jobs. Long-running, cpu-hungry processes running directly on the gateway servers are killed without further notification.

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 195).
tasks	The actual runing code.

[60] https://slurm.schedmd.com/

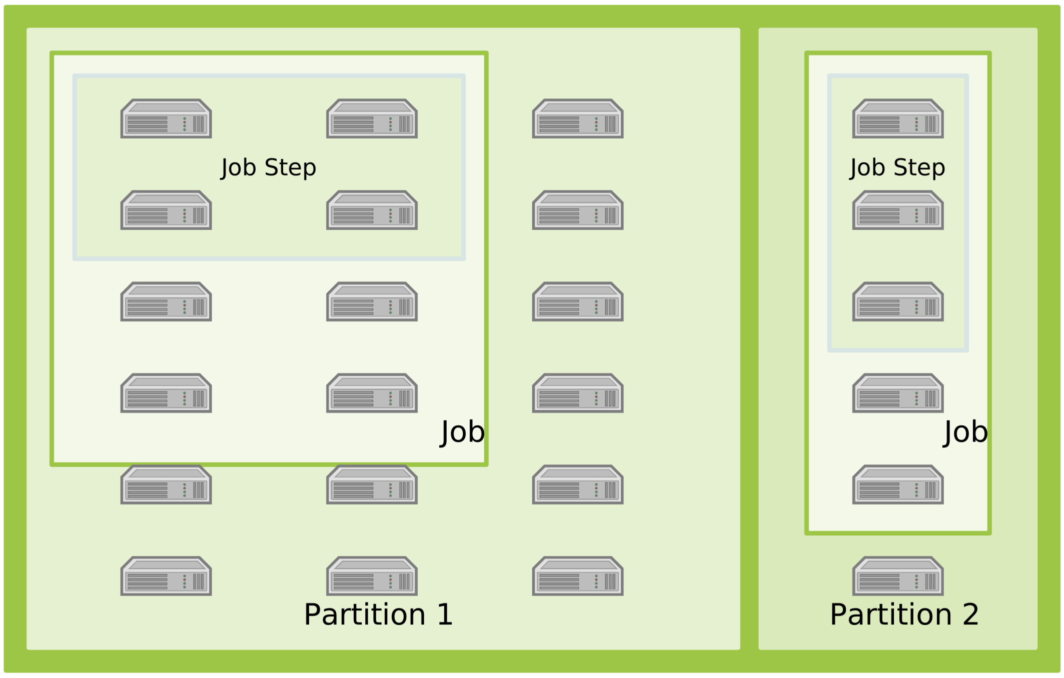


Figure 4.47: Slurm entities.

"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 Partitions

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

4.4.1.1 Standard partitions

The vast majority of compute nodes is represented via the standard public partitions:

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

These are used for all kinds of jobs and are mainly separated by the maximum walltime allowed per job.

4.4.1.2 Faculty partitions

These partitions represent hardware that was later added to LiDO3 and financed from individual faculties or institutes. The access to this hardware is generally available to everyone, but restricted to a maximum job walltime of two hours via the `ext_*_norm` partitions. Further usage is restricted to members of the corresponding faculties or institutes via the `ext_*_prio` partitions.

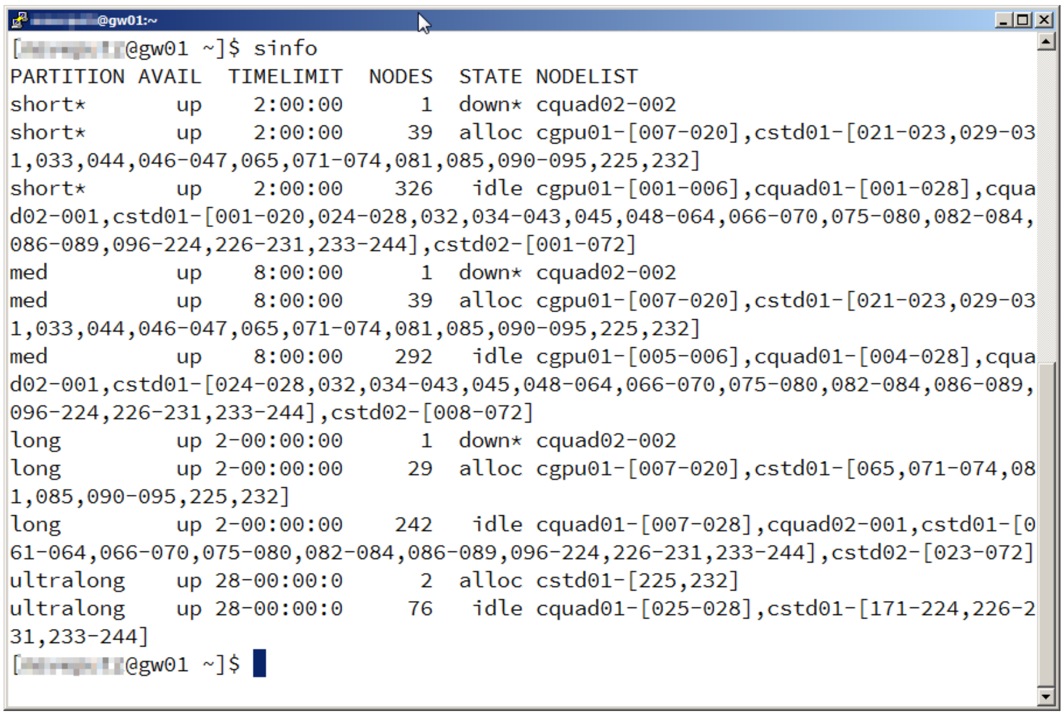
Table 4.3: Partitions with faculty hardware

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	`ext_iom_prio` group members only
ext_iom_norm	02:00:00	—
ext_trr188	28-00:00:0	`ext_tr188` group members only
ext_vwl_prio	28-00:00:0	`ext_vwl_prio` group members only
ext_vwl_norm	02:00:0	—
ext_math_prio	28-00:00:0	`ext_math_prio` group members only
ext_math_norm	02:00:0	—

ext_chem_prio	28-00:00:0	`ext_chem_prio` group members only
ext_chem_norm	02:00:0	—
ext_biochem_prio	28-00:00:0	`ext_biochem_prio` group members only
ext_biochem_norm	02:00:0	—
ext_InfCI	28-00:00:0	`ext_infci` group members only
ext_chem2_prio	28-00:00:0	`ext_chem2_prio` group members only
ext_chem2_norm	02:00:0	—
ext_ace_prio	8-00:00:0	`ext_ace_prio` group members only

4.4.1.3 slurm command `sinfo`

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 4.48: Gathering information about the partitions.

4.4.2 Slurm job submission

A job is in principal the specification of the 'what' and 'where' shall be executed on the cluster. 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 *Partitions*. 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. "*

— 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.

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

4.4.2.1 *Slurm* with serial, threaded or MPI-based programs

To allocate and use the resources of the cluster (especially the compute nodes and their processors and memory), one has to declare the desired amount in advance. Within a *Slurm* job script or via corresponding command line options to the commands `sbatch`, `salloc` or `srun`. As soon as sufficient resources are available, *Slurm* will then start your job, assigning it the requested resources, but *Slurm* will prevent the use of any non-allocated resource. You can not, for example, use any processor's core that you have not asked for when submitting your *Slurm* job.

The first step when planning your resource usage is to evaluate how many ranks or tasks (i.e. independently running programs) you want to start in one job and how many threads each of this ranks will use.

The next step is to use the following three parameters in conjunction with either one of `sbatch`, `salloc` or `srun` to tell *Slurm* how many resources your batch or interactive job needs.

- `--nodes`: The number of compute nodes you want to use.[61]
- `--ntasks-per-node`: The number of ranks or tasks that you want to run on *each* node.[62]
- `--cpus-per-task`: The number of **threads** that *each* of your ranks or tasks will start.[63]

If you use a multithreaded program and no additional parallelising strategy – i.e. if you are **not** using a hybrid parallelisation by combining threads and a MPI library –, your program will only be able to use a single compute node! Even if your *Slurm* job script asks for more than one compute node (via `--nodes=X` with X> 1) this will not make your program run faster: Your program will run on the first assigned compute node and all other compute nodes assigned to your *Slurm* job will remain idle! If you were to start your program in the *Slurm* script by prepending the `srun` command to the program name, *Slurm* would start your program on every single compute node such that multiple instances of your program would run simultaneously, read from the same input files and write to the same output files. Typically, the resulting output files will be useless.

If you use a multithreaded program and allocate only one cpu per task, **all** of your threads will be pinned to one single cpu core and thus your program will run slower then if you would not have used multithreading at all.

There exist many more configuration options that are best described by SchedMD in their CPU Management User and Administrator Guide[64] and Support for Multi-core/Multi-thread Architectures[65].

The memory allocation and management is described in section 4.4.8

[61] See also page 113.
[62] See also page 114.
[63] See also page 113.
[64] `https://slurm.schedmd.com/cpu_management.html`
[65] `https://slurm.schedmd.com/mc_support.html`

4.4.3 Interactive jobs

4.4.3.1 `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 91) – 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 SBATCH *statements inside of Slurm job scripts.*

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 = (–nodes=4) × (–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) × (--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!

4.4.3.2 `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
```

Please note that once Slurm "grants a job allocation", i.e. when the prompt returns after you submitted via `salloc`, you are still logged in to one of the gateway servers. Confirm this by invoking the command `hostname`. You can switch to the compute nodes that got allocated to your interactive Slurm job and run an interactive shell there with

```
srun --pty bash -l
```

If you exit this shell before your allocated interactive Slurm job ran out of time, via `exit` or the `Ctrl-d` command line shortcut, your Slurm job allocation will continue. You can re-login at any time later until the alloted amount of time has expired. This approach is the closest you can get with the Slurm scheduler on a compute cluster to working on your own workstation.

On the other hand, you can also opt to stay logged in on one of the gateway servers and start several interactive jobs simultaneously, e.g. 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.
```

4.4.4 Batched job script execution

If you do not 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...
```

4.4.4.1 `sbatch` - Submit a job script

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 91) 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 77.

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`.

4.4.5 Controlling running and finished jobs

4.4.5.1 `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=democript
   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
```

You can let `squeue` show periodic updates of its output with the iterate parameter. Simply add `-i 30` to the `squeue` parameter list and squeue will run indefinitely and update its output every 30 seconds until you abort it via `STRG+C`.

Please do **not** use the `watch` command in conjunction with `squeue` as this places a heavy burden on the *Slurm* server.

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
    ↪ (Resources)
```

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 back-filling, 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[66] 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
```

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

```
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
```

4.4.5.2 `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
   UserId=<username>(<uid>) GroupId=<username>(<gid>)
    ↪ MCS_label=N/A
   Priority=21158 Nice=0 Account=itmc QOS=normal
   JobState=CANCELLED Reason=None Dependency=(null)
```

4.4.5.3 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
```

4.4.5.4 seff, sacct - show post job performance analysis

In order to be able to see for yourself whether your job has efficently used the allocated resources, 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 resource 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
```

4.4.6 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=62000`
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=255000`
cgpu04-[001]	all private cgpu04 epyc_7252 a6000 gpu ib_1to3	1 × AMD EPYC 7252 3.1 GHz, L3 cache 64 MB 1 × Nvidia®RTX A6000	8	256 GB `MaxMemPerNode=255000`
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=255000`
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=1029000`
cquad03-[001-002]	all private cquad03 xeon_gold_6230 ib_1to3	4 × Intel®Xeon Gold 6230 2.1 GHz, L3 cache 28 MB	80	512 GB `MaxMemPerNode=498000`

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=62000
cstd10-[006-032]	all public cstd01 xeon_e52640v4 ib_1to1 nonblocking_comm	2 × Intel®Xeon E5 2640v4 2.4 GHz, L3 cache 25 MB	20	256 GB MaxMemPerNode=250000
cstd02-[001-044]	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=62000
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=255000
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=187000
cstd05-[001-005]	all private cstd05 epyc_7542 ib_1to3	2 × AMD EPYC 7542 CPU 2.49 GHz, L3 cache 128 MB	64	1024 GB MaxMemPerNode=1029000
cstd06-[001]	all private cstd06 xeon_gold_6242r ib_1to3	2 × Intel Xeon Gold 6242R CPU 3.1 GHz, L3 cache 35 MB	40	96 GB MaxMemPerNode=92000

continued on next page . . .

. . . continued from previous page

Nodelist	Features	CPU Type GPU Type	max. cores	max. memory
cstd07-[001]	all private cstd07 epyc_7542 ib_1to3	2 × AMD EPYC 7542 CPU 2.49 GHz, L3 cache 128 MB	64	256 GB MaxMemPerNode=255000
cstd08-[001]	all private cstd08 epyc_7453 ib_1to3	2 × AMD EPYC 7542 CPU 2.75 GHz, L3 cache 64 MB	56	512 GB MaxMemPerNode=498000
cstd09-[001]	all private cstd09 epyc_7313 ib_1to3	2 × AMD EPYC 7313 CPU 3.00 GHz, L3 cache 128 MB	32	768 GB MaxMemPerNode=757000

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".

4.4.7 Generic Resource (GRES) - request a GPU

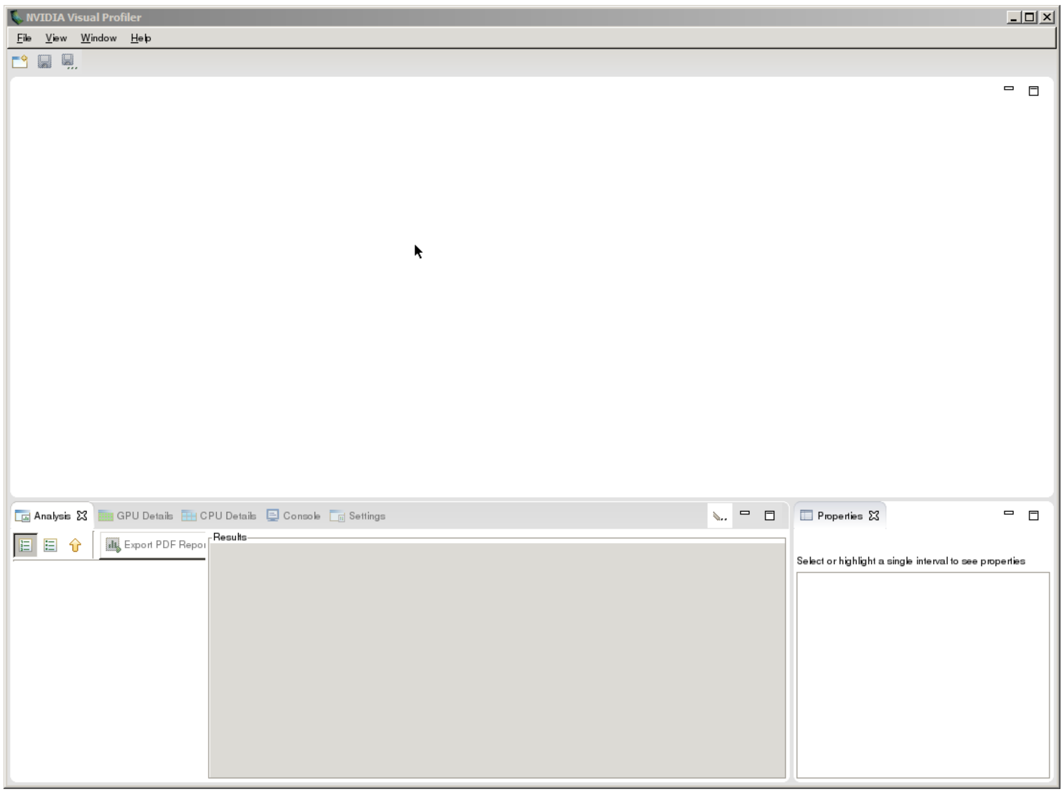
Reserving a GPU node by using constraints (see page 101) is only one half of the story. Other users may be already using the GPU when your job starts on one of those nodes and they seem too valuable to use them just for CPU-bound tasks.

GPUs are defined as a *Generic Resource* (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
```



Select or highlight a single interval to see properties

Figure 4.49: 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

4.4.8 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 aborted. 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 102. The maximum upper limit per core can be derived with the inequality

$$\text{cpus-per-task} \times \text{mem-per-cpu} < \text{MaxMemPerNode}$$

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 (512 MB 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 102.

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**!

