HPC Documentation

Release 0.0

HPC group

Aug 18, 2022

Contents

Attention: Stallo is getting old and will be shutdown this year. Hardware failures cause more and more nodes to fail due to high age. The **shutdown date has been postponed** to help us prepare the infrastructure for the switch. The system will stay in production and continue **service until at least 31. December 2020**.

We will help you with finding alternatives to your computational and storage needs and with moving your workflows and data to one of our other machines like Betzy, Saga and Fram. As we move closer to the shutdown, news, updated information and howtos will be published on the official Sigma2 migration page and *here*.

If you have questions, special needs or problems, please contact us at migration@metacenter.no

Stallo is the local computer cluster of UiT The Arctic University of Norway. If you are new here, you might want to learn the basics first here:

CHAPTER 1

Getting started

Here you will get the basics to work with Stallo. Please study carefully the links at the end of each paragraph to get more detailed information.

1.1 Get an account

If you are associated with UiT The Arctic University of Norway, you may apply locally. *How to get a local account on Stallo*

You can also apply for an account for Stallo or any of the other Norwegian computer clusters at the Metacenter account application. This is also possible if you already have a local account. *Getting an account*

1.2 Change temporary password

The password you got by SMS has to be changed on MAS within one week, or else the loginaccount will be closed again - and you need to contact us for reopening. You can't use the temporary password for logging in to Stallo.

1.3 Connect to Stallo

You may connect to Stallo via *SSH* to `stallo.uit.no`. This means that on Linux and OSX you may directly connect by opening a terminal and writing `ssh username@stallo.uit.no`. From Windows, you may connect via PuTTY which is available in the Software Center. X-forwarding for graphical applications is possible. There exists also a webinterface to allow easy graphical login. Please see the following link for details to all mentioned methods. *Logging in for the first time*

1.4 On nodes and files

When you login, you will be on a login node. Do *not* run any long-lasting programs here. The login node shall only be used for job preparation (see below) and simple file operations.

You will also be in your home directory `/home/username`. Here, you have 300 GB at your disposal that will be backed up regularly. For actual work, please use the global work area at `/global/work/username`. This space is not backed up, but it has a good performance and is 1000 TB in size. Please remove old files regularly. *Storage and backup*

To move files from your computer to Stallo or vice versa, you may use any tool that works with *ssh*. On Linux and OSX, these are scp, rsync, or similar programs. On Windows, you may use WinSCP. *Transferring files to/from Stallo*

1.5 Run a program

There are many programs pre-installed. You may get a list of all programs by typing `module avail`. You can also search within that list. `module avail blast` will search for Blast (case-insensitive). When you found your program of choice, you may load it using `module load BLAST+/2.7.1-intel-2017b-Python-2.7.14`. All program files will now be available, i.e. you can now simply call `blastp -version` to run Blast and check the loaded version. You can also compile your own software, if necessary. *Software Module Scheme*

To eventually run the program, you have to write a job script. In this script, you can define how long the job (i.e. the program) will run and how much memory and compute cores it needs. For the actual computation, you need to learn at least the basics of Linux shell scripting. You can learn some basics here: *Linux command line*.

When you wrote the job script, you can start it with `sbatch jobscript.sh`. This will put the script in the queue, where it will wait until an appropriate compute node is available. You can see the status of your job with `squeue -u username`. *Batch system* and *Job script examples*

Every job that gets started will be charged to your quota. Your quota is calculated in hours of CPU time and is connected to your specific project. To see the status of your quota account(s), type `cost` *CPU-hour quota and accounting*

1.6 Get help

Do you need help with Stallo? Write us an email to support@metacenter.no. You can also request new software (either an update or entirely new software), suggest changes to this documentation, or send us any other suggestions or issues concerning Stallo to that email address. Please also read the rest of this documentation.

Happy researching!

Fig. 1: Found in Via Notari, Pisa; (c) Roberto Di Remigio.

CHAPTER 2

News and notifications

News about planned/unplanned downtime, changes in hardware, and important changes in software will be published on the HPC UiT twitter account https://twitter.com/hpc_uit and on the login screen of stallo. For more information on the different situations see below.

2.1 System status and activity

You can get a quick overview of the system load on Stallo on the Sigma2 hardware page. More information on the system load, queued jobs, and node states can be found on the jobbrowser page (only visible from within the UiT network).