4.4.9 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
```

4.4.10 *SBATCH* statements inside of Slurm job scripts

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 --cpus-per-task=<ncpus>`

  In Slurm context a CPU is a consumable resource offered by a node. It can refer to a socket, a core or a hardware thread, based on the Slurm configuration.

  On LiDO3 a CPU means a single core.

- `#SBATCH --ntasks=<ntasks>`

  The Slurm documentation says about this parameter: *"`sbatch` does not launch tasks, it requests an allocation of resources and submits a batch script. This option advises the Slurm controller that job steps run within the allocation will launch a maximum of number tasks and to provide for sufficient resources. The* *default is one task per node, but note that the `--cpus-per-task` option will change this default."*

— Quoted from the `sbatch` manpage and `sbatch` documentation.[67]

While this explanation is perfectly valid, it nonetheless confuses most people reading it for the first time. Let us consider the following examples to clear things up:

- If you will run a single application that will not use an MPI library, regardless whether the application runs with one or multiple threads, use `--ntasks=1` If you do not know what an MPI library is, use this setting, too.
- If you want to run more than one application concurrently within the same Slurm batch script, e.g. a process that provides data (a database server, some simulation outputting results) and another process that postprocesses this data, ask Slurm for as many separate tasks as you will be running processes at most simultaneously (if your processes run multithreaded, count every thread as one task).
- When using an MPI application without additional threading (i.e. no hybrid parallelisation): If you would normally start your MPI application on a local workstation (i.e. no Slurm context) with

  ```
  $ mpirun -np <ntasks> /path/to/application
    ↪ <application arguments>
  ```

  you specify `<ntasks>` as argument to the `sbatch` option `--ntasks` and instead invoke

  ```
  $ srun /path/to/application <application arguments>
  ```

- `#SBATCH --ntasks-per-node=<ntasks>`

  Request that ntasks be invoked on each node. If used with the `--ntasks` option, the `--ntasks` option will take precedence and the `--ntasks-per-node` will be treated as a maximum count of tasks per node.

  Meant to be used with the `--nodes` option.

This is related to the `--cpus-per-task` option, but does not require knowledge of the actual number of cpus on each node. In some cases, it is more convenient to be able to request that no more than a specific number of tasks be invoked on each node.

Examples of this include submitting a hybrid MPI/OpenMP app where only one MPI "task/rank" should be assigned to each node while allowing the OpenMP portion to utilize all of the parallelism present in the node, or submitting a single setup/cleanup/monitoring job to each node of a pre-existing allocation as one step in a larger job script.

- `#SBATCH --cpus-per-task=<ncores>`

  Request that `<ncores>` cpu cores will be reserved for each task/rank. These will be used by threads started by each task/rank.

  In contrast the option `--cpus-per-task` specifies how many CPU cores each task can use.

- `#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[68]) that is not the login user.

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

[68] 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.

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.

- `#SBATCH --no-requeue`

  Indicates that a job should not rerun if it fails. As per default a job is restarted (keeping its Slurm job id) if it was aborted due to cluster-side issues, e.g. a node failure due to hardware defects. Some job scripts are not prepared for instant restart or precious progress in temporary result files can be deleted by a unexpected restart. In this cases a user can prevent any restart on the job script level.

4.4.11 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`

continued on next page ...

. . . continued from previous page

Action	Slurm
Expected start time [69]	`squeue --start -j <job_id>`
Queues/partitions	`sinfo -s`
`scontrol show partition`	

Node list	`sinfo -N`
	`scontrol show nodes`

Node details	`scontrol show node <nodename>`

Queue [70]	`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>`

[69] See also section *`scontrol, squeue, showq` - Query Job status* on page 95 for background informations.

[70] See also section *Format options for slurm commands* on page 119.

4.4.12 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.

4.4.13 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.

continued on next page ...

. . . continued from previous page

Field	Explanation
`%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
`%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

4.4.14 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: 62188
# cquad01: 255688
# cquad02: 1029788
#SBATCH --mem=60000

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@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: 62188
# cquad01: 255688
# cquad02: 1029788
#SBATCH --mem=60000

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@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 nvidia/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: Abaqus

4.5.3.1 On a single compute node

To run Abaqus in shared memory mode on a **single compute node**, invoke the following script via

```
sbatch run_abaqus_single_node_through_slurm.sh
```

It asks for 1 compute node with 8 cores for 90 minutes:

```
#!/bin/bash -l

#SBATCH --job-name=abaqusjob1
#SBATCH --partition=short
#SBATCH --time 01:30:00
# Request 8 CPU cores in order to run Abaqus with 8 threads
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=8
#SBATCH --mem=50G              # maximum memory required (in GB)
#inactive-SBATCH --mem=50000M # maximum memory required (in MB)
#SBATCH -o %x-%j.out           # write standard output to a file
                               # named after job name given above
                               # and job ID assigned by Slurm.
```

```
#SBATCH -e %x-%j.err          # write error messages a file
                              # named after job name given above
                              # and job ID assigned by Slurm
# send mail when for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de


## change to directory where your Abaqus input files are stored
## Typically, this is also the location where your Slurm job
## script got submitted such that we can use a conveniently
## set environment variable. Otherwise, specify the absolute
## path to your Abaqus input files here instead.
## The actual error message this avoids is, e.g.,
##   "dlopen of libhwloc.so failed"
cd $SLURM_SUBMIT_DIR


## Abaqus can be activated using one of the following module
## files. (Note that the list may have been extended since this
## documenation got written.)
#inactive   module load abaqus/2018-hotfix10
#inactive   module load abaqus/2021-hotfix4
#inactive   module load abaqus/2022-hotfix6
module load abaqus/2023


## When working with user subroutines, the Intel Fortran compiler
## 2019 is required. It is provided by one of the
## intel/studio-xe/19* modulefiles.
module load intel/studio-xe/19.0.3.203


## Users must unset the SLURM environment variable SLURM_GTIDS.
## Failure to do so will cause Abaqus to get stuck due to the
## MPI that Abaqus ships with not supporting the SLURM
## scheduler. (SLURM_GTIDS should be unset for both
## interactive/GUI and batch jobs.)
unset SLURM_GTIDS


## By default, Abaqus will use /tmp to store temporary files.
## But /tmp on LiDO3 compute nodes has a capacity of merely 36
## GiB. Whereas the dedicated local file space /scratch provides
## up to 1.8 TiB. So, use the latter to avoid risking to overflow
## the root file system, causing a compute node to freeze.
## Be aware, though, that this only works for simulations that
## run on a single compute node. (For MPI parallel simulations
## a shared network file system is required by Abaqus.)
## Create a personal subdirectory under /scratch, if necessary.
TMPDIR=/scratch/${USER}
test -d "${TMPDIR}" || mkdir -p "${TMPDIR}"

## ================================================================
## Finally, invoke Abaqus for batch job execution.
##
## To avoid having to adjust both the '#SBATCH --cpus-per-task=..'
## line above and the abaqus invocation line below (and avoid
## risking configuration mismatches) use the conveniently set
## Slurm environment variable SLURM_CPUS_ON_NODE to tell Abaqus
## how many CPU cores are available once this Slurm job runs.
##
## Unfortunately, Slurm does not set such a variable for the
## max. memory. (Its value can be queried by 'ulimit -m' from
## this job script when the job runs.) So, when using the Abaqus
## option "memory=..." make sure that its value matches that of
## the leading #SBATCH --mem="..." line.
##
## Uncomment the appropriate line marked with '#inactive' and
## adapt it to your needs.

#inactive   abaqus job=example mp_mode=threads
    ↪ cpus=${SLURM_CPUS_ON_NODE} scratch=${TMPDIR} double
    ↪ -interactive
#inactive   abaqus job=example user=user_function01.f
    ↪ mp_mode=threads cpus=${SLURM_CPUS_ON_NODE}
    ↪ scratch=${TMPDIR} double -interactive
#inactive   abaqus job=example user=user_function01.f
    ↪ mp_mode=threads cpus=${SLURM_CPUS_ON_NODE}
    ↪ scratch=${TMPDIR} double=both interactive
#inactive   abaqus job=example user=user_function01.f
    ↪ mp_mode=threads cpus=${SLURM_CPUS_ON_NODE}
    ↪ scratch=${TMPDIR} double=both output_precision=full
    ↪ interactive
#inactive   abaqus job=example user=user_function01.f
    ↪ mp_mode=threads cpus=${SLURM_CPUS_ON_NODE}
    ↪ scratch=${TMPDIR} double -interactive
    ↪ standard_parallel=solver
#inactive   abaqus cae noGUI=createTest.py


## When using Abaqus' built-in checkpoint and restart feature
## add the following to the initial input file (refer to official
## Abaqus documentation for detail):
##   *RESTART, WRITE, OVERLAY, FREQUENCY=10
## where OVERLAY saves only one state, i.e. overwrites the
## restart file every time new restart information is written and
## FREQUENCY=N writes restart information every N timesteps. To
## restart the job, create a new input file newJobName with only a
## single line:
##   *RESTART, READ
## Then run Abaqus specifying both the new and old job names:
##
#inactive   abaqus jobname=newJobName oldjob=oldJobName
    ↪ scratch=${TMPDIR}
```

Listing 4.4: Contents of file "run_abaqus.smp_modefor_single_compute_-node.sh'
A copy is available from path `/cluster/sfw/lido3-examples/abaqus/run_abaqus.smp_mode.for_single_compute_node.sh`

The files

```
example.inp user_function01.f
```

should obviously be replaced with your own Abaqus input files.

4.5.3.2 On multiple compute nodes

You can use Abaqus on more than one computing node. The main difference is that you need to tell Abaqus which compute node the Slurm scheduler has picked out for the Slurm job. The listing 4.4 needs a few more tweaks, though, for a simulation with 2 compute nodes with 20 cores each, i.e. 40 parallel Abaqus processes, for 90 minutes:

```
#!/bin/bash -l

#SBATCH --job-name=abaqusjob2
#SBATCH --partition=short
#SBATCH --time 01:30:00
# Request 2x20 CPU cores in order to run Abaqus with 40 MPI
    ↪ processes
#SBATCH --nodes=2 --ntasks-per-node=20 --cpus-per-task=1
#SBATCH --mem=50G              # maximum memory required (in GB)
#inactive-SBATCH --mem=50000M # maximum memory required (in MB)
#SBATCH -o %x-%j.out           # write standard output to a file
```

```
                              # named after job name given above
                              # and job ID assigned by Slurm.
#SBATCH -e %x-%j.err          # write error messages a file
                              # named after job name given above
                              # and job ID assigned by Slurm
# send mail when for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de


## change to directory where your Abaqus input files are stored
## Typically, this is also the location where your Slurm job
## script got submitted such that we can use a conveniently
## set environment variable. Otherwise, specify the absolute
## path to your Abaqus input files here instead.
## The actual error message this avoids is, e.g.,
##   "dlopen of libhwloc.so failed"
cd $SLURM_SUBMIT_DIR


## Abaqus can be activated using one of the following module
## files. (Note that the list may have been extended since this
## documenation got written.)
#inactive   module load abaqus/2018-hotfix10
#inactive   module load abaqus/2021-hotfix4
#inactive   module load abaqus/2022-hotfix6
module load abaqus/2023


## When working with user subroutines, the Intel Fortran compiler
## 2019 is required. It is provided by one of the
## intel/studio-xe/19* modulefiles.
#inactive     module load intel/studio-xe/19.0.3.203


## Users must unset the SLURM environment variable SLURM_GTIDS.
## Failure to do so will cause Abaqus to get stuck due to the
## MPI that Abaqus ships with not supporting the SLURM
## scheduler. (SLURM_GTIDS should be unset for both
## interactive/GUI and batch jobs.)
unset SLURM_GTIDS


## By default, Abaqus will use /tmp to store temporary files.
## But /tmp on LiDO3 compute nodes has a capacity of merely 36
## GiB. Whereas the dedicated local file space /scratch provides
## up to 1.8 TiB. So, use the latter to avoid risking to overflow
## the root file system, causing a compute node to freeze.
## But, to the LiDO team's knowledge, Abaqus sets the variable
## MPI_WORKDIR relative to the path given by Abaqus' command line
## option 'scratch'. At the same time, MPI_WORKDIR must be a
## path that is globally accessible by all MPI processes on all
## compute nodes. Such that MPI_WORKDIR must not be set to a
## local file system. So, for the time being, do not tamper
## with 'scratch=${TMPDIR}' on the Abaqus command line when
## using 'mp_mode=mpi'.

## Let Abaqus know which compute nodes we got assigned from Slurm:
## Locate and copy the default abaqus_v6.env from the currently
## enabled Abaqus version to the current directory. Then tweak it.
rm -f abaqus_v6.env
cp -a $( find $( dirname $( dirname $( which abaqus ))) -name
   ↪ "abaqus_v6.env" ) .
srun hostname | xargs -n 1 | while read hostname; do echo
   ↪ "['${hostname}',1]"; done | paste -sd, | sed -e
   ↪ 's/^/mp_host_list=[/; s/$/]/;' >> abaqus_v6.env;


## ================================================================
## Finally, invoke Abaqus for batch job execution.
##
## To avoid having to adjust both the '#SBATCH --cpus-per-task=..'
## line above and the abaqus invocation line below (and avoid
## risking configuration mismatches) use the conveniently set
## Slurm environment variable SLURM_CPUS_ON_NODE to tell Abaqus
## how many CPU cores are available once this Slurm job runs.
##
## Unfortunately, Slurm does not set such a variable for the
## max. memory. (Its value can be queried by 'ulimit -m' from
## this job script when the job runs.) So, when using the Abaqus
## option "memory=..." make sure that its value matches that of
## the leading #SBATCH --mem="..." line.
##
## Uncomment the appropriate line marked with '#inactive' and
## adapt it to your needs.
```

```
abaqus job=example mp_mode=mpi cpus=${SLURM_CPUS_ON_NODE}
    ↪ parallel=domain domains=${SLURM_NTASKS}
    ↪ cpus=${SLURM_NTASKS} dynamic_load_balancing=on double
    ↪ interactive
```

Listing 4.5: Contents of file "run_abaqus.mpi_mode.for_multiple_compute_-nodes.sh'
A copy is available from path `/cluster/sfw/lido3-examples/abaqus/run_abaqus.mpi_mode.for_multiple_compute_nodes.sh`

4.5.4 Common software example: Ansys CFX

4.5.4.1 On a single compute node

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 for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#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 4.6: 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.

4.5.4.2 On multiple compute nodes

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 mycfxsimulation
#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 for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#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 4.7: Contents of file 'run_cfx_multiple_nodes_through_slurm.sh'

Thirdly, CFX uses SSH for the communication between nodes. Thus you need to setup passphrase-less inter-node SSH access (see section 4.2.5 on page 58), if you are using multiple nodes at once.

4.5.5 Common software example: Ansys Fluent

4.5.5.1 On a single compute node

The following script, when invoked via

```
sbatch run_fluent.sh
```

asks for 1 compute node with 20 cores for 60 minutes and all 60 GiB on that compute node for exclusive usage:

```
#!/bin/bash -l

#SBATCH --job-name myfluentsimulation
#SBATCH --partition=short
#SBATCH --time 0-01:00:00
#SBATCH --mem 60G --exclusive
#SBATCH --nodes=1 --ntasks-per-node=20 --cpus-per-task=1
```

```
#SBATCH --error=slurmjob.%j.stderr
#SBATCH --output=slurmjob.%j.stdout
# send mail when for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de


export FLRECENT="${SLURM_SUBMIT_DIR}"
export OMP_NUM_THREADS=1

# load modulefiles for Ansys Fluent
module purge
module load openmpi/mpi_thread_multiple/no_cuda/4.1.1
module load fluent/2023R1

# run the fluent simulation
fluent 3ddp -g -cflush -t ${SLURM_NTASKS} -pib -mpi=openmpi -i
    ↪ mycase.jou
```

Listing 4.8: Contents of file 'run_fluent_single_compute_node.sh'

The file

```
mycase.jou
```

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

4.5.5.2 On multiple compute nodes

You can use Ansys Fluent on more than one computing node. The main difference is that you need to tell Ansys Fluent which nodes the Slurm scheduler has picked out for the Slurm job. The listing 4.8 needs slight tweaking for a simulation with 4 compute nodes with 20 cores each, i.e. 80 parallel Fluent processes, for 2 days and 30 minutes and 60 GiB per compute node for exclusive usage:

```
#!/bin/bash -l

#SBATCH --job-name myfluentsimulation
#SBATCH --partition=long
#SBATCH --time 2-00:30:00
#SBATCH --mem 60G --exclusive
```

```
#SBATCH --nodes=4 --ntasks-per-node=20 --cpus-per-task=1
#SBATCH --error=slurmjob.%j.stderr
#SBATCH --output=slurmjob.%j.stdout
# send mail when for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de


export FLRECENT="${SLURM_SUBMIT_DIR}"
export OMP_NUM_THREADS=1

# generate node list
NODELIST=$( for node in $( scontrol show hostnames
   ↪ ${SLURM_JOB_NODELIST} | uniq ); do echo
   ↪ "${node}:${SLURM_NTASKS_PER_NODE}"; done | paste -sd, )
# calculate the number of cores actually used
CORES=$(( ${SLURM_JOB_NUM_NODES} * ${SLURM_NTASKS_PER_NODE} ))

# load modulefiles for Ansys Fluent
module purge
module load openmpi/mpi_thread_multiple/no_cuda/4.1.1
module load fluent/2023R1

# run the fluent simulation
fluent 3ddp -g -cflush -t ${CORES} -cnf="${NODELIST}" -pib
   ↪ -mpi=openmpi -i mycase.jou
```

Listing 4.9: Contents of file 'run_fluent_multiple_compute_nodes.sh'

As before, the file

```
mycase.jou
```

mentioned in the command line to invoke Ansys Fluent inside your Slurm job script should obviously be replaced with your own Ansys Fluent problem description file.

4.5.6 Common software example: Ansys Mechanical APDL

4.5.6.1 On a single compute node

In order to run Ansys Mechanical APDL on a **single compute node**, invoke the following script via

```
sbatch run_ansys_smp_mode_single_node.sh
```

It asks for one compute node with 8 cores and 8 GB of main memory for 40 minutes and will invoke Ansys Mechanical APDL with a single process and 8 threads. A copy of it is available on LiDO3 from the path `/cluster/sfw/lido3-examples/ansys-mechanical/run_ansys.smp_mode.for_single_compute_node.sh`.

```
#!/bin/bash
#=================================================================
# SLURM - job script template for Ansys Mechanical APDL
#=================================================================
#=================================================================
# Slurm options
#=================================================================
#SBATCH --job-name=ansys_smp
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=8
#SBATCH --mem=8G
#SBATCH --time=00:40:00
#SBATCH --partition=short
#SBATCH --error=error_file.%j.e
#SBATCH --output=output_file.%j.o
# send mail when for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de

#=================================================================
# Load modules
#=================================================================
module purge
module load ansys/2023R1
# Ansys ships with all the required libraries, both Intel MPI
# and Open MPI. There is no need to load any other of the
# LiDO3 modulefiles for a standalone MPI installation.

# Note that Ansys ships with all the required libraries, both for
# Intel MPI and Open MPI libraries. There is no need to load any
# other of the LiDO3 modulefiles for a standalone MPI
# installation.