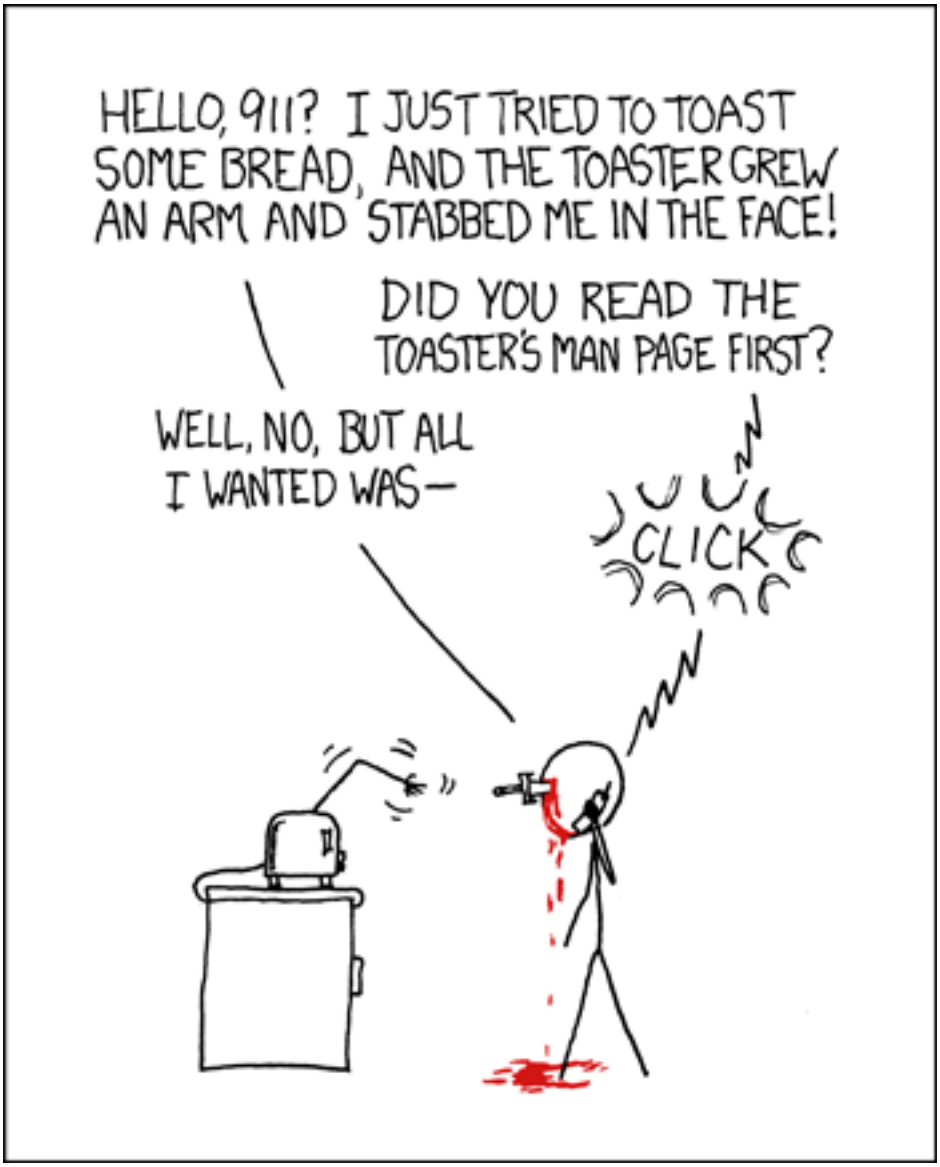
2.2 Planned/unplanned downtime

In the case of a planned downtime, a reservation will be made in the queuing system, so new jobs, that would not finish until the downtime, won't start. If the system goes down, running jobs will be restarted, if possible. We apologize for any inconveniences this may cause.

CHAPTER 3

Contact

If you need help, please file a support request via support@metacenter.no, and our local team of experts will try to assist you as soon as possible. Please state in the email that the request is about Stallo.



Credit: https://xkcd.com/293/

3.1 Postal address

```
Seksjon for digital forskningstjeneste/HPC
Nofima
Muninbakken 9-13
UiT Norges Arktiske Universitet
9037 Tromsø
```

Find us on Mazemap

CHAPTER 4

How to write good support requests

Writing good support requests is not only good for the support team, it is also better for you! Due to us having lots of help requests, the time to understand each problem is at a premium. The easier it is to understand yours, the faster we will get to it. Below is a list of good practices.

4.1 Never send support requests to staff members directly

Always send them to support@metacenter.no and staff members will pick them up there. On support@metacenter.no they get tracked and have higher visibility. Some staff members work on the support line only part time. Sending the request to support@metacenter.no makes sure that somebody will pick it up. Please note in the request that it is about Stallo, as there are more clusters that are handled with this email address.

4.2 Do not manhandle us as simple "Let me Google that for you" assistants

Seriously: have you searched the internet with the exact error message and the name of your application…? Other scientists may have had the very same problem and might have been successful in solving it. By the way: that's almost always how we start to research, too…

4.3 Give descriptive subject

Your subject line should be descriptive. "Problem on Stallo" is not a good subject line since it could be valid for basically every support email we get. The support staff is a team. The subjects are the first thing that we see. We would like to be able to classify emails according to subjects before even opening the email.

4.4 Include actual commands and error messages

We cover this below, but it's so important it needs to be mentioned at the top, too: include the actual command you run and actual error messages. Copy and paste. If you don't include this, we will be slightly annoyed and immediately ask this.

Please, do not screen shoot your ssh terminal and send us pictures (jpg, png, tiff...) of what you saw on your monitor! From these, we would be unable to cut & paste commands or error messages, unnecessarily slowing down our research on your problem. Your sample output does not need at all to "look good", and we don't need to know what fancy ssh- or terminal software you have: a simple text-based cut & paste directly into the mail or ticket is the best we can work on with.

4.5 New problem–new email

Do not send support requests by replying to unrelated issues. Every issue gets a number and this is the number that you see in the subject line. Replying to unrelated issues means that your email gets filed under the wrong thread and risks being overlooked.

4.6 The XY problem

This is a classic problem. Please read http://xyproblem.info. Often we know the solution but sometimes we don't know the problem.

In short (quoting from http://xyproblem.info):

- User wants to do X.
- User doesn't know how to do X, but thinks they can fumble their way to a solution if they can just manage to do Y.
- User doesn't know how to do Y either.
- User asks for help with Y.
- Others try to help user with Y, but are confused because Y seems like a strange problem to want to solve.
- After much interaction and wasted time, it finally becomes clear that the user really wants help with X, and that Y wasn't even a suitable solution for X.

To avoid the XY problem, if you struggle with Y but really what you are after is X, please also tell us about X. Tell us what you really want to achieve. Solving Y can take a long time. We have had cases where after enormous effort on Y we realized that the user wanted X and that Y was not the best way to achieve X.

4.7 Tell us also what worked

Rather often we get requests of the type "I cannot get X to run on two nodes". The request then does not mention whether it worked on one node or on one core or whether it never worked and that this was the first attempt. Perhaps the problem has even nothing to do with one or two nodes. In order to better isolate the problem and avoid wasting time with many back and forth emails, please tell us what actually worked so far. Tell us what you have tried to isolate the problem. This requires some effort from you but this is what we expect from you.