# Change to directory where input file is stored (see input_file
# definition below) and where result files should be saved to.
directoryForInputAndOutput=${SLURM_SUBMIT_DIR}

#=================================================================
# Prepare variable parts of Ansys command line options
#=================================================================
job_name=ansys_result
input_file=testlido.dat

# Some Slurm environment variables may only be set if they were
# explicitly passed to Slurm, either on the command line or in the
# Slurm job script. Set them to 1, if unset.
# (Note: the leading colons are there on purpose, indeed. This is
#        bash syntax for a conditional assignment.)
: ${SLURM_NTASKS:=1}
: ${SLURM_CPUS_PER_TASK:=1}

# Calculate now the argument to the Ansys command line option -np,
# i.e. the total number of requested (MPI or OpenMP)
# processes. (Of couse, you could hardcode the number passed as
# argument to -np in the ansys invocation line. But experience
# has shown that hardcoding is too errorprone. When tends to
# forget to adapt one of the two locations when one adapts this
# script to request from Slurm more or less compute resources:
# either the SBATCH line or the ansys invocation line.)
numberOfProcesses=$((( ${SLURM_NTASKS} * ${SLURM_CPUS_PER_TASK}
   ↪ )))

#=================================================================
# Program execution
#=================================================================
# help screen for interactive job
#anshelp

# Start calculation in batch mode in shared-memory
   ↪ multiprocessing (SMP) mode,
# i.e. on a single compute node
cd ${directoryForInputAndOutput}


# ansysXXX command line options explained:
# -b, -b list, -b nolist
#            Run in batch mode, taking input from standard input or
#            the file specified with -i.
#            "-b" and "-b list" are synonymous and both cause the
#            input file contents to be echoed into the output file.
#            "-b nolist" does not include the input commands in the
#            output file.
# -dir dirname
#           Specifies which directory to solve in. By default, it
#           will be the directory you launch the executable from.
#           But you may want to write files into a different
#           location.
# -i filename
#           Specify the name of the input file in the working
#           directory to read as input
# -o filename
#           Specify the name of the output file in the working
#           directory to place output into.
# -j jobname
#           By default, the jobname is "file" and all files
#           created are file.nnn. Specify a unique name as a
#           string for jobname.
#           The value is unrelated to the Slurm jobname.
# -np n
#           This specifies the number of processors to use during
#           solve.
# -m value
#           Defines the total memory to reserve for the program.
#           The ANSYS documentation recommends to reserve the
#           required memory up front rather than letting ANSYS
#           grab as it needs.
#           However, within a running Slurm job, it should not be
#           necessary to reserve the memory up front and we have
#           so far no reason to state otherwise! Why? You are
#           guaranteed to receive as much memory for the duration
#           of your Slurm job as you requested once the Slurm job
#           started. No other process can eat away your RAM
#           (unless you log in to the compute node interactively
#           via 'ssh' and start other memory-intensive processes.)
#           On a ordinary server - without Slurm supervision -
#           where anybody can log in and start new processes while
#           your process is already running, on the other hand,
#           processes can indeed fight over memory such that it
#           reserving everything ANSYS might need up front avoids
#           seeing ANSYS crash late in a simulation due to low
#           remaining RAM.
# -s read, -s noread
#           Specifies whether the program reads the "start.ans"
#           file at start-up from where ANSYS got installed (typ-
#           ically /cluster/sfw/ansys/*/*/ansys/apdl/start.ans).
#           If you omit the -s option, ANSYS reads the start.ans
#           file in interactive mode and not in batch mode.
# -l language
#           Specifies a language file to use other than
#           US English.
# -smp
#            Stands for shared-memory multiprocessing.
#            Run ANSYS with multiple threads (when you requested
#            only a single compute node, i.e. used
#            '#SBATCH --nodes=1' above)
# -dis
#            Enables Distributed ANSYS. See the Parallel Processing
#            Guide for more information.
# -mpi intelmpi, -mpi openmpi
#            Specifies the type of MPI to use. "-mpi intelmpi" and
#            "-mpi openmpi" should both work smoothly on LiDO3.
#            See the Parallel Processing Guide for more
#            information.
# -machines string
#            Specifies the machines (in "string", using colons as
#            separators, without whitespaces) on which to run a
#            Distributed ANSYS analysis.
#            Example: cstd01-001:20:cstd01-002:20:cstd01-003:20
#            Dynamically determined in this Slurm job script
#            template.
#            See Starting Distributed ANSYS in the Parallel
#            Processing Guide for more information.
ansys231 -b -j ${job_name} -dir ${directoryForInputAndOutput} -i
    ↪ ${input_file} -s read -l en-us -smp -np ${numberOfProcesses}
```

Listing 4.10: Contents of file 'run_ansys.smp_mode.for_single_compute_-node.sh'
A copy is available from path `/cluster/sfw/lido3-examples/ansys-mechanical/run_ansys.smp_mode.for_single_compute_node.sh`

4.5.6.2 On multiple compute nodes

In order to run Ansys Mechanical APDL on **multiple compute nodes**, it is mandatory to set up passphrase-less inter-node SSH access (see section 4.2.5 on page 58). Because Ansys Mechanical APDL uses SSH to initially set up the MPI environment required for a cross-node simulation.

With passphrase-less inter-node SSH access in place, invoke the following script via

```
sbatch run_ansys_distributed_mode_multiple_nodes.sh
```

It asks for two compute nodes, each with 20 cores and 30 GB of main memory for 40 minutes. A copy of the Slurm job template to run Ansys Mechanical APDL as shown in listing 4.11 is available on LiDO3 from the path `/cluster/sfw/lido3-examples/ansys-mechanical/run_ansys.distributed_mode.for_multiple_compute_nodes.sh`.

```
#!/bin/bash
#===================================================================
# SLURM - job script template for Ansys Mechanical APDL
#===================================================================
#===================================================================
# Slurm options
#===================================================================
#SBATCH --job-name=ansys_dis
#SBATCH --nodes=2 --ntasks-per-node=1 --cpus-per-task=20
#optional: use compute nodes with higher bandwidth interconnect:
    ↪ SBATCH --constraint=cstd02
#SBATCH --mem=30G
#SBATCH --time=00:40:00
#SBATCH --partition=short
#SBATCH --error=error_file.%j.e
#SBATCH --output=output_file.%j.o
# send mail when for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de

#===================================================================
# Load modules
#===================================================================
module purge
module load ansys/2023R1
# Ansys ships with all the required libraries, both Intel MPI
# and Open MPI. There is no need to load any other of the
# LiDO3 modulefiles for a standalone MPI installation.

# Note that Ansys ships with all the required libraries, both for
# Intel MPI and Open MPI libraries. There is no need to load any
# other of the LiDO3 modulefiles for a standalone MPI
# installation.


# Change to directory where input file is stored (see input_file
# definition below) and where result files should be saved to.
directoryForInputAndOutput=${SLURM_SUBMIT_DIR}
```

```
#=====================================================================
# Prepare variable parts of Ansys command line options
#=====================================================================
job_name=ansys_result
input_file=testlido.dat

# Some Slurm environment variables may only be set if they were
# explicitly passed to Slurm, either on the command line or in the
# Slurm job script. Set them to 1, if unset.
# (Note: the leading colons are there on purpose, indeed. This is
#        bash syntax for a conditional assignment.)
: ${SLURM_JOB_NUM_NODES:=1}
: ${SLURM_NTASKS:=1}
: ${SLURM_NTASKS_PER_NODE:=1}
: ${SLURM_CPUS_PER_TASK:=1}

# Calculate now the argument to the Ansys command line option -np,
# i.e. the total number of requested (MPI or OpenMP)
# processes. (Of couse, you could hardcode the number passed as
# argument to -np in the ansys invocation line. But experience
# has shown that hardcoding is too errorprone. When tends to
# forget to adapt one of the two locations when one adapts this
# script to request from Slurm more or less compute resources:
# either the SBATCH line or the ansys invocation line.)
numberOfProcesses=$((( ${SLURM_JOB_NUM_NODES} *
    ↪ ${SLURM_NTASKS_PER_NODE} * ${SLURM_CPUS_PER_TASK} )))

# When running Distributed Ansys across multiple hosts, we need to
# specify the number of cores we want to use on each machine. (The
# following instruction assumes a homogeneous compute node
# request: the same number of tasks and cpus-per-task on each
# compute node.)
numberOfProcessesPerNode=$((( ${SLURM_NTASKS_PER_NODE} *
    ↪ ${SLURM_CPUS_PER_TASK} )))
machinesString=$( scontrol show hostnames | sed -e
    ↪ "s/\$/:${numberOfProcessesPerNode}/" | paste -sd: )
#DEBUG echo ${machinesString}

#=====================================================================
# Program execution
#=====================================================================
# help screen for interactive job
#anshelp

# Start calculation in batch mode in distributed mode, i.e. on
# multiple compute nodes
cd ${directoryForInputAndOutput}
```

```
# ansysXXX command line options explained:
# -b, -b list, -b nolist
#           Run in batch mode, taking input from standard input or
#           the file specified with -i.
#           "-b" and "-b list" are synonymous and both cause the
#           input file contents to be echoed into the output file.
#           "-b nolist" does not include the input commands in the
#           output file.
# -dir dirname
#           Specifies which directory to solve in. By default, it
#           will be the directory you launch the executable from.
#           But you may want to write files into a different
#           location.
# -i filename
#           Specify the name of the input file in the working
#           directory to read as input
# -o filename
#           Specify the name of the output file in the working
#           directory to place output into.
# -j jobname
#           By default, the jobname is "file" and all files
#           created are file.nnn. Specify a unique name as a
#           string for jobname.
#           The value is unrelated to the Slurm jobname.
# -np n
#           This specifies the number of processors to use during
#           solve.
# -m value
#           Defines the total memory to reserve for the program.
#           The ANSYS documentation recommends to reserve the
#           required memory up front rather than letting ANSYS
#           grab as it needs.
#           However, within a running Slurm job, it should not be
#           necessary to reserve the memory up front and we have
#           so far no reason to state otherwise! Why? You are
#           guaranteed to receive as much memory for the duration
#           of your Slurm job as you requested once the Slurm job
#           started. No other process can eat away your RAM
#           (unless you log in to the compute node interactively
#           via 'ssh' and start other memory-intensive processes.)
#           On a ordinary server - without Slurm supervision -
#           where anybody can log in and start new processes while
#           your process is already running, on the other hand,
#           processes can indeed fight over memory such that it
#           reserving everything ANSYS might need up front avoids
#           seeing ANSYS crash late in a simulation due to low
#           remaining RAM.
# -s read, -s noread
#           Specifies whether the program reads the "start.ans"
#           file at start-up from where ANSYS got installed (typ-
#           ically /cluster/sfw/ansys/*/*/ansys/apdl/start.ans).
#           If you omit the -s option, ANSYS reads the start.ans
#           file in interactive mode and not in batch mode.
# -l language
#           Specifies a language file to use other than
#           US English.
# -smp
#           Stands for shared-memory multiprocessing.
#           Run ANSYS with multiple threads (when you requested
#           only a single compute node, i.e. used
#           '#SBATCH --nodes=1' above)
# -dis
#           Enables Distributed ANSYS. See the Parallel Processing
#           Guide for more information.
# -mpi intelmpi, -mpi openmpi
#           Specifies the type of MPI to use. "-mpi intelmpi" and
#           "-mpi openmpi" should both work smoothly on LiDO3.
#           See the Parallel Processing Guide for more
#           information.
# -machines string
#           Specifies the machines (in "string", using colons as
#           separators, without whitespaces) on which to run a
#           Distributed ANSYS analysis.
#           Example: cstd01-001:20:cstd01-002:20:cstd01-003:20
#           Dynamically determined in this Slurm job script
#           template.
#           See Starting Distributed ANSYS in the Parallel
#           Processing Guide for more information.


## Using Intel MPI
ansys231 -b -j ${job_name} -dir ${directoryForInputAndOutput} -i
   ↪ ${input_file} -s read -l en-us -np ${numberOfProcesses}
   ↪ -dis -mpi intelmpi -machines ${machinesString}

## Using Open MPI
```

```
#ansys231 -b -j ${job_name} -dir ${directoryForInputAndOutput} -i
    ↪ ${input_file} -s read -l en-us -np ${numberOfProcesses}
    ↪ -dis -mpi openmpi -machines ${machinesString}
```

Listing 4.11: Contents of file 'run_ansys.distributed_mode.for_multiple_-compute_nodes.sh
A copy is available from path `/cluster/sfw/lido3-examples/ansys-mechanical/run_ansys.distributed_mode.for_multiple_compute_nodes.sh`

4.5.7 Common software example: Ansys Workbench

Ansys Workbench is launched via the command `runwb2` after an Ansys module was loaded. If you want to use it interactively, it is strongly recommended to use a ThinLinc session (for using ThinLinc, see section *4.2.4 Cendio ThinLinc*). The alternative, SSH shells with X forwarding (within Linux or Windows), can cause run time issues.

Within a ThinLinc session open a terminal window and enter:

```
module load ansys/2023R1

salloc -N1 --mem=60G -c20 --exclusive -p short
```

After that, do not run the `srun` command. This is very important. Instead, look for the allocated node of your job and connect from a different shell in your running ThinLinc session via `ssh` to this node. It is also important not to choose insufficient parameters for memory or cores. We recommend to use the node exclusively (i.e. use the Slurm option `--exclusive`).

Next, temporarily – for the remainder of this running shell – change your home directory to your work directory (or a subdirectory therein). The benefit of it being that this temporay home directory of yours is writable to on compute nodes. (Remember that your home directory is read-only on the compute nodes, see section 4.3.2.1.)

```
ssh -X <node>
export HOME=/work/<yourusername>
```

The following changes your working directory to this temporary home directory. To avoid that Ansys Workbench will be launched from somewhere inside your default home directory (i.e. some subdirectory of `/home/$USER`) in the subsequent line. Finally, `runwb2` launches the workbench window.

```
cd ~/
runwb2
```

4.5.8 Common software example: Gaussian

In order to use Gaussian on the LiDO3 cluster, three prerequisites need to be fulfilled: Firstly, you need to enlist to the user list kept at the Faculty of Chemistry and Chemical Biology's dean's office. This is stipulated by the Gaussian license. Secondly, you need to be added to the unix group `gaussian` on LiDO3. Thirdly, for financial accounting reasons, you need to tell Slurm that you intend to use Gaussian and with how many cores you intend to do so. On Slurm job submission, the latter is checked; `sbatch` will show a screen message if the option `--license=gaussian:X` has not been provided at all or with a number that does not correspond to the number of cores requested.

The following script, when invoked via

```
sbatch run_gaussian_single_node_through_slurm.sh
```

asks for 1 compute node with 10 cores, 15 GB memory for 240 minutes and 10 Gaussian licenses for the duration of 4 hours. A copy of this Slurm job template as shown in listing 4.12 is available on LiDO3 from the path `/cluster/sfw/lido3-examples/` ↪ `gaussian/run_gaussian_single_node_through_slurm.sh`.

```
#!/bin/sh

#SBATCH --partition=med
#SBATCH --nodes=1 --tasks-per-node=10 --cpus-per-task=1
#SBATCH --time=4:00:00

# Let Slurm know that you will be using as many Gaussian licenses
# as you asked for compute cores, using the same value as for
# --tasks-per-node
#SBATCH --license=gaussian:10

# memory requirement for all CPUs in megabytes
#SBATCH --mem=15000

#SBATCH --job-name=Gaussian_simulation_H2O

## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
```

```
#SBATCH --output=/work/myusername/slurm_job%A
#SBATCH --error=/work/myusername/slurm_job%A

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de
#SBATCH --mail-type=TIME_LIMIT

# 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



## Change to the directory where your Gaussian input files
## are stored.
## PLEASE CHANGE this path to the actual one!
cd /work/${USER}/gaussian/task123

## stores the hostname in a text file
srun hostname -s > slurmhosts.${SLURM_JOB_ID}.txt

## tweak environment settings to be able to use the
## gaussian version you need (for example the latest version)
module load gaussian/16/Rev.C01-with-ATLAS-BLAS

## to save off your local working directory on /work
## where your input files are located
export wdir=`pwd`

## Point the directory Gaussian uses to store large files to a
## directory on a local file system (as opposed to, e.g., the
## parallel file system of LiDO3).
export GAUSS_SCRDIR=/scratch/${SLURM_JOBID}

## In the case your job prematurely terminates (i.e.  wall time
## exceeded, job cancellation via 'scancel'), several temporary
## files Gaussian created will be left in the /scratch directory.
## This has led multiple times in the past to a situation where
## the local /scratch directories filled up quicker than the
## automatic cleanup script would remove the orphaned files.
##
## To save off your data and clean up the /scratch when a job
## aborts, a trap function can be used. It is called once the
## login shell intercepts certain system signal (here: either one
## of the signals USR1, SIGQUIT, SIGINT, SIGABRT, SIGKILL, SIGHUP,
## SIGTERM, EXIT, TERM).
finalize_job()
{
  printf "%s  Shell function finalize_job invoked\n\n" "$( date )"

  cp -u ${GAUSS_SCRDIR}/* ${wdir}
  rm -rf ${GAUSS_SCRDIR}
}
trap -- 'finalize_job' USR1 SIGQUIT SIGINT SIGABRT SIGKILL \
                       SIGHUP SIGTERM EXIT TERM

## Your simulation will run faster if you point GAUSS_SCRDIR
## (the directory Gaussian uses to store large files that will
## be deleted after the job terminates successfully) to a local
## file system, not to /work. So, create a directory for
## your calculcations on the current node and copy your input
## files to this scratch folder. To make the directory name
## unique, use the variable $SLURM_JOBID defined automatically
## by SLURM.
mkdir ${GAUSS_SCRDIR}
cp -a * ${GAUSS_SCRDIR}/
cd ${GAUSS_SCRDIR}

## Run single-threadedly
unset OMP_NUM_THREADS

## start your Gaussian calculation
## NOTES:
## * Please adapt the expression within <> in your own script!
## * The '&' to start Gaussian and the 'wait' command are
##   mandatory for this to work. The shell needs to intercept the
##   system signals, if Gaussian runs in foreground, the 'trap'
##   will get seemingly ignored.
g16 <your-gaussian-input.gin> &
wait

## IMPORTANT: At the end of your job you will need to copy
## everything back from scratch to your original
## work directory. Otherwise, your results will get lost,
## because you can not access the local file system that
## easy and more - and contents in /scratch not belonging to
## a running job any more get cleaned at regular intervalls
## automatically.
## Given that we already have a shell function that takes
## care of this task when the Slurm job aborts, rely on it
## for a Gaussian simulation that ran smootly, too.
printf "%s  Start salvaging Gaussian output from local hard disk
    ↪ drive to persistent storage\n\n" "$( date )"
finalize_job
printf "%s  Finished salvaging Gaussian output from local hard
    ↪ disk drive to persistent storage\n\n" "$( date )"

## Optional: remove trap definition
trap - USR1 SIGQUIT SIGINT SIGABRT SIGKILL \
       SIGHUP SIGTERM EXIT TERM
```

Listing 4.12: Contents of file 'run_gaussian_single_node_through_slurm.sh'
A copy is available from path `/cluster/sfw/lido3-examples/gaussian/run_gaussian_single_node_through_slurm.sh`

The file

```
your-gaussian-input.gin
```

mentioned in listing 4.12 should obviously be replaced with your own Gaussian problem description file.

4.5.9 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
## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
#SBATCH --output=/work/myusername/tmp/slurm_job
#SBATCH --error=/work/myusername/tmp/slurm_job
```

```
# send mail when for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL
## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de

## If you intend to use the MathWorks Parallel Computing toolbox
## (commands like 'parfor', 'parpool', 'parcluster'), but are
## computing on a single compute node only, then uncomment the
## following block.
##
## Why do we recommend the following settings? Matlab uses a
## subdirectory named ~/.matlab/local_cluster_jobs to store files
## to synchronize the worker processes of Matlab threads started
## in the background by Parallel Computing toolbox. For improved
## I/O performance ~/.matlab is actually a shortcut (technically
## a symbolic link) that redirects to /work/<user>/.matlab (see
## section~\ref{subsec:sumbolic:linkhomedir} of the "LiDO3 First
## Contact" document). So, Matlab writes - in the background -
## temporary files to your $WORK directory and reads from them.
## Matlab tends to not remove these files nor to clean up the
## directory when simulations finish. Matlab expects you to do
## this yourself. If you do not, more and more files accumulate
## in this directory with every completed Matlab simulation.
## Unfortunately - and there lies the performance problem -,
## every time you are using the Parallel Computing toolbox in
## later simulations, Matlab checks the contents of all files in
## /work/<user>/.matlab/local_cluster_jobs once and then monitors
## those files for changes. By asking the metadata servers of the
## parallel file system whether the file has changed (technically
## 'stat <filename>' is called internally).  Dozens of times per
## second. If several Matlab simulations are running concurrently
## and the /work/<user>/.matlab/local_cluster_jobs contains a few
## thousand files already, the metadata servers of the parallel
## file system get tens of thousand 'stat' requests per second,
## most of which are pointless because they are for temporary
## files for long-completed simulations.
## Solution: Have Matlab stores these temporary files on a local
## file system, not the parallel file system. Local 'stat'
## commands are faster anyway, simulation is run on a single
## compute node anyway such that synchronisation via the parallel
## file system is overkill.
## (Another solution would be to frequently wipe the entire
## directory ~/.matlab/local_cluster_jobs, but that could only be
## done safely if no Matlab simulation is running any more.
## Automating that is more complicated that redirecting Matlab to
## store the temporary files on a local scratch file system.)
#### # Change default 'prefdir' setting from ~/.matlab/R2022a
```

```
#### export MATLAB_PREFDIR=/scratch/${LOGNAME}/.matlab/R2022a
#### test -d ${MATLAB_PREFDIR} || mkdir -p ${MATLAB_PREFDIR}

## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
cd /work/myusername/tmp
module purge
module load matlab/r2022b

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

## Uncomment this block, too, if you are relying on MathWorks
## Parallel Computing toolbox.
#### # Check whether other Matlab instances are running on the
#### # same compute node (started from concurrently running Slurm
#### # jobs). If not, remove the directory Matlab used internally
#### # to synchronise parallel workers. Matlab tends to not clean
#### # up this directory itself (see https://www.mathworks.com/
#### # help/parallel-computing/run-code-on-parallel-pools.html)
#### # How do we check for other Slurm jobs of our own? By
#### # counting the number of processes that invoke a Slurm job
#### # script. If it is more than one, other Slurm jobs of ours
#### # are running on the very same compute node, too.
#### if test "$( ps --user ${LOGNAME} --no-headers -f | grep -c
   ↪ '/var/spool/slurm/d/job.*/slurm_script' )" -eq 1; then
####    echo "Deleting ${MATLAB_PREFDIR}"
####    rm -rf ${MATLAB_PREFDIR}
#### fi
```

Listing 4.13: 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 4.14: 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 4.15: Contents of file 'EstimatePi.m'

4.5.10 Common software example: ORCA

4.5.10.1 On a single compute node

In order to run ORCA[71] you may find the following Slurm job script helpful. It asks for 1 compute node with 20 cores for exclusive usage, i.e. no concurrent Slurm job can consume RAM, CPU or local hard disk drive ressources, as much main memory as available for 2 days. ORCA will use the local hard disk partition `/scratch` for temporary and result files. The Slurm job script takes care that upon simulation end the results are transferred to the parallel file system; it also remove files written to `/scratch` in case the job is cancelled, e.g. due to have reached the requested time limit.

[71] https://en.wikipedia.org/wiki/ORCA_(quantum_chemistry_program)

```
#!/bin/bash -l

#SBATCH --partition=long
#SBATCH --time=2-0:00:00
#SBATCH --nodes=1 --tasks-per-node=20 --cpus-per-task=1
#SBATCH --exclusive
# Let Slurm know that you want the entire main memory, every
# megabyte not needed for the operating system itself.
#SBATCH --mem=0
#SBATCH --job-name=orca-simulation-123
#SBATCH --output=%x-%j.out    # write standard output to a file
                              # named after job name given above
                              # and job ID assigned by Slurm.
#SBATCH --error=%x-%j.err     # write error messages a file
                              # named after job name given above
                              # and job ID assigned by Slurm
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de

# 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


## Change to the directory where your ORCA input files
## are stored.
## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
cd /work/myusername/orca/task123

## Input file for ORCA simulation
## PLEASE CHANGE orca.inp TO THE ACTUAL INPUT FILE NAME!
INPUT_FILE=orca.inp

## Store the simulation results to this very directory
WORK_DIRECTORY_TO_STORE_RESULTS=$PWD

## ORCA version to use. See 'module avail orca' for available
## versions
module purge
module load orca/4.2.1-shared

## ORCA writes quite sizeable files and reads from these
## temporary files. In order to speed up this I/O that glues
## together the individual programs an ORCA simulation run will
## subsequently and automatically start point ORCA to a local file
## system (as opposed to, e.g., the parallel file system of LiDO3)
## for these temporary files.
## This has also a run time benefit of approximately 15 \% for,
## e.g., the component ORCA-CIS/TD-DFT.
LOCAL_COMPUTE_DIRECTORY=/scratch/${SLURM_JOB_ID}
mkdir -p ${LOCAL_COMPUTE_DIRECTORY}

## If your job prematurely terminates (i.e. wall time exceeded,
## job cancellation via 'scancel'), temporary and result files
    ↪ ORCA
## created will be left in the /scratch directory.
## This has led multiple times in the past to a situation where
## the local /scratch directories filled up quicker than the
## automatic cleanup script would remove the orphaned files.
##
## To save off your data and clean up the /scratch when a job
## aborts, a trap function can be used. It is called once the
## login shell intercepts certain system signal (here: either one
## of the signals USR1, SIGQUIT, SIGINT, SIGABRT, SIGKILL, SIGHUP,
## SIGTERM, EXIT, TERM).
finalize_job()
{
  echo  "function finalize_job called at `date`"
  rm --force *.tmp
  cp --archive --update --verbose \
      ${LOCAL_COMPUTE_DIRECTORY}/* \
      ${WORK_DIRECTORY_TO_STORE_RESULTS}
  rm --recursive --force ${LOCAL_COMPUTE_DIRECTORY}
}
trap -- 'finalize_job' USR1 SIGQUIT SIGINT SIGABRT SIGKILL \
                       SIGHUP SIGTERM EXIT TERM

## Transfer ORCA input file(s) to the local hard disk drive
cp --archive --update --verbose \
      ${INPUT_FILE} \
      ${LOCAL_COMPUTE_DIRECTORY}

## start your ORCA calculation
## NOTES:
## * ORCA prefers to get invoked by its absolute path. Hence
##   the prefixing of the 'which' command.
## * The '&' at the end of the line that invokes ORCA and the
##   'wait' command are mandatory for this to work. The shell
##   needs to intercept the system signals; if ORCA runs in
##   foreground, the 'trap' will get seemingly ignored.
cd ${LOCAL_COMPUTE_DIRECTORY}
$(which --skip-alias orca) ${INPUT_FILE} &
wait

## IMPORTANT: At the end of your job you will need to copy
## everything back from scratch to your original
## work directory. Otherwise, your results will get lost,
## because you can not access the local file system that
## easy and more - and contents in /scratch not belonging to
## a running job any more get cleaned at regular intervalls
## automatically.
finalize_job

## Optional: remove trap definition
trap - USR1 SIGQUIT SIGINT SIGABRT SIGKILL \
       SIGHUP SIGTERM EXIT TERM
```

Listing 4.16: Contents of file 'run_orca_single_node_through_slurm.sh'
A copy is available from path `/cluster/sfw/lido3-examples/orca/run_orca.for_single_compute_node.sh`

Make sure to update your e-mail address, the path to the directory with the ORCA input file(s), the ORCA input file name itself in your copy of the sample Slurm job script in listing 4.16. Possibly, adapt the `module load` statement as well to use another ORCA version. Let us assume you named your Slurm job script copy `orca.slurm`. Then, on one of the gateways, submit this Slurm job script from an arbitrary path in the parallel file system, e.g. the path where your ORCA input files are stored, via

```
cd /work/myusername/orca/task123
sbatch orca.slurm
```

The messages ORCA writes during execution to standard output and standard error will be written to the directory where you invoked `sbatch`.

4.5.10.2 On multiple compute nodes

Please note that in case you intend to run an ORCA simulation that uses more than one compute node, you can not – to our knowledge – benefit from faster I/O of the local hard disk drive of a compute node. Because all involved ORCA processes need access to one common file space; it must not be scattered to several different `/scratch` file spaces on different compute nodes.[72] So, ORCA has to read from and write to the parallel file system directly. As a side effect, the manual cleanup is not required any more. This simplifies the Slurm job script, to be run on 3 compute nodes, to:

```
#!/bin/bash -l

#SBATCH --partition=long
#SBATCH --time=2-0:00:00
#SBATCH --nodes=3 --tasks-per-node=20 --cpus-per-task=1
#SBATCH --exclusive
# Let Slurm know that you want the entire main memory, every
# megabyte not needed for the operating system itself.
#SBATCH --mem=0
#SBATCH --job-name=orca-simulation-123
#SBATCH --output=%x-%j.out    # write standard output to a file
                              # named after job name given above
                              # and job ID assigned by Slurm.
#SBATCH --error=%x-%j.err     # write error messages a file
                              # named after job name given above
                              # and job ID assigned by Slurm
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de


## Change to the directory where your ORCA input files
## are stored.
## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
cd /work/myusername/orca/task123

## Input file for ORCA simulation
## PLEASE CHANGE orca.inp TO THE ACTUAL INPUT FILE NAME!
INPUT_FILE=orca.inp

## ORCA version to use. See 'module avail orca' for available
## versions
module load orca/4.2.1-shared

## start your ORCA calculation; ORCA's own (parallel) launcher
## will take all needed information from the scheduler environment
```

[72]We are investigating options to have Slurm automatically create an on-demand file system that transparently combines the `/scratch` file spaces of several compute nodes into a single one. The advantage of such a solution is clear; any drawbacks, side effects as well as the costs (license fees etc.) of such a solution remain to be clarified.

```
orca ${INPUT_FILE}
```

Listing 4.17: Contents of file 'run_orca_multiple_nodes_through_slurm.sh'
A copy is available from path /cluster/sfw/lido3-examples/orca/run_orca.for_multiple_compute_nodes.sh

4.5.11 Common software example: Python

A user can install additional Python packages and store them in his home directory, preferably though – for better performance – in the parallel file system under /work (also accessible via the environment variable $WORK). Section 4.3.6.2 explains the procedure.

Depending on whether you installed Python packages via conda, you may need to include the statement

```
conda activate
    ↪ <name_of_a_particular_conda_env_you_created_previously>
```

in either your startup file for all shells, ${HOME}/.bashrc, or include the statement in every Slurm job script. The latter approach is recommended as it makes reproducing - for you and anyone trying to trace your steps at a later time - the results of a particular Slurm job easier: all dependencies are in one place and not scattered over several files (the file ${HOME}/.bashrc, the environment variables that were set when you submitted the Slurm job and the Slurm job script itself).

4.5.11.1 On a single compute node

The following script, when invoked via

```
 sbatch StartPythonBatchJobViaSLURM.sh
```

asks for 1 compute node with 4 cores for 90 minutes and 15 seconds:

```
#!/bin/bash -l
#SBATCH --job-name=PythonAnalysis
# 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=4
```

```
# memory requirement for the entire job in megabytes
#SBATCH --mem=6000
## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
#SBATCH --output=/work/myusername/tmp/slurm_job
#SBATCH --error=/work/myusername/tmp/slurm_job
# send mail when for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL
## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de


# Change directory to where my Python input files are stored
## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
cd /work/myusername/tmp

# Load the required environment modules to use Python
# (note that several Python versions are available, not just
#  3.11.7 as in this example, see output of the command
#  'module avail python/' in an interactive shell to list them.)
module purge
module load python/3.11.7-gcc114-base

# Run the Python script
srun python3 my_script.py
```

Listing 4.18: Contents of file 'StartPythonBatchJobViaSLURM.sh

4.5.11.2 On multiple compute nodes

As of yet, there is no in-built support in Python for parallelising scripts over the compute nodes of a cluster, e.g. distributed parallel programming. In order to have a Slurm job script run Python processes on multiple compute nodes, you need to look into third-party Python modules like, for instance, MPI4Py.

If you think we are wrong on this, feel free to correct us.

4.5.11.3 Multithreading

Python tutorials traditionally tell you to use `multiprocessing.cpu_count()` to determine the number of compute cores and then to use this number to start as many threads. These tutorials, however, assume you have exclusive access to all CPU resources on a given system. This is not necessarily the case when you work on a compute cluster: Cluster job scheduling systems such as SLURM may limit the number of cores **available** to your Python job – as opposed to the number of physically installed cores in a compute node. If you only requested a subset of the physical cores available on a compute node (see column 'max. cores' in table 4.4 on page 102) and you use `multiprocessing.cpu_count()` to determine how many threads to start, your script might try to use way more cores than your Slurm job has available. This may lead to overload and timeouts. Example: say you submit 16 Slurm jobs, each requesting 4 cores, and request it to be run in any of the partitions `short`, `ext_vwl_norm`, `ext_iom_norm`, `ext_chem_norm`, `ext_chem2_norm`, `ext_math_norm`. 4 of these Slurm jobs could get scheduled to compute nodes with 20 physical cores; if each Slurm job starts 20 instead of 4 threads, the compute node will be overloaded 4 times. All 16 of these Slurm jobs could get scheduled to compute nodes with 64 physical cores; if each Slurm job starts 64 threads, the compute node will be overloaded 16 times.

So, do not rely on `multiprocessing.cpu_count()`, but instead on something like the function `available_cpu_count` from listing 4.19 to reliably determine the number of **available** cores:

```
#!/usr/bin/env python3
#
# Cluster job scheduling systems such as SLURM, Platform
# LSF, Altair PBS Professional may limit the number of cores
# _available_ to your Python job - as opposed to the number
# of physically installed cores in a compute node.
#
# If you use multiprocessing.cpu_count() to determine how
# many threads to start, your script might try to use way
# more cores than it has available, which may lead to
# overload and timeouts.

# Solution based on https://stackoverflow.com/a/55423170
import os

def available_cpu_count():
    """ Number of *available* virtual or physical CPUs
        on this system """


    # Tested with Python 3.3 - 3.13 on Linux
    try:
        res = len(os.sched_getaffinity(0))

        if res > 0:
            return res
    except (KeyError, ValueError):
        pass
    # Works with Python 2.x and 3.x on Linux,
    # relies on SLURM environment variable
    try:
        res = int(os.environ['SLURM_CPUS_PER_TASK'])

        if res > 0:
            return res
    except (KeyError, ValueError):
        pass


    # Works with Python 3.13
    try:
        res = os.process_cpu_count()

        if res:
            return res
    except (KeyError, ValueError):
        pass


    raise Exception('Can not determine number of CPUs on this
   ↪ system')


if __name__ == "__main__":
    print("#Cores: ", available_cpu_count())
```

Listing 4.19: Determine number of available cores to Python code

4.5.12 Common software example: R

4.5.12.1 Using own R packages

A user can build additional R packages and store them in his home directory, preferably though – for better performance – in the parallel file system under `/work` (also accessible via the environment variable `${WORK}`). That is the preferred way over having to install them systemwide: When users install their R packages individually, conflict situations are avoided where user A needs a set of R packages in a certain version and user B needing them in older or newer versions. Installing them to a user's home directory – or to some location below `${WORK}` – allows the user to quickly check whether up- or downgrading R packages resolves issues he is having with them.

Given that the home directory is writable only on both gateway servers, a user should

- either only build R packages on the gateway servers, not the compute nodes, if the resulting libraries should be written to the home directory:

  ```
      module load R/4.2.1-gcc122-base
      R CMD INSTALL --library=~/R/4.2.1
     ↪ --configure-args=--with-mpi=${OMPI_HOME}
     ↪ /path/to/Rpackage.tar.gz
  ```

or

- compile on gateway servers or compute nodes while pointing the R library path to somewhere below `${WORK}`

  ```
      module load R/4.2.1-gcc122-base
      R CMD INSTALL --library=${WORK}/R/4.2.1
     ↪ --configure-args=--with-mpi=${OMPI_HOME}
     ↪ /path/to/Rpackage.tar.gz
  ```

The latter approach has proven to work best and is hence recommended.

⚠ The problem with the former approach is as follows: Experience has shown that there exists a race condition on LiDO3 when several Slurm job scripts of the very same user are started simultaneously and all try to access R packages from a user's home directory. While R in some of these Slurm jobs will successfully locate an R package named `foo` additionally installed under `/home/<user>/R/4.2.1`, R might also bail out with

```
Error in loadNamespace(x) : there is no package called 'foo'
```

Reason behind seems a timing issue related to the fact that the servers providing the user's home directory are less performant than those BeeGFS servers providing the parallel file system under `/work`.

When installing, though, additional R packages in a non-standard path like `/work/<user>/R/4.2.1`, R needs a helping hand to locate them: Either set the environment variable `R_LIBS` to the location where the additional R packages got installed

```
export R_LIBS=${WORK}/R/4.2.1
```

Add this line to either every Slurm job script or to the file `${HOME}/.bashrc` in order to set this permanently.

Or create a file named `.Rprofile` in your home directory, `${HOME}` with the following content[73]

```
.libPaths("/work/<user>/R/4.2.1")
```

and, subsequently, create said directory as well:

```
$ mkdir -p /work/<user>/R/4.2.1
```

Do not forget to supply this path when compiling and installing R packages (see next section).

Of couse, if the R packages got installed into a different path like

`/work/<user>/R/x86_64-pc-linux-gnu-library/4.1` adapt the paths in the above listings accordingly. (See section 4.5.12.3 for a smarter solution that allows to switch quickly between different R versions along with their according additional privately installed R packages.)