4.8 Specify your environment

Have you or your colleague compiled the code? Which modules were loaded? If you use non-default modules and you do not tell us about it, we will waste time when debugging with in a different environment.

4.9 Simple cases: Be specific, include commands and errors

Whatever you do, don't say that "X didn't work". Exactly give the commands you ran, environment (see above), and output error messages. The actual error messages mean a lot - include all the output, it is easy to copy and paste it.

The better you describe the problem the less we have to guess and ask.

Sometimes, just seeing the actual error message is enough to give an useful answer. For all but the simplest cases, you will need to make the problem reproducible, which you should *always* try anyway. See the following points.

4.10 Complex cases: Create an example which reproduces the problem

Create an example that we can ideally just copy paste and run and which demonstrates the problem. It is otherwise very time consuming if the support team needs to write input files and run scripts based on your possibly incomplete description. See also next point. Make this example available to us. We do not search and read read-protected files without your permission.

4.11 Make the example as small and fast as possible

You run a calculation which crashes after running for one week. You are tempted to write to support right away with this example but this is not a good idea. Before you send a support request with this example, first try to reduce it. Possibly and probably the crash can be reproduced with a much smaller example (smaller system size or grid or basis set). It is so much easier to schedule and debug a problem which crashes after few seconds compared to a run which crashes after hours. Of course this requires some effort from you but this is what we expect from you in order to create a useful support request. Often when isolating the problem, the problem and solution crystallize before even writing the support request.

CHAPTER 5

Frequently asked questions

5.1 Passwords

5.1.1 I forgot my password - what now?

You can reset it here: https://www.metacenter.no/user/

5.1.2 How do I change my password on Stallo?

The password can be changed on the password metacenter page, log in using your username on Stallo and the NOTUR domain.

The `passwd` command known from other Linuxes does not work. The Stallo system is using a centralised database for user management. This will override the password changes done locally on Stallo.

5.1.3 What is the ssh key fingerprint for stallo.uit.no?

The SHA256 key fingerprint is: `SHA256:YJpwZ91X5FNXTc/5SE1j9UR1UAAI4FFWVwNSoWoq6Hc`

The MD5 key fingerprint is: `MD5:36:a8:c5:f3:21:24:bb:bc:07:6f:af:4a:fe:3e:cb:9a`

If you are more of a visual person, use `ssh -o VisualHostKey=yes stallo.uit.no` and compare it to this visual key:

```
+---[RSA 1024]----+
|       oo+ooo**BO^|
|      . + ..+ o.BO|
|   . . * . o . +.=|
|    o * . . . . o.|
|     o   S     . .|
|    .   o         |
|   . . .          |
```

(continues on next page)

(continued from previous page)

```
| .  ..E          |
|  .. .           |
+----[SHA256]-----+
```

5.2 Installing software

5.2.1 I need Python package X but the one on Stallo is too old or I cannot find it

You can choose different Python versions with either the module system or using Anaconda/Miniconda. See here: *Software Module Scheme* and *Python, R, Matlab and Perl*.

In cases where this still doesn't solve your problem or you would like to install a package yourself, please read the next section below about installing without sudo rights.

If we don't have it installed, and installing it yourself is not a good solution for you, please contact us and we will do our best to help you.

5.2.2 Can I install Python software as a normal user without sudo rights?

Yes. Please see *Python, R, Matlab and Perl*.

5.3 Compute and storage quota

5.3.1 How can I check my disk quota and disk usage?

To check how large your disk quota is, and how much of it you have used, you can use the following command:

```
$ quota -s
```

Only home and project partitions have quota.

5.3.2 How many CPU hours have I spent?

For a simple summary, you can use the command `cost`, for more details, you can use:

```
$ gstatement --hours --summarize -p PROSJEKT -s YYYY-MM-DD -e YYYY-MM-DD
```

For a detailed overview over usage you can use:

```
$ gstatement --hours -p PROSJEKT -s YYYY-MM-DD -e YYYY-MM-DD
```

For more options see:

```
$ gstatement --help
```

5.4 Connecting via ssh

5.4.1 How can I export the display from a compute node to my desktop?

If you need to export the display from a compute node to your desktop you should

1. First login to Stallo with display forwarding.
2. Then you should reserve a node, with display forwarding, trough the queuing system.

Here is an example:

```
$ ssh -Y stallo.uit.no                  # log in with port forwarding
$ srun -N 1 -t 1:0:0 --pty bash -I      # reserve and log in on a compute node
```

This example assumes that you are running an X-server on your local desktop, which should be available for most users running Linux, Unix and Mac Os X. If you are using Windows you must install some X-server on your local PC.

5.4.2 How can I access a compute node from the login node?

Log in to stallo.uit.no and type e.g.:

```
$ ssh compute-1-3
```

or use the shorter version:

```
$ ssh c1-3
```

5.4.3 My ssh connections are dying / freezing

How to prevent your ssh connections from dying / freezing.

If your ssh connections more or less randomly are dying / freezing, try to add the following to your *local* `~/.ssh/config` file:

```
ServerAliveCountMax 3
ServerAliveInterval 10
```

(*local* means that you need to make these changes to your computer, not on stallo)

The above config is for OpenSSH, if you're using PUTTY you can take a look at this page explaining keepalives for a similar solution.

5.5 Jobs and queue system

5.5.1 I am not able to submit jobs longer than two days

Please read about *Partitions (queues) and services*.

5.5.2 Where can I find an example of job script?

You can find job script examples in *Job script examples*.

Relevant application specific examples (also for beginning users) for a few applications can be found in *Application guides*.