Use the follower one-liner to check from the command line whether the paths were set correctly such that R successfully locates the additional R packages:

```
Rscript -e 'ip = as.data.frame(installed.packages()[,c(1,3:4)]);
    ↪ ip = ip[is.na(ip$Priority),1:2,drop=FALSE]; ip;'
```

4.5.12.2 Installing own R packages

In this section, we assume you set up R to find additional R package in `${WORK}/R/4.2.1`. If you are using a different path, make sure to replace `${WORK}/R/4.2.1` in the following examples appropriately.

[73]See also an improved version in section 4.5.12.3 that supports switching between multiple R versions.

Most R packages are under constant and heavy development. If you want to download the most recent version from CRAN, compile and install it – including all its dependencies and, in turn, their dependencies – the easiest way known to us so far is:

```
module load R/4.2.1-gcc122-base
R -e "install.packages('package',
    ↪ repos='http://cran.us.r-project.org',
    ↪ lib='${WORK}/R/4.2.1', type='source')"
```

If you want to compile an older version of a specific R package or the R package in question requires some additional configure arguments, download the tarball manually and install it like this

```
module load R/4.2.1-gcc122-base
R CMD INSTALL --library=${WORK}/R/4.2.1
    ↪ --configure-args=--with-mpi=${OMPI_HOME}
    ↪ /path/to/Rpackage.tar.gz
```

The drawback for this approach is that any dependency `Rpackage.tar.gz` might require is not automatically downloaded, compiled and installed. The procedure to compile all dependencies can become rather tedious. But it can be semi-automated: The following convenience script facilitates downloading R packages, especially if you need to install more than one:

```
#!/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
```

Listing 4.20: Contents of file 'load_r_package'

Copy this content to a new file named `load_r_package` and set the executable bit for the script via

```
chmod 755 load_r_package
```

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_package
```

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 package is found that matches this string. Example:

```
./load_r_package digest
```

will download the most recent version of the R package `digest`, at the time of writing `digest_0.6.33.tar.gz`.

Once this script is made available, we can use it to automatically download and then compile and install an R package via

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

package=digest
./load_r_package ${package} && R CMD INSTALL
    ↪ --configure-args=--with-mpi=${OMPI_HOME}
    ↪ ${package}_*.tar.gz || ( echo "ERROR"; read junk )
```

Wrap this in a Bash shell loop to install an entire series of R packages (in the correct order of the respective dependencies):

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

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

Make sure to replace the strings `<version of your liking>` and `<list ↪ of desired packages>` in the instructions above appropriately. In case the R package(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_package ${package} && R CMD
    ↪ INSTALL --configure-args=--with-mpi=${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/myusername/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 package `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 packages to `<list of ↪ desired packages>` 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 package 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 an R package as external system library dependencies. It typically requires a lot of work to compile them as a user in such a way that R will find them both when the R package gets compiled and used later on in simulations. So, if you are faced with a situation that a R package requires some system software, please inform the LiDO team what system software is required. (Better yet, tell the LiDO team additionally what R package you are trying to compile. It might be that more libraries than the one initially reported as missing will need to be installed.)

4.5.12.2.1 Tweaking compiler optimisation flags

Most R packages that invoke C or C++ code compile these source files with generic optimisation flags: `-O2 -g`[74] and use the GNU Compiler Collection (`gcc`, `g++`) to compile the code. If your R simulations run for a large amount of time, you may want to experiment with whether you can decrease the run time of your simulation by tweaking the optimisation flags the GNU compilers uses. But do check that a more aggressive optimisation still yields correct R simulation results. To instruct the GNU C and C++ compiler to use even more optimisations during source code compilation and additionally choose those that it sees fit for the CPU architecture auto-detected at build time, invoke:

```
mkdir ~/.R
cat > ~/.R/Makevars <<EOF
CFLAGS += -O3 -Wall -mtune=native -march=native
CXXFLAGS += -O3 -Wall -mtune=native -march=native
EOF
```

This creates the directory `~/.R` and the file `~/.R/Makevars`. The latter will contain two lines with additional compiler flags, namely `CFLAGS` and `CXXFLAGS`. If you change these settings, you need to re-compile your R packages and assess the results: compare both time-to-solution of your R scripts as well as the simulation results! More aggressive optimisation sometimes leads to wrong simulation results, do to accumulation of small numerical artifacts that accompany more aggressive source code optimisations.

[74] See the GCC man page, `man gcc`, for documentation on the code optimisations being applied with `-O2` and what other optimisation flags are available.

4.5.12.3 Using multiple versions of R along with self-compiled R packages

When building additional R packages yourself, please be aware that R requires that all R packages 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 package with `R/3.6.3-gcc93-base`, but tried to invoke `R` after loading the modulefile `R/4.0.0-gcc93-base` or `R/4.1.2-gcc93-base` some time later. Or vice versa. In these cases where conflicting R versions are 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 4.21: Default contents of file '.Rprofile', problematic when using multiple R versions

use[75]

```
# 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)
```

[75] The following is based on the solution presented in this web discussion.[76]

```
        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 4.22: Contents of file '.Rprofile', compatible with using multiple R versions. A copy is available from path `/cluster/sfw/lido3-examples/R/.Rprofile`

Note that you need to compile additional R packages for every R version you intend to use. Because `libPath` in `~/.Rprofile` points to `/work/<user>/R/<R ↪ version>/`, the compiled R libraries will end up in subdirectories of this custom R library path, not in a subdirectory of the home directory.

4.5.12.4 Example Slurm job script for R

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
## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
#SBATCH --output=/work/myusername/tmp/slurm_job
#SBATCH --error=/work/myusername/tmp/slurm_job
# send mail when for certain job events
# Possible values: NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-type=ALL

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de
```

```
# Change directory to where my R input files are stored
## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
cd /work/myusername/tmp

# Load the required environment modules to use R
# (note that several R versions are available, not just
#  4.2.1 as in this example, see output of the command
#  'module avail R/' in an interactive shell to list them.)
module purge
module load R/4.2.1-gcc122-base

# Uncomment the following block if you want to make
# known to R your own R packages installed (installed
# in ${WORK}/R/4.2.1) *and* you do *not* want to use
# the file ~/.Rprofile (see next section).
### export R_LIBS=/work/myusername/R/4.2.1

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

4.5.13 Common software example: TOPAS Tool for Particle Simulation

4.5.13.1 Running a single TOPAS simulation

The TOPAS[77] software is not available via environment modulefile (see 4.3.5). In order to use the software, a user needs to register with TOPAS MC Inc., download the software pre-compiled for CentOS 7 from their site, typically as a tarball, e.g. `topas_3_9_centos7.tar.gz` and extract it to his `$HOME` or `$WORK` directory:

```
$ cd $WORK
$ tar xvzf topas_3_9_centos7.tar.gz
```

You are possibly going to need to download and extract the Geant4 headers, i.e. the archive `Geant4Headers_1121p1.zip`, too, if you intend to build an application that uses the Geant4 toolkit.

[77] https://sites.google.com/a/topasmc.org/home/code-repository-authorized-users-only

In order to use TOPAS, you are definitely going to need to download some Geant4 datasets[78], e.g.

```
$ cd $WORK
$ test -d G4Data || mkdir G4Data
$ cd G4Data
$ wget https://cern.ch/geant4-data/datasets/G4NDL.4.7.tar.gz
$ tar xzf G4NDL.4.7.tar.gz
$ wget https://cern.ch/geant4-data/datasets/G4INCL.1.2.tar.gz
$ tar xzf G4INCL.1.2.tar.gz
...
```

In order to run TOPAS Monte Carlo simulations using Slurm on LiDO3, a job script as in listing 4.23 theoretically suffices

```
#!/bin/bash -l
#SBATCH --partition=med
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=20
#SBATCH --time=0-03:00:00

# memory requirement for all CPUs in megabytes
#SBATCH --mem=6144
#SBATCH --job-name=TOPAS_simulation

## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
#SBATCH --output=/work/myusername/slurm_job%A
#SBATCH --error=/work/myusername/slurm_job%A

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de
#SBATCH --mail-type=TIME_LIMIT

export TOPAS_G4_DATA_DIR=${WORK}/G4Data
cd /path/to/your/main/topas-input-file
${WORK}/topas/bin/topas ${TOPAS_INPUT_FILE}
```

Listing 4.23: Contents of simple Slurm job script to run TOPAS
(this template is not recommended due to unnecessarily long runtimes!)

The problem with this simple approach is twofold: on the one hand, the prebuild binary Jonatahn Perl from the TOPAS developer group precompiles for CentOS7 is not optimised for the hardware of LiDO3 compute nodes. The LiDO3 team provides

[78] https://geant4.web.cern.ch/download/

two TOPAS binaries optimised for compute nodes of type `cstd01-*`, `cstd02-*`, `cquad01-*` and `cquad02-*`; they merely differ in the amount of additional aggressive compiler optimisation flags. The two binaries are

```
/cluster/sfw/topas/3.9/hw-opt
/cluster/sfw/topas/3.9/aggressive-opt
```

On the other hard there is the I/O pattern of TOPAS: It queries thousands of files from the extracted Geant4 datasets. On average, a running TOPAS simulations implicitly queries the parallel file system every 2 microseconds about the metadata of a Geant4 dataset file. To access its content, but also to verify it has changed and needs to be re-read. When several dozen or even hundreds of TOPAS simulations are being run, a lot of network I/O happens in the background to access Geant4 dataset files. A parallel file system, though, is optimised for throughput, ideally reading from and writing to a handful of large files. That is not what happens with Geant4 dataset files: A typical TOPAS user downloads all 12 Geant datatasets[79] and will use them in his Monte Carlo simulations. Extracting the tarballs yields over 45,000 small files. Accessing them, regardless whether they are stored in a user's home directory (provided by NFS) or in the parallel file system (provided by BeeGFS), over the network one by one yields a bad performance. The simplest way to observe and verify the performance penalty yourself is probably the following experiment: unpacking all 12 Geant4 dataset tarballs and copying these 45,000+ files from `$WORK` to the local hard disk drive (see section 4.3.2.4) of a LiDO3 compute node as part of a Slurm job takes approximately 17 minutes. Copying instead the 12 Geant4 database tarballs (not the extracted files) from `$WORK` to a local hard disk drive and extracting the 45,000+ files from these tarballs onto the local hard disk drive finishes within less than 45 seconds. When not merely one Slurm job performs this experiment, but, e.g., 20 Slurm jobs, the ratio gets worse: 45 minutes for the former approach versus still 45 seconds for the latter approach. In other words, having several dozen or more processes access the 45,000+ Geant4 database files over the network – whether this access be as part of running TOPAS simulations or by simple copy instructions – poses a several performance bottleneck.

An obvious runtime optimisation for TOPAS Slurm jobs is to have the Slurm job script transfer all tarballs from whereever you chose to store them to the local hard disk drive of the compute node Slurm assigned to your job and have the Slurm job script extract these tarballs there:

```
# Customise these three environment variables
TOPAS_G4_DATA_DIR=/work/myusername/G4Data
```

[79] https://geant4.web.cern.ch/download/

```
LOCAL_BASE_DIRECTORY=/scratch/${USER}/${SLURM_JOBID}
LOCAL_TOPAS_G4_DATA_DIR=${LOCAL_BASE_DIRECTORY}/G4Data

# Generic instructions from here onwards
# Extract Geant4 dataset tarballs to local disk
mkdir -p ${LOCAL_TOPAS_G4_DATA_DIR}
for file in ${TOPAS_G4_DATA_DIR}/*.tar.gz; do
    tar -x${TAR_FLAG}zf ${file} -C ${LOCAL_TOPAS_G4_DATA_DIR}
done
```

and adjust the environment variable TOPAS uses to locate the Geant4 database files

```
export TOPAS_G4_DATA_DIR=${LOCAL_TOPAS_G4_DATA_DIR}
```

While at it, additional, though small, TOPAS runtime gains can be achieved by three additional steps:

1. Copy the TOPAS input file and any files it (recursively) may include and may repeatedly read from (think: phase space files).
2. Pin the TOPAS threads to the compute cores the Slurm job got assigned. See https://youtu.be/PSJKNQaqwB0[80] for an introduction to LIKWID, the open source software we used for pinning.
3. On a ccNUMA architecture, the default memory placement is a 'first touch policy placement'. A better, though not optimal strategy is to place memory round-robin across all compute cores. See this PDF document[81] for an in-depth explanation of all this.
4. Automatically adjust the main TOPAS input file to make sure that only as many TOPAS threads are started as got requested by the Slurm option `--cpus-per-task`.

Add an automatic cleanup procedure to the mix that ensures that the files the Slurm job script copied to the local hard disk drives get wiped whenever the Slurm jobs ends or aborts and you end up with the more complex Slurm job script in listing 4.24. It has a header section where you specify where you installed TOPAS and the Geant4 database tarballs to as well as the location of the TOPAS input file. The complex lower part of the Slurm job script aims to be generic and typically does not

[80] thisvideo
[81] https://moodle.nhr.fau.de/pluginfile.php/5823/mod_resource/content/2/04_ccNUMA.pdf

need to be touched (nor understood in all detail). A copy of the Slurm job template to run TOPAS as shown in listing 4.24 is available on LiDO3 from the path `/cluster/sfw/lido3-examples/ topas/slurmjob-single-simulation.sh`.

```
#!/bin/bash -l
#SBATCH --partition=med
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=20
#SBATCH --time=0-03:00:00

# memory requirement for all CPUs in megabytes
#SBATCH --mem=6144
#SBATCH --job-name=TOPAS_simulation

## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
#SBATCH --output=/work/myusername/slurm_job%A
#SBATCH --error=/work/myusername/slurm_job%A

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de
#SBATCH --mail-type=TIME_LIMIT

# 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


### Start customisation section
# (Note: no whitespace allowed in directory or file names!)

# Location of unpacked TOPAS binary
#
# Variant 1:
# Official CentOS 7 TOPAS binary, provided by Jonathan Perl
   ↪ himself,
# downloaded from https://www.topasmc.org/download.
# Advantage   : tested & validated
# Disadvantage: not optimised for LiDO3 hardware, runs slower
TOPAS_BASE_DIRECTORY=/work/${USER}/topas
#
# Variant 2:
# Binary compiled from source by LiDO team using compiler
# optimisation flags for compute nodes cstd0[12]-* and
# cquad0[12]-*
# Advantage    : untested & not yet validated
# Disadvantage: runs faster than prebuild binaries from variant 1
###TOPAS_BASE_DIRECTORY=/cluster/sfw/topas/3.9/hw-opt
#
# Variant 3:
# Binary compiled from source by LiDO team using even more
# aggressive compiler optimisation flags suitable for compute
# nodes cstd0[12]-* and cquad0[12]-*
# Advantage    : untested & not yet validated
# Disadvantage: runs faster than prebuild binaries from variant 1
###TOPAS_BASE_DIRECTORY=/cluster/sfw/topas/3.9/aggressive-opt

TOPAS_INPUT_FILES_DIRECTORY=/work/${USER}/analysis
TOPAS_INPUT_FILE=input.txt
TOPAS_G4_DATA_DIR=/work/${USER}/G4Data

LOCAL_BASE_DIRECTORY=/scratch/${USER}/${SLURM_JOBID}
LOCAL_TOPAS_INPUT_FILES_DIRECTORY=${LOCAL_BASE_DIRECTORY}/input
LOCAL_TOPAS_G4_DATA_DIR=${LOCAL_BASE_DIRECTORY}/G4Data

# Be verbose about the files that are transferred to the local
# hard disk drive of a compute node. Makes debugging easier.
#VERBOSE_FILE_TRANSFER=yes
VERBOSE_FILE_TRANSFER=no
### End customisation section


### Beyond this point, there should be no need to change the
### generic instructions.
COPY_FLAG=""
TAR_FLAG=""
if [ "${VERBOSE_FILE_TRANSFER}" == "y" -o
     "${VERBOSE_FILE_TRANSFER}" == "yes" ]; then
    COPY_FLAG="v"
    TAR_FLAG="v"
fi


###################################################################
## To avoid that unnecessary file I/O over the network (whether
## it be to the home directory $HOME or the parallel file
## system $WORK) copy the TOPAS application, its input files
## and the Géant4 data TOPAS operates on to the local hard disk
## of the assigned compute node. To a temporary location.
## (Note: it probably suffices to just copy the Géant4 data.)
```

```
printf "%s  Start copying files to local hard disk drive on
    ↪ %s\n\n" "$( date )" "$( hostname -s )"


# Extract Geant4 dataset tarballs to local disk
if [ -d "${TOPAS_G4_DATA_DIR}" ]; then
    mkdir -p ${LOCAL_TOPAS_G4_DATA_DIR}
    for file in ${TOPAS_G4_DATA_DIR}/*.tar.gz; do
        tar -x${TAR_FLAG}zf ${file} -C ${LOCAL_TOPAS_G4_DATA_DIR}
    done
    ## 20-200 times slower alternative: copy thousands of files
    ## extracted from the tarball to the local disk
    ### cp -aL${COPY_FLAG} ${TOPAS_G4_DATA_DIR}/*
    ↪ ${LOCAL_TOPAS_G4_DATA_DIR}
else
    if [ -z "${TOPAS_G4_DATA_DIR}" ]; then
        echo "Variable TOPAS_G4_DATA_DIR in Slurm job script is
    ↪ empty. Aborted." 1>&2
        exit 1;
    elif [ ! -d "${TOPAS_G4_DATA_DIR}" ]; then
        echo "Directory <$TOPAS_G4_DATA_DIR>, configured via
    ↪ variable TOPAS_G4_DATA_DIR in Slurm job script, does not
    ↪ exist. Aborted." 1>&2
        exit 2;
    fi
fi

if [ -e ${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE} ]; then
    mkdir -p ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}
    cp -aL${COPY_FLAG}
    ↪ ${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE}
    ↪ ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}
else
    if [ -z "${TOPAS_INPUT_FILES_DIRECTORY}" ]; then
        echo "Variable TOPAS_INPUT_FILES_DIRECTORY in Slurm job
    ↪ script is empty. Aborted." 1>&2
        exit 3;
    elif [ ! -d "${TOPAS_INPUT_FILES_DIRECTORY}" ]; then
        echo "Directory <$TOPAS_INPUT_FILES_DIRECTORY>,
    ↪ configured via variable TOPAS_INPUT_FILES_DIRECTORY in
    ↪ Slurm job script, does not exist. Aborted." 1>&2
        exit 4;
    fi
    if [ -z "${TOPAS_INPUT_FILE}" ]; then
        echo "Variable TOPAS_INPUT_FILE in Slurm job script is
    ↪ empty. Aborted." 1>&2
        exit 5;
    elif [ ! -e
    ↪ "${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE}" ]; then
        echo "TOPAS input file
    ↪ <${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE}>,
    ↪ configured via variables TOPAS_INPUT_FILES_DIRECTORY and
    ↪ TOPAS_INPUT_FILE in Slurm job script, does not exist.
    ↪ Aborted." 1>&2
        exit 6;
    fi
fi

## Copy input files
## Variant 1: Selectively copy all input files referenced in
   ↪ ${TOPAS_INPUT_FILE}
## and all files it (recursively) includes via 'includeFile ='
   ↪ statements.
## Copy also all files and directories referred to in input files
   ↪ using
## the keywords '*InputFile' and '*DicomDirectory'. If you need
   ↪ to extend
## this keyword list and do not know how, contact the LiDO team
   ↪ for help.
## Preserve any directory substructure information.
cnt=1;
maxRecursion=10;
filesToParse=${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE};
inputFilesAndDirs=""
while test -n "${filesToParse}" -a ${cnt} -le ${maxRecursion}; do
    cnt=$(( cnt + 1 ));
    for file in ${filesToParse}; do
        if [ -f "${file}" -o \( -L "${file}" -a -f "$( readlink
    ↪ --canonicalize "${file}" )" \) ]; then
            dirname=$( dirname ${file} );
            inputFilesAndDirs="${inputFilesAndDirs} ${file}";
            # Add the included file(s) to the list of files still
            # to parse. (Only parse files with less than, say,
            # 500 kB. To avoid searching phase space files of
            # several megabytes (or even larger).)
            if [ $( stat --format="%s" "${file}" ) -lt 512000 ];
    ↪ then
                # Some files or directories are explicitly named
                filesToParse=$( echo ${filesToParse}; sed -n -E
    ↪ 's/#.*$//;
    ↪ s/(includeFile|^.*InputFile|^.*DicomDirectory)[[:space:]]*=//p;'
    ↪ ${file} | tr -d ' \r' | xargs --no-run-if-empty -n 1 | sed
    ↪ -e "s|^|${dirname}/|" );
                # For other files only their basename is
                # specified, e.g.
                #   s:So/PhasespacesourcePMMA/PhaseSpaceFileName
    ↪ = "a"
                # will cause TOPAS to look for both "a.header"
                # and "a.phsp". Add both files to list.
                filesToParse=$( echo ${filesToParse}; sed -n -E
    ↪ 's/#.*$//; s/^.*PhaseSpaceFileName[[:space:]]*=//p;'
    ↪ ${file} | tr -d ' \r' | xargs --no-run-if-empty -n 1 | sed
    ↪ -E 's/^(.*)$/\1.header\n\1.phsp/' | sed -e
    ↪ "s|^|${dirname}/|" );
            else
                echo "Warning: Skipped parsing large file
    ↪ <${file}> for include keywords" 1>&2
            fi
        elif [ -d "${file}" ]; then
            inputFilesAndDirs="${inputFilesAndDirs} ${file}";
        fi;
        # Remove just parsed file from list of files
        # still to parse
        filesToParse=$( echo ${filesToParse} | sed -E
    ↪ "s|^${file}\b||;" );
    done;
done;
# Remove ${TOPAS_INPUT_FILES_DIRECTORY} prefix from all files
# to copy in order to avoid that this path prefix gets
# re-created on the local hard disk drive as well.
inputFilesAndDirs=$( echo ${inputFilesAndDirs} | sed -e
    ↪ "s|${TOPAS_INPUT_FILES_DIRECTORY}/||g" )
tar -C ${TOPAS_INPUT_FILES_DIRECTORY} -chf - ${inputFilesAndDirs}
    ↪ | tar -C ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}
    ↪ -x${TAR_FLAG}f -;

## Variant 2: potentially very slow alternative to the complex
## while-for-loop above: brute forcibly copy everything from
## input directory to local HDD
### cp -aL${COPY_FLAG} ${TOPAS_INPUT_FILES_DIRECTORY}/*
    ↪ ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}

printf "%s  Finished copying files to local hard disk drive on
    ↪ %s\n\n" "$( date )" "$( hostname -s )"
##################################################################


##################################################################
## In the case your job prematurely terminates (i.e. wall time
## exceeded or you abort it via 'scancel'), temporary files
## we copied to /scratch will be left in that directory.
## This has led multiple times in the past to a situation where
## the local /scratch directories filled up quicker than the
## automatic cleanup script would remove the orphaned files.
##
## To save off your data and clean up the /scratch when a job
## aborts, a trap function can be used. It is called once the
## login shell intercepts certain system signal (here: either
## one of the signals USR1, SIGQUIT, SIGINT, SIGABRT, SIGKILL,
## SIGHUP, SIGTERM, EXIT, TERM).
##
## The 'SBATCH --signal=B:SIGQUIT@<time>' instruction we added
## to the header of this scripts ensures that Slurm will send
## an abort signal up to <time> seconds before the job reaches
## the configured time limit. This signal is handled here, too.
finalize_job()
{
  printf "%s  Shell function finalize_job invoked\n\n" "$( date )"

  # Check whether copy destination directory is home directory.
  # If so, do not try to salved any data because the home
  # directory is read-only on the compute nodes. Hence, we can
  # not copy files to $HOME.
  # Test this by checking whether ${HOME} is a substring of the
  # variable ${TOPAS_INPUT_FILES_DIRECTORY}. How? By trying to
  # remove the substring and checking the resulting two strings.
  if [ "${TOPAS_INPUT_FILES_DIRECTORY##${HOME}}" !=
    ↪ "${TOPAS_INPUT_FILES_DIRECTORY}" ]; then
      printf "%s  Can not copy any file TOPAS created locally
    ↪ back to ${TOPAS_INPUT_FILES_DIRECTORY}, because destination
    ↪ is read-only on %s\n\n" "$( hostname -s )"
  else
      cp -ru${COPY_FLAG} ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}/*
    ↪ ${TOPAS_INPUT_FILES_DIRECTORY}
  fi
  rm -rf ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}
    ↪ ${LOCAL_TOPAS_G4_DATA_DIR} ${LOCAL_BASE_DIRECTORY}
}
trap -- 'finalize_job' USR1 SIGINT SIGABRT SIGKILL \
                       SIGHUP SIGTERM EXIT TERM

time_limit()
{
  printf "\n\n### TIME LIMIT
    ↪ ######################################\n";
  printf "%s\nSlurm send us signal SIGQUIT:\n";
  printf "This job is about to exceed its configure time";
  printf "limit.\n Salvaging generated data from /scratch to";
  printf "persistent\nstorage.\n" "$( date )"
  printf
    ↪ "##################################################\n\n";
  finalize_job
}
trap -- 'time_limit' SIGQUIT
###################################################################


## Switch to local copy of input files and start TOPAS
## application
export TOPAS_G4_DATA_DIR=${LOCAL_TOPAS_G4_DATA_DIR}
cd ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}

# Tweak input file such that TOPAS will use (at most) as many
    ↪ threads
# as we requested from SLURM with this job script
sed -i -E
    ↪ "s|Ts/NumberOfThreads[[:space:]]*=.*\$|Ts/NumberOfThreads =
    ↪ ${SLURM_CPUS_PER_TASK}|"
    ↪ ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE}

# Dynamically determine the physical numbers of all cores
# Slurm assigned to this Slurm job.
# With a Slurm request that exclusively reserves an entire
# compute node like
#   --cpus-per-task=20 --constraint=[cstd01|cstd02]
# the answer is trivial: 0-19. For a Slurm job that asked
# for an entire quad compute node using
#   --cpus-per-task=48 --constraint=[cquad01|cquad02]
# the answer is trivial, too: 0-47.
# But when asking less than 20 cores on a cstd01/cstd02 compute
# node or less than 48 cores on a cquad01/cquad02 compute node,
# we can not assume 0-<max number requested cpus per task>.

# Unfortunately, the general case of a non-exclusively assigned
# compute node make things complicated for the helper application
# 'likwid-pin'. Say the compute node has 20 physical cores,
# physical numbers 0-19.  You asked for 5. Slurm assigned your
# job the physical cores 4, 5, 11, 12, 18. 'likwid-pin' (up to
# version 5.3.0), sadly, detects that all physical cores and that
# you only got a subset.  In that case, it interprets any core
# numbers as 'logical' numbers, i.e it numbers the physical cores
# 4, 5, 11, 12, 18 as 0, 1, 2, 3, 4. To pin the processes to the
# assigned cores you cannot use
# $ likwid-pin -c N:4,5,11,12,18 topas
# as LIKWID implicitly converts this to
# $ likwid-pin -c L:N:4,5,11,12,18 topas
# but have to use
# $ likwid-pin -c L:N:0,1,2,3,4 topas
# or
# $ likwid-pin -c L:N:0-4 topas
# So, we need to determine those logical numbers of the fly.
ASSIGNED_CORES=$( cnt=0; numactl --show | sed -n -e
    ↪ '/^physcpubind: / { s/^physcpubind: //; p; }' | xargs -n 1
    ↪ | while read core; do echo $cnt; cnt=$(( cnt + 1 )); done |
    ↪ paste -sd, )
ASSIGNED_SOCKETS=$( numactl --show | sed -n -e '/^nodebind: / {
    ↪ s/nodebind: //; s/ /,/g; s/,$//; p; }' )

# Start TOPAS application, but:
# * do pin the application to all those cores of a compute node
#   that Slurm assigned to this job, without touching the code
#   using 'likwid-pin' (see https://youtu.be/PSJKNQaqwB0)
# * place memory round-robin across all assigned compute cores,
#   not an optimal strategy, but oftentimes it leads to better
#   runtimes than the default memory placement called
#   'first touch policy placement' (see https://moodle.nhr.fau.de/
#   pluginfile.php/5823/mod_resource/content/2/04_ccNUMA.pdf)
module load gcc/12.2.0 likwid/5.3.0   # for pinning
numactl --interleave=${ASSIGNED_SOCKETS} \
    likwid-pin -c L:N:${ASSIGNED_CORES} \
    ${TOPAS_BASE_DIRECTORY}/bin/topas ${TOPAS_INPUT_FILE} &
wait

## IMPORTANT: At the end of your job you will need to copy
## everything back from scratch to your original
## work directory. Otherwise, your results will get lost,
## because you can not access the local file system that
## easy and more - and contents in /scratch not belonging to
## a running job any more get cleaned at regular intervalls
## automatically.
## Given that we already have a shell function that takes
## care of this task when the Slurm job aborts, rely on it
## for a TOPAS simulation that ran smoothly, too.
printf "%s  Start salvaging TOPAS output from local hard disk
    ↪ drive to persistent storage\n\n" "$( date )"
finalize_job
printf "%s  Finished salvaging TOPAS output from local hard disk
    ↪ drive to persistent storage\n\n" "$( date )"

## Optional: remove trap definition
trap - USR1 SIGQUIT SIGINT SIGABRT SIGKILL \
       SIGHUP SIGTERM EXIT TERM
```

Listing 4.24: Contents of Slurm job script to efficiently run TOPAS (this template is recommended for an isolated simulation)
A copy is available from path `/cluster/sfw/lido3-examples/topas/slurmjob-single-simulation.sh`

Switching from the simple Slurm job script from listing 4.23 to 4.24, we have seen runtime reductions in a range between 15 and 450 percent for TOPAS simulations running between several minutes up to several days on LiDO3. So, using the complex Slurm job script from listing 4.24 does pay off on LiDO3.

4.5.13.2 Running hundreds of TOPAS simulations

With Monte Carlo simulations, you typically do not run one simulation, but you run hundreds of them. Most likely, you will organize them in some predictable way involving an index variable, either in the directory name for the TOPAS input file or the TOPAS input file name itself or both, e.g.

```
cd /work/myusername/analysis/1PA020_Angle001; topas input.txt
cd /work/myusername/analysis/1PA020_Angle002; topas input.txt
cd /work/myusername/analysis/1PA020_Angle003; topas input.txt
cd /work/myusername/analysis/1PA020_Angle004; topas input.txt
```

or

```
cd /work/myusername/analysis; topas 1PA020_Angle001.txt
cd /work/myusername/analysis; topas 1PA020_Angle002.txt
cd /work/myusername/analysis; topas 1PA020_Angle003.txt
cd /work/myusername/analysis; topas 1PA020_Angle004.txt
```

It would be a waste of time to write a Slurm job script for every single TOPAS simulation. Even writing a script that creates these Slurm job scripts is unnecessary. Instead write a single Slurm job script and use Slurm's job array feature: add the line

```
#SBATCH --array=1-1000
```

to the top of listing 4.24, once submitted with

```
$ sbatch topasslurmjobscript.sh
```

Slurm will start 1,000 instances of this Slurm job script. These Slurm jobs differ only in one respect: Slurm will automatically set the environment variable `SLURM_ARRAY_TASK_ID`, differing between 1 and 1,000. This variable can be used to automatically select the appropriate TOPAS input directory or input file:

```
TOPAS_INPUT_FILES_DIRECTORY=$( printf
    ↪ "/work/${USER}/analysis%04d" ${SLURM_ARRAY_TASK_ID} )
```

will set `${TOPAS_INPUT_FILES_DIRECTORY}` to

```
/work/${USER}/analysis0001
/work/${USER}/analysis0002
/work/${USER}/analysis0003
...
/work/${USER}/analysis1000
```

Similarly, you can define the TOPAS input file depending on the Slurm array index via either one of the following instructions

```
TOPAS_INPUT_FILE=input${SLURM_ARRAY_TASK_ID}.txt
TOPAS_INPUT_FILE=$(printf "input%05d.txt" ${SLURM_ARRAY_TASK_ID})
```

such that the variable `${TOPAS_INPUT_FILE}` will be set automatically to

```
input1.txt
input2.txt
input3.txt
...
input1000.txt
```

and

```
input00001.txt
input00002.txt
input00003.txt
...
input01000.txt
```

respectively.

Slurm job arrays always start at 1; the maximum job array size on LiDO3 is 1024. What to do if you have more than 1024 input directories or files? Use an offset variable and let the shell do some arithmetics involving the offset variable and the Slurm array index:

```
...
#SBATCH --array=1-250
...
OFFSET=2000
TOPAS_INPUT_FILE=input$((${SLURM_ARRAY_TASK_ID}+${OFFSET})).txt
```

will set the variable `${TOPAS_INPUT_FILE}` automatically to

```
input2001.txt
input2002.txt
input2003.txt
...
input2250.txt
```

A Slurm job script to be start TOPAS as Slurm job arry is shown in the following listing 4.25; a copy of this template is also available on LiDO3 from the path `/cluster/sfw/lido3-examples/topas/ slurmarrayjob.sh`.

```
#!/bin/bash -l
#SBATCH --partition=med
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=20
#SBATCH --time=0-03:00:00

# memory requirement for all CPUs in megabytes
#SBATCH --mem=6144
#SBATCH --job-name=TOPAS_simulation

## PLEASE CHANGE myusername TO YOUR ACTUAL USERNAME!
#SBATCH --output=/work/myusername/slurm_job%A_%a
#SBATCH --error=/work/myusername/slurm_job%A_%a

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@tu-dortmund.de
#SBATCH --mail-type=TIME_LIMIT

# Start 1000 instances of this job script
# (Note that max value is 1024. Submit multiple Slurm job arrays
#  and adjust variable SLURM_JOB_ARRAY_INDEX_TO_FILE_INDEX_OFFSET
#  accordingly for input files that have a higher index.)
#SBATCH --array=1-1000

# 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


### Start customisation section
# (Note: no whitespace allowed in directory or file names!)
SLURM_JOB_ARRAY_INDEX_TO_FILE_INDEX_OFFSET=0

# Location of unpacked TOPAS binary
#
# Variant 1:
# Official CentOS 7 TOPAS binary, provided by Jonathan Perl
   ↪ himself,
# downloaded from https://www.topasmc.org/download.
# Advantage   : tested & validated
# Disadvantage: not optimised for LiDO3 hardware, runs slower
TOPAS_BASE_DIRECTORY=/work/${USER}/topas
#
# Variant 2:
# Binary compiled from source by LiDO team using compiler
# optimisation flags for compute nodes cstd0[12]-* and
# cquad0[12]-*
# Advantage   : untested & not yet validated
# Disadvantage: runs faster than prebuild binaries from variant 1
###TOPAS_BASE_DIRECTORY=/cluster/sfw/topas/3.9/hw-opt
#
# Variant 3:
# Binary compiled from source by LiDO team using even more
# aggressive compiler optimisation flags suitable for compute
# nodes cstd0[12]-* and cquad0[12]-*
# Advantage   : untested & not yet validated
# Disadvantage: runs faster than prebuild binaries from variant 1
###TOPAS_BASE_DIRECTORY=/cluster/sfw/topas/3.9/aggressive-opt

# If all your TOPAS input files are stored in a single directory,
# use
TOPAS_INPUT_FILES_DIRECTORY=/work/${USER}/analysis
# If, instead, your input directory names are zero-padded,
# e.g. named along the lines of
#    analysis0001/, analysis0002/, analysis0003/, ...
# comment out the previous line and uncomment and adapt the
# following line. (The magic lies in 'printf' and the
# argument '%04d' which turns non-zero padded numbers
# into numbers with up to 3 leading zeros which you try with
# $ 'printf "%04d\n" 1'
## TOPAS_INPUT_FILES_DIRECTORY=$( printf
   ↪ "/work/${USER}/analysis%04d" $(( ${SLURM_ARRAY_TASK_ID} +
   ↪ ${SLURM_JOB_ARRAY_INDEX_TO_FILE_INDEX_OFFSET} )) )
#
# If, instead, you just want to pick the n-th subdirectory in a
# directory with many subdirectories, each of which containing a
# single set of TOPAS input files, you can use the automatically
# assigned Slurm job array task ID (and possibly an offset to work
# around the limit of at most 1024 entries in a Slurm job array)
# to be that number 'n'.
##TOPAS_INPUT_FILES_DIRECTORY=$( /bin/ls -d
   ↪ /path/to/my/topas/input/directories/topasinput* | awk "{ if
   ↪ (NR == $(( ${SLURM_ARRAY_TASK_ID} +
   ↪ ${SLURM_JOB_ARRAY_INDEX_TO_FILE_INDEX_OFFSET} )) ) { print;
   ↪ }}" )