5.5.3 When will my job start?

How can I find out when my job will start?

To find out approximately when the job scheduler thinks your job will start, use the command:

```
squeue --start -j <job_id>
```

This command will give you information about how many CPUs your job requires, for how long, as well as when approximately it will start and complete. It must be emphasized that this is just a best guess, queued jobs may start earlier because of running jobs that finishes before they hit the walltime limit and jobs may start later than projected because new jobs are submitted that get higher priority.

5.5.4 How can I see the queing situation?

How can I see how my jobs are doing in the queue, if my jobs are idle, blocked, running?

On the webpage http://stallo-login2.uit.no/slurmbrowser/html/squeue.html you can find information about the current load on stallo, some information about the nodes, and the information you would get from the showq command, which is described below. You can also find information about your job and if you the job is running, you can find graphs about its usage of the CPUs, memory and so on.

If you prefer to use the command line, to see the job queue use:

```
$ squeue
```

5.5.5 Why does my job not start or give me error feedback when submitting?

Most often the reason a job is not starting is that Stallo is full at the moment and there are many jobs waiting in the queue. But sometimes there is an error in the job script and you are asking for a configuration that is not possible on Stallo. In such a case the job will not start.

To find out how to monitor your jobs and check their status see *Monitoring your jobs*.

Below are a few cases of why jobs don't start or error messages you might get:

Memory per core

> "When I try to start a job with 2GB of memory pr. core, I get the following error: `sbatch: error: Batch job submission failed: Requested node configuration is not available` With 1GB/core it works fine. What might be the cause to this?"

On Stallo we have two different configurations available; 16 core and 20 core nodes - with both a total of 32 GB of memory/node. If you ask for full nodes by specifying both number of nodes and cores/node together with 2 GB of memory/core, you will ask for 20 cores/node and 40 GB of memory. This configuration does not exist on Stallo. If you ask for 16 cores, still with 2GB/core, there is a sort of buffer within SLURM no allowing you to consume absolutely all memory available (system needs some to work). 2000MB/core works fine, but not 2 GB for 16 cores/node.

The solution we want to push in general is this:

```
#SBATCH -ntasks=80 # (number of nodes * number of cores, i.e. 5*16 or 4*20 = 80)
```

If you then ask for 2000MB of memory/core, you will be given 16 cores/node and a total of 16 nodes. 4000MB will give you 8 cores/node - everyone being happy. Just note the info about PE *CPU-hour quota and accounting*; mem-per-cpu 4000MB will cost you twice as much as mem-per-cpu 2000MB.

Please also note that if you want to use the whole memory on a node, do not ask for 32GB, but for 31GB or 31000MB as the node needs some memory for the system itself. For an example, see here: *Example on how to allocate entire memory on one node*

Step memory limit

"Why do I get `slurmstepd: Exceeded step memory limit` in my log/output?"

For slurm, the memory flag is a hard limit, meaning that when each core tries to utilize more than the given amount of memory, it is killed by the slurm-deamon. For example `$SBATCH --mem-per-cpu=2GB` means that you maximum can use 2 GB of memory per core. With memory intensive applications like Comsol or VASP, your job will likely be terminated. The solution to this problem is to specify the number of tasks irrespectively of cores/node and ask for as much memory you will need.

For instance:

```
#SBATCH --ntasks=20
#SBATCH --time=0-24:05:00
#SBATCH --mem-per-cpu=6000MB
```

QOSMaxWallDurationPerJobLimit

QOSMaxWallDurationPerJobLimit means that MaxWallDurationPerJobLimit has been exceeded. Basically, you have asked for more time than allowed for the given QOS/Partition. Please have a look at *Partitions (queues) and services*

Priority vs. Resources

Priority means that resources are in principle available, but someone else has higher priority in the queue. Resources means the at the moment the requested resources are not available.

5.5.6 Why is my job not starting on highmem nodes although the highmem queue is empty?

To prevent the highmem nodes from standing around idle, normal jobs may use them as well, using only 32 GB of the available memory. Hence, it is possible that the highmem nodes are busy, although you do not see any jobs queuing or running on *squeue -p highmem*.

5.5.7 How can I customize emails that I get after a job has completed?

Use the mail command and you can customize it to your liking but make sure that you send the email via the login node.

As an example, add and adapt the following line at the end of your script:

```
echo "email content" | ssh stallo-1.local 'mail -s "Job finished: ${SLURM_JOBID}"␣
↪firstname.lastname@uit.no'
```

5.5.8 How can I run many short tasks?

The overhead in the job start and cleanup makes it unpractical to run thousands of short tasks as individual jobs on Stallo.

The queueing setup on stallo, or rather, the accounting system generates overhead in the start and finish of a job of about 1 second at each end of the job. This overhead is insignificant when running large parallel jobs, but creates scaling issues when running a massive amount of shorter jobs. One can consider a collection of independent tasks as one large parallel job and the aforementioned overhead becomes the serial or unparallelizable part of the job. This is because the queuing system can only start and account one job at a time. This scaling problem is described by Amdahls Law.

If the tasks are extremly short, you can use the example below. If you want to spawn many jobs without polluting the queueing system, please have a look at *Running many sequential jobs in parallel using job arrays*.

By using some shell trickery one can spawn and load-balance multiple independent task running in parallel within one node, just background the tasks and poll to see when some task is finished until you spawn the next:

```
#!/usr/bin/env bash

# Jobscript example that can run several tasks in parallel.
# All features used here are standard in bash so it should work on
# any sane UNIX/LINUX system.
# Author: roy.dragseth@uit.no
#
# This example will only work within one compute node so let's run
# on one node using all the cpu-cores:
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20

# We assume we will (in total) be done in 10 minutes:
#SBATCH --time=0-00:10:00

# Let us use all CPUs:
maxpartasks=$SLURM_TASKS_PER_NODE

# Let's assume we have a bunch of tasks we want to perform.
# Each task is done in the form of a shell script with a numerical argument:
# dowork.sh N
# Let's just create some fake arguments with a sequence of numbers
# from 1 to 100, edit this to your liking:
tasks=$(seq 100)

cd $SLURM_SUBMIT_DIR

for t in $tasks; do
  # Do the real work, edit this section to your liking.
  # remember to background the task or else we will
  # run serially
  ./dowork.sh $t &

  # You should leave the rest alone...

  # count the number of background tasks we have spawned
  # the jobs command print one line per task running so we only need
  # to count the number of lines.
  activetasks=$(jobs | wc -l)
```

(continues on next page)

(continued from previous page)

```
  # if we have filled all the available cpu-cores with work we poll
  # every second to wait for tasks to exit.
  while [ $activetasks -ge $maxpartasks ]; do
    sleep 1
    activetasks=$(jobs | wc -l)
  done
done

# Ok, all tasks spawned. Now we need to wait for the last ones to
# be finished before we exit.
echo "Waiting for tasks to complete"
wait
echo "done"
```

And here is the `dowork.sh` script:

```
#!/usr/bin/env bash

# Fake some work, $1 is the task number.
# Change this to whatever you want to have done.

# sleep between 0 and 10 secs
let sleeptime=10*$RANDOM/32768

echo "Task $1 is sleeping for $sleeptime seconds"
sleep $sleeptime
echo "Task $1 has slept for $sleeptime seconds"
```

CHAPTER 6

Support staff

6.1 Our vision

Give our users a competitive advantage in the international science race.

6.1.1 Goal

Contribute to make our scientists the most efficient users of High Performance Computing (HPC), by creating the best support environment for any HPC user, and present them to the most efficient solutions to support their highest demands in HPC.

6.1.2 Services

The HPC-Group works within three areas:

- Technical and Operational computing resource support to the national program, and local initiatives.
- Basic and Advanced user support to projects and research groups that utilizes HPC or is expecting to do so in the future.
- Involvement in national development projects and initiatives, for instance in GRID Computing and new technology areas.

6.1.3 National resource site

Since 2000 UiT has been a Resource Site in the National High Performance Computing Consortium (NOTUR), a joint mission between the four Norwegian Universities in Bergen, Oslo, Trondheim and Tromsø. Since December 2014 UNINETT Sigma2 manages the national infrastructure and offers High Performance Computing and Data Storage.

6.1.4 Regional resource site

The HPC group at UiT also has a role to play as a regional resource site within HPC, offering HPC services to other institutions in Northern Norway. A strategic collaboration between UiT and the Norwegian Polar Institute within HPC has been ongoing since 1998.

6.2 Our team

- Radovan Bast
- Roy Dragseth
- Steinar Henden
- Stig Rune Jensen
- Dan Jonsson
- Espen Tangen
- Ilia Zhakun
- Jørn Dietze

CHAPTER 7

Open Question & Answer Sessions for All Users

7.1 Learn new tricks and ask & discuss questions

Meet the HPC staff, discuss problems and project ideas, give feedback or suggestions on how to improve services, and get advice for your projects.

Join us on Zoom, get yourself a coffee or tea and have a chat. It doesn't matter from which institute or university you are, you are **welcome to join at any time**.

This time we will start with a **short seminar about "Helpful Tools & Services we offer you might not know about"**. Afterwards we have the open question and answer session.

We can talk about: - Problems/questions regarding the Stallo/Vilje shutdown and migration - Questions regarding data storage and management - Help with programming and software management - Help with project organization and data management - Anything else If you think you might have a challenging question or topics for us, you can also send them to us before, so we can come prepared and help you better. If you have general or specific questions about the event: jorn.dietze@uit.no

7.2 Next event

- **2020-10-13, 13:00 - 15:00**, online Zoom meeting

7.2.1 Past events

- 2020-10-10, 13:00 - 15:00, online Zoom meeting
- 2020-04-23, 13:00 - 15:00, online Zoom meeting
- 2020-02-19, 10:00 - 12:00, main kantina
- 2019-11-12, 14:00 - 16:00, main kantina
- 2019-09-11, 10:00 - 12:30, MH bygget atrium

7.2.2 Similar events which serve as inspiration

- https://scicomp.aalto.fi/news/garage.html
- https://openworking.wordpress.com/2019/02/04/coding-problems-just-pop-over/

CHAPTER 8

Stallo Shutdown

Stallo is getting old and will be shutdown this year. Hardware failures cause more and more nodes to fail due to high age. The system will stay in production and continue **service until at least 31. December 2020**.

We will help you with finding alternatives to your computational and storage needs and with moving your workflows and data to one of our other machines like Betzy, Saga and Fram.

If you have questions, special needs or problems, please contact us at migration@metacenter.no

8.1 Alternatives

For an overview of alternatives especially for local projects, please have look this presentation

8.1.1 Computional resources

- Betzy
- Saga
- Fram
- Kubernetes based cloud infrastructure: NIRD toolkit

8.1.2 Storage

- NIRD
- UiT research storage for short- to mid-term storage of work in progress datasets (only for UiT researchers)
- UiT Research Data Portal for publishing final datasets and results
- Overview over public research data repositories

CHAPTER 9

About Stallo

9.1 Resource description

Key numbers about the Stallo cluster: compute nodes, node interconnect, operating system, and storage configuration.