# If your TOPAS input files are incrementally numbered, use
TOPAS_INPUT_FILE=input$(( ${SLURM_ARRAY_TASK_ID} +
   ↪ ${SLURM_JOB_ARRAY_INDEX_TO_FILE_INDEX_OFFSET} )).txt
#
# If, instead, you use zero-padded numbers in your TOPAS
# input files names uncomment and adapt the following line
## TOPAS_INPUT_FILE=$( printf "input%05d.txt" $((
   ↪ ${SLURM_ARRAY_TASK_ID} +
   ↪ ${SLURM_JOB_ARRAY_INDEX_TO_FILE_INDEX_OFFSET} )) )
#

TOPAS_G4_DATA_DIR=/work/${USER}/G4Data

LOCAL_BASE_DIRECTORY=/scratch/${USER}/${SLURM_JOBID}_${SLURM_ARRAY_TASK_ID}
LOCAL_TOPAS_INPUT_FILES_DIRECTORY=${LOCAL_BASE_DIRECTORY}/input
LOCAL_TOPAS_G4_DATA_DIR=${LOCAL_BASE_DIRECTORY}/G4Data

# Be verbose about the files that are transferred to the local
# hard disk drive of a compute node. Makes debugging easier.
#VERBOSE_FILE_TRANSFER=yes
VERBOSE_FILE_TRANSFER=no
### End customisation section


### Beyond this point, there should be no need to change the
### generic instructions.
COPY_FLAG=""
TAR_FLAG=""
if [ "${VERBOSE_FILE_TRANSFER}" == "y" -o
     "${VERBOSE_FILE_TRANSFER}" == "yes" ]; then
    COPY_FLAG="v"
    TAR_FLAG="v"
fi


###################################################################
## To avoid that unnecessary file I/O over the network (whether
## it be to the home directory $HOME or the parallel file
## system $WORK) copy the TOPAS application, its input files
## and the Géant4 data TOPAS operates on to the local hard disk
## of the assigned compute node. To a temporary location.
## (Note: it probably suffices to just copy the Géant4 data.)
printf "%s  Start copying files to local hard disk drive on
   ↪ %s\n\n" "$( date )" "$( hostname -s )"


# Extract Geant4 dataset tarballs to local disk
if [ -d "${TOPAS_G4_DATA_DIR}" ]; then
    mkdir -p ${LOCAL_TOPAS_G4_DATA_DIR}
    for file in ${TOPAS_G4_DATA_DIR}/*.tar.gz; do
        tar -x${TAR_FLAG}zf ${file} -C ${LOCAL_TOPAS_G4_DATA_DIR}
    done
    ## 20-200 times slower alternative: copy thousands of files
    ## extracted from the tarball to the local disk
    ### cp -aL${COPY_FLAG} ${TOPAS_G4_DATA_DIR}/*
   ↪ ${LOCAL_TOPAS_G4_DATA_DIR}
else
    if [ -z "${TOPAS_G4_DATA_DIR}" ]; then
        echo "Variable TOPAS_G4_DATA_DIR in Slurm job script is
   ↪ empty. Aborted." 1>&2
        exit 1;
    elif [ ! -d "${TOPAS_G4_DATA_DIR}" ]; then
        echo "Directory <$TOPAS_G4_DATA_DIR>, configured via
   ↪ variable TOPAS_G4_DATA_DIR in Slurm job script, does not
   ↪ exist. Aborted." 1>&2
        exit 2;
    fi
fi

if [ -e ${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE} ]; then
    mkdir -p ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}
    cp -aL${COPY_FLAG}
   ↪ ${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE}
   ↪ ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}
else
    if [ -z "${TOPAS_INPUT_FILES_DIRECTORY}" ]; then
        echo "Variable TOPAS_INPUT_FILES_DIRECTORY in Slurm job
   ↪ script is empty. Aborted." 1>&2
        exit 3;
    elif [ ! -d "${TOPAS_INPUT_FILES_DIRECTORY}" ]; then
        echo "Directory <$TOPAS_INPUT_FILES_DIRECTORY>,
    ↪ configured via variable TOPAS_INPUT_FILES_DIRECTORY in
    ↪ Slurm job script, does not exist. Aborted." 1>&2
        exit 4;
    fi
    if [ -z "${TOPAS_INPUT_FILE}" ]; then
        echo "Variable TOPAS_INPUT_FILE in Slurm job script is
    ↪ empty. Aborted." 1>&2
        exit 5;
    elif [ ! -e
    ↪ "${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE}" ]; then
        echo "TOPAS input file
    ↪ <${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE}>,
    ↪ configured via variables TOPAS_INPUT_FILES_DIRECTORY and
    ↪ TOPAS_INPUT_FILE in Slurm job script, does not exist.
    ↪ Aborted." 1>&2
        exit 6;
    fi
fi

## Copy input files
## Variant 1: Selectively copy all input files referenced in
    ↪ ${TOPAS_INPUT_FILE}
## and all files it (recursively) includes via 'includeFile ='
    ↪ statements.
## Copy also all files and directories referred to in input files
    ↪ using
## the keywords '*InputFile' and '*DicomDirectory'. If you need
    ↪ to extend
## this keyword list and do not know how, contact the LiDO team
    ↪ for help.
## Preserve any directory substructure information.
cnt=1;
maxRecursion=10;
filesToParse=${TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE};
inputFilesAndDirs=""
while test -n "${filesToParse}" -a ${cnt} -le ${maxRecursion}; do
    cnt=$(( cnt + 1 ));
    for file in ${filesToParse}; do
        if [ -f "${file}" -o \( -L "${file}" -a -f "$( readlink
    ↪ --canonicalize "${file}" )" \) ]; then
            dirname=$( dirname ${file} );
            inputFilesAndDirs="${inputFilesAndDirs} ${file}";
            # Add the included file(s) to the list of files still
            # to parse. (Only parse files with less than, say,
            # 500 kB. To avoid searching phase space files of
            # several megabytes (or even larger).)
            if [ $( stat --format="%s" "${file}" ) -lt 512000 ];
    ↪ then
                # Some files or directories are explicitly named
                filesToParse=$( echo ${filesToParse}; sed -n -E
    ↪ 's/#.*$//;
    ↪ s/(includeFile|^.*InputFile|^.*DicomDirectory)[[:space:]]*=//p;'
    ↪ ${file} | tr -d ' \r' | xargs --no-run-if-empty -n 1 | sed
    ↪ -e "s|^|${dirname}/|" );
                # For other files only their basename is
                # specified, e.g.
                #   s:So/PhasespacesourcePMMA/PhaseSpaceFileName
    ↪ = "a"
                # will cause TOPAS to look for both "a.header"
                # and "a.phsp". Add both files to list.
                filesToParse=$( echo ${filesToParse}; sed -n -E
    ↪ 's/#.*$//; s/^.*PhaseSpaceFileName[[:space:]]*=//p;'
    ↪ ${file} | tr -d ' \r' | xargs --no-run-if-empty -n 1 | sed
    ↪ -E 's/^(.*)$/\1.header\n\1.phsp/' | sed -e
    ↪ "s|^|${dirname}/|" );
            else
                echo "Warning: Skipped parsing large file
    ↪ <${file}> for include keywords" 1>&2
            fi
        elif [ -d "${file}" ]; then
            inputFilesAndDirs="${inputFilesAndDirs} ${file}";
        fi;
        # Remove just parsed file from list of files
        # still to parse
        filesToParse=$( echo ${filesToParse} | sed -E
    ↪ "s|^${file}\b||;" );
    done;
done;
# Remove ${TOPAS_INPUT_FILES_DIRECTORY} prefix from all files
# to copy in order to avoid that this path prefix gets
# re-created on the local hard disk drive as well.
inputFilesAndDirs=$( echo ${inputFilesAndDirs} | sed -e
    ↪ "s|${TOPAS_INPUT_FILES_DIRECTORY}/||g" )
tar -C ${TOPAS_INPUT_FILES_DIRECTORY} -chf - ${inputFilesAndDirs}
    ↪ | tar -C ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}
    ↪ -x${TAR_FLAG}f -;

## Variant 2: potentially very slow alternative to the complex
## while-for-loop above: brute forcibly copy everything from
## input directory to local HDD
### cp -aL${COPY_FLAG} ${TOPAS_INPUT_FILES_DIRECTORY}/*
    ↪ ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}
```

```
printf "%s  Finished copying files to local hard disk drive on
    ↪ %s\n\n" "$( date )" "$( hostname -s )"
####################################################################


####################################################################
## In the case your job prematurely terminates (i.e. wall time
## exceeded or you abort it via 'scancel'), temporary files
## we copied to /scratch will be left in that directory.
## This has led multiple times in the past to a situation where
## the local /scratch directories filled up quicker than the
## automatic cleanup script would remove the orphaned files.
##
## To save off your data and clean up the /scratch when a job
## aborts, a trap function can be used. It is called once the
## login shell intercepts certain system signal (here: either
## one of the signals USR1, SIGQUIT, SIGINT, SIGABRT, SIGKILL,
## SIGHUP, SIGTERM, EXIT, TERM).
##
## The 'SBATCH --signal=B:SIGQUIT@<time>' instruction we added
## to the header of this scripts ensures that Slurm will send
## an abort signal up to <time> seconds before the job reaches
## the configured time limit. This signal is handled here, too.
finalize_job()
{
  printf "%s  Shell function finalize_job invoked\n\n" "$( date )"

  # Check whether copy destination directory is home directory.
  # If so, do not try to salved any data because the home
  # directory is read-only on the compute nodes. Hence, we can
  # not copy files to $HOME.
  # Test this by checking whether ${HOME} is a substring of the
  # variable ${TOPAS_INPUT_FILES_DIRECTORY}. How? By trying to
  # remove the substring and checking the resulting two strings.
  if [ "${TOPAS_INPUT_FILES_DIRECTORY##${HOME}}" !=
    ↪ "${TOPAS_INPUT_FILES_DIRECTORY}" ]; then
      printf "%s  Can not copy any file TOPAS created locally
    ↪ back to ${TOPAS_INPUT_FILES_DIRECTORY}, because destination
    ↪ is read-only on %s\n\n" "$( hostname -s )"
  else
      cp -ru${COPY_FLAG} ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}/*
    ↪ ${TOPAS_INPUT_FILES_DIRECTORY}
  fi
  rm -rf ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}
    ↪ ${LOCAL_TOPAS_G4_DATA_DIR} ${LOCAL_BASE_DIRECTORY}
}
trap -- 'finalize_job' USR1 SIGINT SIGABRT SIGKILL \
                       SIGHUP SIGTERM EXIT TERM
```

```
time_limit()
{
  printf "\n\n### TIME LIMIT
    ↪ #####################################\n";
  printf "%s\nSlurm send us signal SIGQUIT:\n";
  printf "This job is about to exceed its configure time";
  printf "limit.\n Salvaging generated data from /scratch to";
  printf "persistent\nstorage.\n" "$( date )"
  printf
    ↪ "#####################################################\n\n";
  finalize_job
}
trap -- 'time_limit' SIGQUIT
##################################################################


## Switch to local copy of input files and start TOPAS
## application
export TOPAS_G4_DATA_DIR=${LOCAL_TOPAS_G4_DATA_DIR}
cd ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}

# Tweak input file such that TOPAS will use (at most) as many
    ↪ threads
# as we requested from SLURM with this job script
sed -i -E
    ↪ "s|Ts/NumberOfThreads[[:space:]]*=.*\$|Ts/NumberOfThreads =
    ↪ ${SLURM_CPUS_PER_TASK}|"
    ↪ ${LOCAL_TOPAS_INPUT_FILES_DIRECTORY}/${TOPAS_INPUT_FILE}

# Dynamically determine the physical numbers of all cores
# Slurm assigned to this Slurm job.
# With a Slurm request that exclusively reserves an entire
# compute node like
#   --cpus-per-task=20 --constraint=[cstd01|cstd02]
# the answer is trivial: 0-19. For a Slurm job that asked
# for an entire quad compute node using
#   --cpus-per-task=48 --constraint=[cquad01|cquad02]
# the answer is trivial, too: 0-47.
# But when asking less than 20 cores on a cstd01/cstd02 compute
# node or less than 48 cores on a cquad01/cquad02 compute node,
# we can not assume 0-<max number requested cpus per task>.

# Unfortunately, the general case of a non-exclusively assigned
# compute node make things complicated for the helper application
# 'likwid-pin'. Say the compute node has 20 physical cores,
# physical numbers 0-19.  You asked for 5. Slurm assigned your
# job the physical cores 4, 5, 11, 12, 18. 'likwid-pin' (up to
# version 5.3.0), sadly, detects that all physical cores and that
# you only got a subset.  In that case, it interprets any core
# numbers as 'logical' numbers, i.e it numbers the physical cores
# 4, 5, 11, 12, 18 as 0, 1, 2, 3, 4. To pin the processes to the
# assigned cores you cannot use
# $ likwid-pin -c N:4,5,11,12,18 topas
# as LIKWID implicitly converts this to
# $ likwid-pin -c L:N:4,5,11,12,18 topas
# but have to use
# $ likwid-pin -c L:N:0,1,2,3,4 topas
# or
# $ likwid-pin -c L:N:0-4 topas
# So, we need to determine those logical numbers of the fly.
ASSIGNED_CORES=$( cnt=0; numactl --show | sed -n -e
    ↪ '/^physcpubind: / { s/^physcpubind: //; p; }' | xargs -n 1
    ↪ | while read core; do echo $cnt; cnt=$(( cnt + 1 )); done |
    ↪ paste -sd, )
ASSIGNED_SOCKETS=$( numactl --show | sed -n -e '/^nodebind: / {
    ↪ s/nodebind: //; s/ /,/g; s/,$//; p; }' )

# Start TOPAS application, but:
# * do pin the application to all those cores of a compute node
#   that Slurm assigned to this job, without touching the code
#   using 'likwid-pin' (see https://youtu.be/PSJKNQaqwB0)
# * place memory round-robin across all assigned compute cores,
#   not an optimal strategy, but oftentimes it leads to better
#   runtimes than the default memory placement called
#   'first touch policy placement' (see https://moodle.nhr.fau.de/
#   pluginfile.php/5823/mod_resource/content/2/04_ccNUMA.pdf)
module load gcc/12.2.0 likwid/5.3.0    # for pinning
numactl --interleave=${ASSIGNED_SOCKETS} \
    likwid-pin -c L:N:${ASSIGNED_CORES} \
    ${TOPAS_BASE_DIRECTORY}/bin/topas ${TOPAS_INPUT_FILE} &
wait

## IMPORTANT: At the end of your job you will need to copy
## everything back from scratch to your original
## work directory. Otherwise, your results will get lost,
## because you can not access the local file system that
## easy and more - and contents in /scratch not belonging to
## a running job any more get cleaned at regular intervalls
## automatically.
## Given that we already have a shell function that takes
## care of this task when the Slurm job aborts, rely on it
## for a TOPAS simulation that ran smoothly, too.
printf "%s  Start salvaging TOPAS output from local hard disk
    ↪ drive to persistent storage\n\n" "$( date )"
finalize_job
```

```
printf "%s  Finished salvaging TOPAS output from local hard disk
    ↪ drive to persistent storage\n\n" "$( date )"

## Optional: remove trap definition
trap - USR1 SIGQUIT SIGINT SIGABRT SIGKILL \
       SIGHUP SIGTERM EXIT TERM
```

Listing 4.25: Contents of Slurm job array script to efficiently run hundreds of TOPAS simulation
A copy is available from path `/cluster/sfw/lido3-examples/topas/slurmarrayjob.sh`

4.5.14 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

## PLEASE CHANGE my.name TO YOUR ACTUAL MAIL ADDRESS!
#SBATCH --mail-user=my.name@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
```

4.5.15 Signals and traps

4.5.15.1 Have a job automatically clean up when exceeding requested wall-clock time limit

By default, *Slurm* (up to version 22.05) sends two different signals to a job that exceeded its requested walltime. The first signal is `SIGQUIT` and can be intercepted by a shell or user program, the second `SIGTERM` definitely pulls the plug on all your processes started as part of your Slurm job. The amount of time Slurm waits after the first signal is send before sending the second is controlled by the Slurm configuration parameter `KillWait`. You can use

```
$ scontrol show config | grep KillWait
```

to verify that it is set on LiDO3 to 30 s.

It may, however, be that your Slurm job needs more time than when receiving the first signal `SIGQUIT`. For instance, moving result files from the `/scratch` file system to the parallel file system or cleaning up any remaining temporary files may take longer than these 30 seconds. In these cases, the user needs to set up three things in a 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 shell invoked by your Slurm job file (see the shebang line, e.g. #/bin/bash!, at the beginning of your Slurm job script) 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

## Optional: remove trap definition
trap - SIGQUIT
```

4.5.15.2 Passing Signals to your running application

If your program needs to be informed about unforseen circumstances via pre-defined signals, this can be somewhat hard via the MPI's own wrappers. For example, the OpenMPI 4.0 `mpirun` manpage states

```
When orterun receives a SIGTERM and SIGINT, it will attempt to
kill the entire job by sending all processes
in the job a SIGTERM, waiting a small number of seconds,
then sending all processes in the job a SIGKILL.
SIGUSR1 and SIGUSR2 signals received by orterun are propagated
to all processes in the job.

A SIGTSTOP signal to mpirun will cause a SIGSTOP signal to be
sent to all of the programs started by mpirun and likewise a
SIGCONT signal to mpirun will cause a SIGCONT sent.

Other signals are not currently propagated by orterun.
```

Thus it would not be possible for your application to receive a SIGINT signal via mpirun/orterun.
In this case one has to use `srun -X` to start the application. Let's again cite the manpage from `Slurm`, too:

```
-X, --disable-status
Disable the display of task status when srun receives a single
SIGINT (Ctrl-C). Instead immediately forward the SIGINT to the
running job.
Without this option a second Ctrl-C in one second is required to
forcibly terminate the job and srun will immediately exit.
May also be set via the environment variable SLURM_DISABLE_STATUS.
```

```
This option applies to job allocations.
```

Based on the learnings from the previous section 4.5.15.1 we can now prolongate or map any signal send to our `Slurm` job to the running application.