	Aggregated	Per node
Peak performance	312 Teraflop/s	332 Gigaflop/s / 448 Gigaflops/s
# Nodes	304 x HP BL460 gen8 blade servers 328 x HP SL230 gen8 servers	1 x HP BL460 gen8 blade servers 1 x. HP SL230 gen8 servers
# CPU's / # Cores	608 / 4864 656 / 6560	2 / 16 2 / 20
Processors	608 x 2.60 GHz Intel Xeon E5 2670 656 x 2.80 GHz Intel Xeon E5 2680	2 x 2.60 GHz Intel Xeon E5 2670 2 x 2.80 Ghz Intel Xeon E5 2680
Total memory	26.2 TB	32 GB (32 nodes with 128 GB)
Internal storage	155.2 TB	500 GB (32 nodes with 600GB raid)
Centralized storage	2000 TB	2000 TB
Interconnect	Gigabit Ethernet + Infiniband [1]	Gigabit Ethernet + Infiniband [1]

Compute racks	11
Infrastructure racks	2
Storage racks	3

1) All nodes in the cluster are connected with Gigabit Ethernet and QDR Infiniband.

9.2 Stallo - a Linux cluster

This is just a quick and brief introduction to the general features of Linux Clusters.

9.2.1 A Linux Cluster - one machine, consisting of many machines

On one hand you can look at large Linux Clusters as rather large and powerful supercomputers, but on the other hand you can look at them as just a large bunch of servers and some storage system(s) connected with each other through a (high speed) network. Both of these views are fully correct, and it's therefore important to be aware of the strengths and the limitations of such a system.

9.2.2 Clusters vs. SMP's

Until July 2004, most of the supercomputers available to Norwegian HPC users were more or less large Symmetric Multi Processing (SMP's) systems; like the HP Superdome's at UiO and UiT, the IBM Regatta at UiB and the SGI Origion and IBM p575 systems at NTNU.

On SMP systems most of the resources (CPU, memory, home disks, work disks, etc) are more or less uniformly accessible for any job running on the system. This is a rather simple picture to understand, it's nearly as your desktop machine – just more of everything: More users, more CPU's, more memory, more disks etc.

On a Linux Cluster the picture is quite different. The system consists of several independent compute nodes (servers) connected with each other through some (high speed) network and maybe hooked up on some storage system. So the HW resources (like CPU, memory, disk, etc) in a cluster are in general distributed and only locally accessible at each server.

9.3 Linux operating system (Rocks): http://www.rocksclusters.org/

Since 2003, the HPC-group at has been one of five international development sites for the Linux operating system Rocks. Together with people in Singapore, Thailand, Korea and USA, we have developed a tool that has won international recognition, such as the price for "Most important software innovation " both in 2004 and 2005 in HPCWire. Now Rocks is a de-facto standard for cluster-management in Norwegian supercomputing.

9.4 Stallo - Sami mythology

In the folklore of the Sami, a Stallo (also Stallu or Stalo) is a sami wizard. "The Sami traditions up North differ a bit from other parts of Norwegian traditions. You will find troll and draug and some other creatures as well, but the Stallo is purely Sami. He can change into all kinds of beings,; animals, human beings, plants, insects, bird – anything. He can also "turn" the landscape so you miss your direction or change it so you don't recognise familiar surroundings. Stallo is very rich and smart, he owns silver and reindeers galore, but he always wants more. To get what he wants he tries to trick the Samis into traps, and the most popular Sami stories tell how people manage to fool Stallo." NB! Don't mix Stallo with the noaide! He is a real wizard whom people still believe in.

CHAPTER 10

Guidelines for use of computer equipment at the UiT The Arctic University of Norway

10.1 Definitions

Explanation of words and expressions used in these guidelines.

users: Every person who has access to and who uses the University's computer equipment. This includes employees students and others who are granted access to the computer equipment.

user contract: Agreement between users and department(s) at the University who regulate the user's right of access to the computer equipment. The user contract is also a confirmation that these guidelines are accepted by the user.

data: All information that is on the computer equipment. This includes both the contents of data files and software.

computer network: Hardware and/or software which makes it possible to establish a connection between two or more computer terminals. This includes both private, local, national and international computer networks which are accessible through the computer equipment.

breakdown: Disturbances and abnormalities which prevent the user from (stoppage) maximum utilization of the computer equipment.

computer equipment: This includes hardware, software, data, services and computer network.

hardware: Mechanical equipment that can be used for data processing.

private data: Data found in reserved or private areas or that are marked as private. The data in a user's account is to be regarded as private irrespective of the rights attached to the data.

resources: Resources refers to the computer equipment including time and the capacity available for the persons who are connected to the equipment.

10.2 Purpose

The purpose of these guidelines is to contribute towards the development of a computer environment in which the potential provided by the computer equipment can be utilized in the best possible way by the University and by society at large. This is to promote education and research and to disseminate knowledge about scientific methods and results.

10.3 Application

These guidelines apply to the use of the University's computer equipment and apply to all users who are granted access to the computer equipment. The guidelines are to be part of a user contract and are otherwise to be accessible at suitable places such as the terminal room. The use of the computer equipment also requires that the user knows any possible supplementary regulations.

10.4 Good Faith

Never leave any doubt as to your identity and give your full name in addition to explaining your connection to the University. A user is always to identify him/ herself by name, his/ her own user identity, password or in another regular way when using services on the computer network. The goodwill of external environments is not to be abused by the user accessing information not intended for the user, or by use of the services for purposes other than that for which they are intended. Users are to follow the instructions from system administrators about the use of computer equipment. Users are also expected to familiarize themselves with the user guides, manuals, documentation etc. in order to reduce the risk of breakdowns or loss of data or equipment (through ignorance). On termination of employment or studies, it is the users responsibility to ensure that copies of data owned or used by the University are secured on behalf of the University.