```
[...]
#SBATCH --signal=B:SIGINT@30
[...]
APP_PID=0
signal_trap() {
  kill -SIGINT $APP_PID
  wait
  exit 0
}
[...]
trap -- 'signal_trap' SIGINT
srun -X my-application &
APP_PID=$!
wait

## Optional: remove trap definition
trap - SIGINT
```

Note, that we need to favor `srun` over `mpirun` to get the signal from the kill command to our actual application.

List of common signals and their corresponding number:

```
1      SIGHUP
2      SIGINT
3      SIGQUIT
6      SIGABRT
9      SIGKILL
14     SIGALRM
15     SIGTERM
```

4.5.15.2.1 Sending arbitrary signals to a Slurm job

If you want to send a signal to your job besides the timeout-triggered signal described above, you can trigger these signals on your own via

```
scontrol -s USR1 JOB_ID
or
scancel --signal=KILL JOB_ID
```

4.5.16 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[82] 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 one task per node needed.
srun --nodes 7 --tasks 7 mkdir -p /scratch/${USER}_${SLURM_JOBID}

# STEP 02:
# No explicit allocation, hence use all 42 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 24 tasks for a not well scaling program.
srun --ntasks 24 --nodes 4 --exclusive \
 reduce_mpi_data < output.txt > result.txt &
```

[82] CPU cores to be more precise.

```
# STEP 04:
# sub-allocate a single node.
# The gzip cannot run on separate nodes to compress output.txt.
# Thanks to the ampersand `&` in the previous srun command, this
# step runs concurrently with the previous step.
OMP_NUM_THREAD=10 srun --ntasks 10 --nodes 1 \
 --exclusive gzip output.txt &

# wait for the steps to finish
wait
```

4.5.17 Example for parallel debugging with TotalView

TotalView [83] is a HPC debugging software for parallel debugging of C/C++, Fortran and mixed-language python applications. It is a available as a module [85].

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

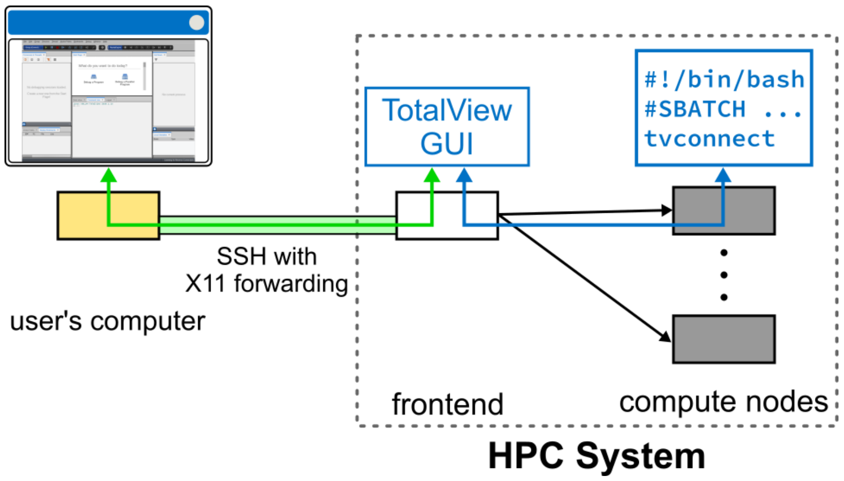


Figure 4.50: TotalView debugging overview

[83]TotalView[84] website
[85]see section *Modules in job scripts* on page 77

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 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 [86]. 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`.

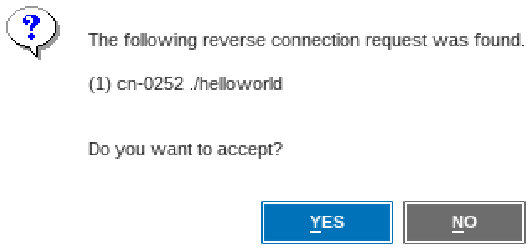
The next step is to preprend the usual `mpirun` or `srun` call in your *Slurm* job script with `tvconnect`.

```
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.

[86] see the documentation[87] for a detailed description


The following reverse connection request was found.

(1) cn-0252 ./helloworld

Do you want to accept?

Figure 4.51: 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

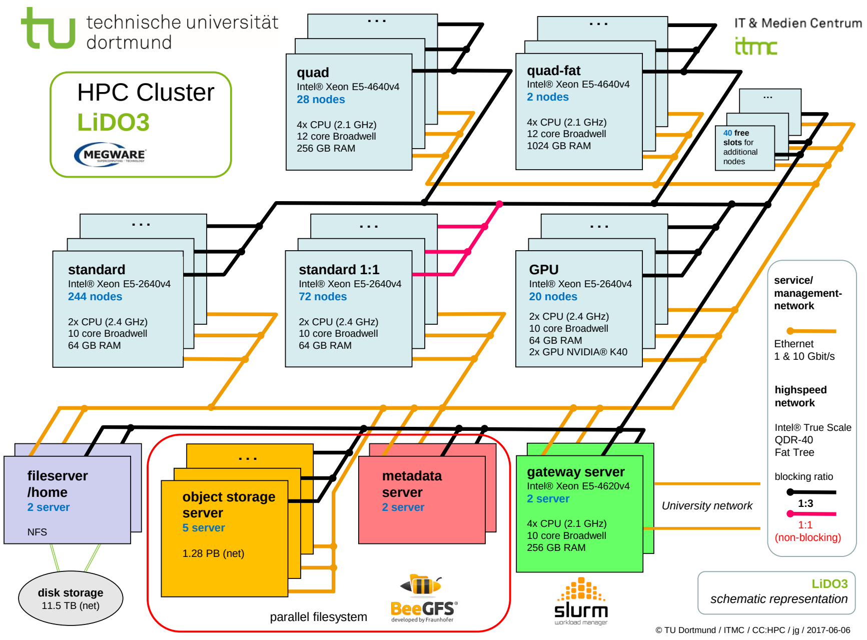


Figure 4.52: Schematic representation of the LiDO3 architecture.

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.

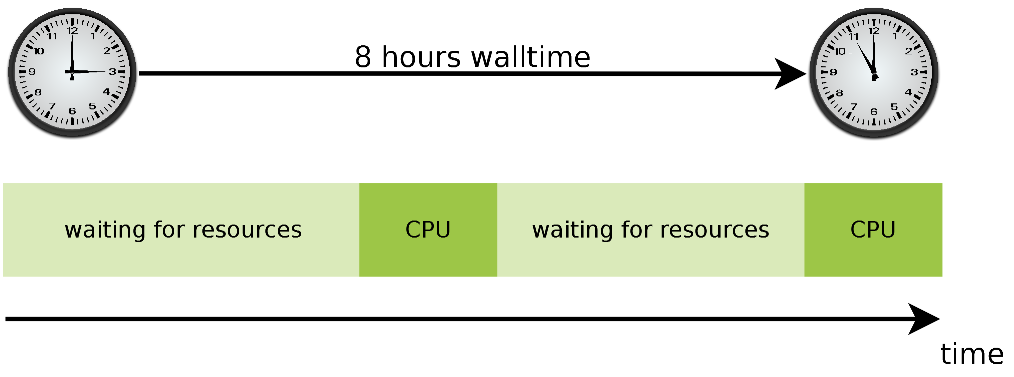


Figure 4.53: A job waiting more than utilizing the CPU uses eight hours walltime total.

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.

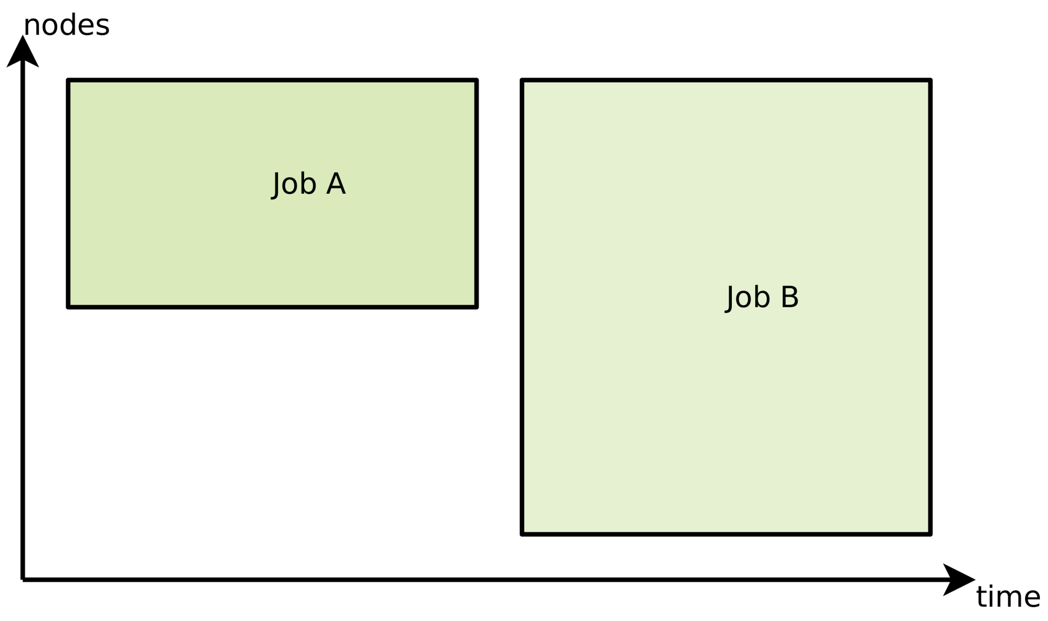


Figure 4.54: *Job B* is waiting for nodes used by *Job A*.

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*.

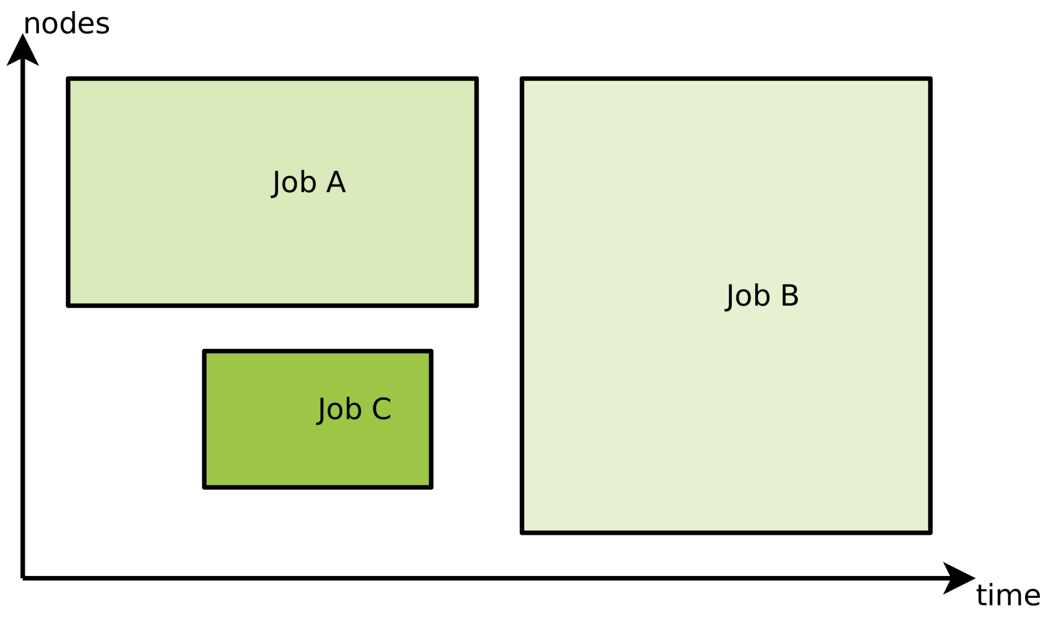


Figure 4.55: *Job C* is started before *Job B* because it will be finished before *Job B* can start.

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

4.8 Get support / contact

For support and further assistance, please write an email to the ITMC service desk[88] (service.itmc@tu-dortmund.de) to open a support ticket.

Acknowledging the use of LiDO3 in publications and informing the LiDO team about these publications is crucial for the funding of this and future HPC machines. Please inform us via email[89] (lido-team.itmc@lists.tu-dortmund.de), if wou wrote any papers using utilizing LiDO3.

Of course you can still contact the LiDO team via email for other purposes if a support ticket does not fit.

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[90]. 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`
```

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

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
```

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
```

4.9.4 Can I have Visual Studio Code on LiDO3?

Short answer: No, we will not install it globally and strongly discourage its use on LiDO3. Because with 200+ active LiDO3 users and 2 gateway servers with 256 GiB of RAM each to serve those 200+ users, Visual Studio Code as of June 2023 has a prohibitively expensive memory footprint.

Long answer:

You probably have already used Visual Studio Code (or short *vscode*) before and like it a lot. So do we, it is a great editor from the user's perspective. But there is a downside from our system administrators' point of view:
vscode is a resource hugging monster. And that is due to the fact that it is built upon the Electron framework[91]. That basically means that the single binary `code` contains a Javascript based web server started in the background running inside a V8 Javascript engine[92] that hosts the application itself - the GUI you interact with is more or less a stripped-down browser. So what you perceive as a simple editor application is another browser installed on your system with the storage footprint a browser nowadays usually has.

[91] https://en.wikipedia.org/wiki/Electron_(software_framework)
[92] https://v8.dev/

On startup *vscode* consumes a lot of memory, which is totally fine as long as it happens on your local workstation or laptop. We have seen instances of *vscode* using 40 GB of reserved RAM when freshly started and with no files open.

If three or more users open *vscode* on one of the LiDO3 gateway servers, this has a significant impact on every other user currently working on that gateway. For that reason we have removed the *vscode* software from the gateway servers.

You might feel inclined to use vscode locally on your workstation with the Visual Studio Code Remote - SSH extension[93] enabled to circumvent that sitation. But, sadly, that leads to the same results: One could think that the extension merely synchronizes the editid files via a SSH connection and all the memory expensive stuff is happening locally on your machine. But no, the authors of the extension felt it was a good idea to start a complete V8 Javascript engine with the full Electron stack on the remote side (which would be one of the LiDO3 gateways in this case) and do all the file parsing, IntelliSense (completions), code navigation, and debugging remotely. So all the memory expensive stuff does not happen on your local machine! That also means that all your extensions that deal with the source code you write, run on the remote SSH host, i.e. the LiDO3 gateways (see section *Managing extensions*)[94].

Some people reported that issue here[95] and here[96], but it was closed without a solution. Nonetheless, we have seen cases where a gateway was completely idle, started to get used by two users, both using Visual Studio Code Remote - and that 256 GiB of RAM were not enough to satisfy the memory needs of both editor instances. As a result, the Visual Studio Code instance of one user crashed with an out-of-memory event.

4.9.5 I can not open X11 programs on one gateway or compute node, but not on others

If you are logged in to the LiDO3 cluster and you get an error message like

```
X11 connection rejected because of wrong authentication
```

when you open a graphical program (like `emacs`, `matlab`, `mathematica`, `paraview` etc.), there is typically a problem with orphaned entries in the file `/work/$USER/.Xauthority`.[97] This file contains authorization codes (so-called *magic cookies*) to allow accessing the

[93] https://code.visualstudio.com/docs/remote/ssh
[94] https://code.visualstudio.com/docs/remote/ssh
[95] https://github.com/microsoft/vscode-remote-release/issues/1110
[96] https://github.com/microsoft/vscode/issues/151205
[97] See the wikipedia page on X window authorization[98] for more details.

X11 display server on the gateway server or compute node while being connected to it over the network from your laptop or workstation. The file is managed by program `xauth`; typically the operating system takes care for you in the background of automatically adding the appropriate magic cookie to this file on login and removing it again on logout. If, however, a login session on the gateways or an interactive Slurm session on a compute node is terminated, magic cookies might not get removed properly. They pile up in the file `/work/$USER/.Xauthority`. Subsequently opened login sessions might get assigned the same display name of one of those orphaned X11 sessions, the new magic cookie will not get stored. X11 programs will try to use the old magic cookie, but authorization will fail and you end up with the error message above.

To resolve this issue, either delete the entire content of `/work/$USER/.Xauthority`, e.g. via

```
$ sed -i -e 'd' /work/$USER/.Xauthority
```

or remove individual entries from it via `xauth remove <entry>` where `<entry>` is one of the names in column 1 of the output of

```
$ xauth list
```

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
$ mkdir /work/${NEWUSER}/.Mathematica
$ ln -s /work/${NEWUSER}/.Mathematica
    ↪ /home/${NEWUSER}/.Mathematica
$ mkdir "/work/${NEWUSER}/Wolfram Mathematica"
$ ln -s "/work/${NEWUSER}/Wolfram Mathematica"
    ↪ "/home/${NEWUSER}/Wolfram Mathematica"
$ mkdir /work/${NEWUSER}/.Wolfram
$ ln -s /work/${NEWUSER}/.Wolfram /home/${NEWUSER}/.Wolfram
#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 4.26: 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 4.27: 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 4.26:

Listing 4.28: 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 4.29: 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 4.30: 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 4.2.5 on page 58) 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 [99]	`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`

[99] See also section *Format options for slurm commands* on page 119.

Table 4.9: Job control.

Action	Slurm	Torque/PBS	Maui
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`	

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[100]
- Apple Logo - Wiki Commons[101]
- Tux Logo - Wiki Commons[102]
- Computer shape - Openclipart[103]
- Server shape Openclipart[104]
- Light bulb - Openclipart[105]
- Warning triangle - Openclipart[106]
- Clock - Openclipart[107]
- TU Dortmund ITMC - itmc.tu-dortmund.de[108]
- Mordor Meme generated with imgflip[109]
- TotalView pictures - PC2 TotalView HowTo[110]
- Remaining screeshots and figures - created by the LiDO Team

[100] http://commons.wikimedia.org/wiki/Category:Microsoft_Windows_logos
[101] http://commons.wikimedia.org/wiki/File:Apple_logo_black.svg?uselang=de
[102] http://commons.wikimedia.org/wiki/Tux#/media/File:Tux.svg
[103] https://openclipart.org/detail/17391/computer
[104] https://openclipart.org/detail/171414/router
[105] https://openclipart.org/detail/211389/lightbulb
[106] https://openclipart.org/detail/14428/h0us3s-Signs-Hazard-Warning-9-by-h0us3s
[107] https://openclipart.org/detail/217065/3-oclock
[108] https://www.itmc.tu-dortmund.de/cms/de/home/anfahrt/anfahrt-hauptgebaeude/index.html
[109] https://imgflip.com/memegenerator/One-Does-Not-Simply
[110] https://wikis.uni-paderborn.de/pc2doc/Noctua-Software-TotalView