10.5 Data Safety

Users are obliged to take the necessity measures to prevent the loss of data etc. by taking back-up copies, careful storage of media, etc. This can be done by ensuring that the systems management take care of it. Your files are in principle personal but should be protected so that they cannot be read by others. Users are obliged to protect the integrity of their passwords or other safety elements known to them, in addition to preventing unauthorized people from obtaining access to the computer equipment. Introducing data involves the risk of unwanted elements such as viruses. Users are obliged to take measures to protect the computer equipment from such things. Users are obliged to report circumstances that may have importance for the safety or integrity of the equipment to the closest superior or to the person who is responsible for data safety.

10.6 Respect for Other Users Privacy

Users may not try to find out another persons password, etc., nor try to obtain unauthorized access to another persons data. This is true independent of whether or not the data is protected. Users are obliged to familiarize themselves with the special rules that apply to the storage of personal information (on others). If a user wishes to register personal information, the user concerned is obliged to ensure that there is permission for this under the law for registration of information on persons or rules authorized by the law or with acceptance of rights given to the University. In cases where registration of such information is not permitted by these rules the user is obliged to apply for (and obtain?) the necessary permission. Users are bound by the oaths of secrecy concerning personal relationships of which the

user acquires knowledge through use of computer equipment, ref. to the definition in section 13 second section of the Administration Law, (forvaltningslovens section 13 annet ledd).

10.7 Proper Use

The computer equipment of the University may not be used to advance slander or discriminating remarks, nor to distribute pornography or spread secret information, or to violate the peace of private life or to incite or take part in illegal actions. This apart, users are to restrain from improper communication on the network.

The computer equipment is to be used in accordance with the aims of the University. This excludes direct commercial use.

10.8 Awareness of the Purposes for Use of Resources

The computer equipment of the University is to strengthen and support professional activity, administration, research and teaching. Users have a co-responsibility in making the best possible use of the resources.

10.9 Rights

Data is usually linked to rights which make their use dependent on agreements with the holder of the rights. Users commit themselves to respecting other people's rights. This applies also when the University makes data accessible. The copying of programs in violation of the rights of use and/or license agreement is not permitted.

10.10 Liability

Users themselves are responsible for the use of data which is made accessible via the computer equipment. The University disclaims all responsibility for any loss that results from errors or defects in computer equipment, including for example, errors or defects in data, use of data from accessible databases or other data that has been obtained through the computer network etc. The University is not responsible for damage or loss suffered by users as a consequence of insufficient protection of their own data.

10.11 Surveillance

The systems manager has the right to seek access to the individual user's reserved areas on the equipment for the purpose of ensuring the equipment's' proper functioning or to control that the user does not violate or has not violated the regulations in these guidelines. It is presupposed that such access is only sought when it is of great importance to absolve the University from responsibility or bad reputation. If the systems manager seeks such access, the user should be warned about it in an appropriate way. Ordinarily such a warning should be given in writing and in advance. If the use of a workstation, terminal or other end user equipment is under surveillance because of operational safety or other considerations, information about this must be given in an appropriate way. The systems managers are bound by oaths of secrecy with respect to information about the user or the user's activity which they obtain in this way, the exception being that circumstances which could represent a violation of these guidelines may be reported to superior authorities.

10.12 Sanctions

Breach of these guidelines can lead to the user being denied access to the University's data services, in addition to which there are sanctions that the University can order, applying other rules. Breach of privacy laws, oaths of secrecy etc. can lead to liability or punishment. The usual rules for dismissal or (forced) resignation of employees or disciplinary measures against students, apply to users who misuse the computer equipment. The reasons for sanctions against a user are to be stated, and can be ordered by the person who has authority given by the University. Disciplinary measures against students are passed by the University Council, ref. section 47 of the University law.

10.13 Complaints

Complaints about sanctions are to be directed to the person(s) who order sanctions. If the complaint is not complied with, it is sent on to the University Council for final decision. Complaints about surveillance have the same procedure as for sanctions. The procedure for complaints about dismissal or resignation of employees are the usual rules for the University, and rules otherwise valid in Norwegian society. Decisions about disciplinary measures against students cannot be complained about, See § 47 of the University law.

CHAPTER 11

Getting an account

11.1 Getting started on the machine (account, quota, password)

Before you start using Stallo, please read the UiT The Arctic University of Norway's *Guidelines for use of computer equipment at the UiT The Arctic University of Norway*.

A general introduction to Sigma2 be found at www.sigma2.no

11.2 How to get an account and a CPU quota on Stallo

To be able to work on Stallo you must have an account and you must have been granted CPU time on the system. There are two ways to achieve this:

11.2.1 Research Council Quota

National users (including users from UiT) may apply for an account and a CPU quota from the Research Councils share of Stallo. If you want to apply for such a quota please use this National quota form.

11.2.2 UiT Quota

"Local users" (i.e. users from UiT and users affiliated with UiT in some way) can apply for an account and a quota from UiT's share of Stallo. If you want to apply for such a quota follow the instructions here:

How to get a local account on Stallo

To get a local account on Stallo, you need to provide us with:

- Your full name, date of birth, and nationality.

- Your position (master student, PhD, PostDoc, staff member, visitor/guest).
- Your mobile phone number. This is necessary for recovery of passwords.
- Your institutional mail address (i.e. your work email at the research institution to which you belong)
- The name and address of the instruction you belong to; also including name of the center, institute etc.
- A preferred username. Note: If you already have a Notur user account name, please enter that one. A username is defined as a sequence of two to thirty lowercase alphanumeric characters where the first letter may only be a lowercase letter.
- Necessary additional group and account memberships.
- Optional: If you know in advance, please let us know: how many CPU hours you expect to use, how much long-term storage space (GB) you will need, and what software you will use. Partial answers are also welcome. The more we know about the needs of our users, the better services we can provide and the better we can plan for the future.

If you are a staff member and need to get a local project, we need information about the project:

- Name of the project
- Brief description of the project
- Field of science for the project
- Name of additonal members of the project

If you are a student, PhD or post-doc, you need to also provide us with the name of your advisor and name of the project you are to be a member of.

Compile this list in a proper manner and send to support@metacenter.no with a comment that the application is for a local account on Stallo.

Please note that most users who are allowed to apply for a UiT quota also are allowed to apply for a quota from the Research Council – at the same time!

Before you start using Stallo, please read the UiT The Arctic University of Norway's *Guidelines for use of computer equipment at the UiT The Arctic University of Norway*.

CHAPTER 12

Logging in for the first time

12.1 Log in with SSH

An *SSH* client (Secure SHell) is the required tool to connect to Stallo. An *SSH* client provides secure encrypted communications between two hosts over an insecure network.

If you already have *ssh* installed on your UNIX-like system, have a user account and password on a Notur system, login may be as easy as typing

```
ssh <machine name>          (for instance: ssh stallo.uit.no)
```

into a terminal window.

If your user name on the machine differs from your user name on the local machine, use the -l option to specify the user name on the machine to which you connect. For example:

```
ssh <machine name> -l [username]
```

And if you need X-forwarding (for instance, if you like to run Emacs in it's own window) you must log in like this:

```
ssh -X -Y <machine name>
```

No matter how you login, you will need to confirm that the connection shall be trusted. The SHA256 key fingerprint of `stallo.uit.no` is:

```
SHA256:YJpwZ91X5FNXTc/5SE1j9UR1UAAI4FFWVwNSoWoq6Hc
```

So you should get precisely this message the first time you login via ssh:

```
The authenticity of host 'stallo.uit.no (129.242.2.68)' can't be established.
RSA key fingerprint is SHA256:YJpwZ91X5FNXTc/5SE1j9UR1UAAI4FFWVwNSoWoq6Hc.
Are you sure you want to continue connecting (yes/no)?
```

If you see this message with precisely this code, you can continue by typing `yes` and pressing *Enter*. If you connect to Stallo for the first time and ssh does *not* show you this key, please contact support@metacenter.no immediately.

12.1.1 Log in with an ssh key

To avoid entering your password every time you login and to increase security, you can log in with an ssh keypair. This keypair consists of a private key that you have to store on your computer and a public key that you can store on Stallo. On Linux or OSX simply type:

```
ssh-keygen
```

and follow the instructions on the screen. Please use a good passphrase. You will have to enter this passphrase the first time you log in after a reboot of the computer, but not anymore afterwards. To copy the public key to Stallo, type:

```
ssh-copy-id <username>@stallo.uit.no
```

To learn more about ssh keys, have a look at this page.

On Windows, you can use PuTTYgen that comes with PuTTY. More information on ssh.com.

12.1.2 SSH clients for Windows and Mac

At the OpenSSH page you will find several *SSH* alternatives for both Windows and Mac.

Please note that Mac OS X comes with its own implementation of *OpenSSH*, so you don't need to install any third-party software to take advantage of the extra security *SSH* offers. Just open a terminal window and jump in.

12.1.3 Learning more about SSH

To learn more about using SSH, please also consult the OpenSSH page page and take a look at the manual page on your system (*man ssh*).

12.2 Obtain a new password

When you have been granted an account on stallo.uit.no, your username and password is sent to you separately. The username by email and the password by SMS. The password you receive by SMS is temporally and only valid for 7 days.

12.2.1 The passwd command does not seem to work. My password is reset back to the old one after a while. Why is this happening?

The Stallo system is using a centralised database for user management. This will override the password changes done locally on Stallo.

The password can be changed here, log in using your username on stallo and the NOTUR domain.

12.3 Logging on the compute nodes

Information on how to log in on a compute node.

Some times you may want to log on a compute node (for instance to check out output files on the local work area on the node), and this is also done by using SSH. From stallo.uit.no you can log in to compute-x-y the following way:

```
ssh -Y compute-x-y      (for instance: ssh compute-5-8)
```

or short

```
ssh -Y cx-y         (for instance: ssh c5-8)
```

If you don't need display forwarding you can omit the "-Y" option above.

If you for instance want to run a program interactively on a compute node, and with display forwarding to your desktop you should in stead do something like this:

1. first login to Stallo with display forwarding,
2. then you should reserve a node through the queuing system

Below is an example on how you can do this:

```
1) Log in on Stallo with display forwarding.

   $ ssh -Y stallo.uit.no

2) Reserve and log in on a compute node with display forwarding.
   (start an interactive job.)

   $ srun -N 1 -t 1:0:0 --pty bash -i

3) Open a new terminal window, type squeue -j <jobid> (it shows you which node(s) was␣
↪allocated
   to that specific job). Then ssh -Y <nodename> to that node and start your␣
↪preferred gui.
```

12.4 Graphical logon to Stallo

If you want to run applications with graphical user interfaces we recommend you to use the remote desktop service on Stallo.

If you have a new account and you have never logged in on Stallo before, first log in with a classical ssh connection (see above). Afterwards you can use the graphical logon. Otherwise it can happen that your /home will not be created and you will get an error message of the type: "Could not chdir to home directory /home/your_user_name: No such file or directory"

Important:

If you are connecting from outside the networks of UNINETT and partners you need to log into stallo-gui.uit.no with ssh to verify that you have a valid username and password on the Stallo system. After a successful login the service will automatically allow you to connect from the ip-address your client currently has. The connection will be open for at least 5 minutes after you log in. There is no need to keep the ssh-connection open after you have connected to the remote desktop, in fact you will be automatically logged off after 5 seconds.

CHAPTER 13

CPU-hour quota and accounting

13.1 CPU quota

To use the batch system you have to have a cpu quota, either local or national. For every job you submit we check that you have sufficient quota to run it and you will get a warning if you do not have sufficient cpu-hours to run the job. The job will be submitted to queue, but will not start until you have enough cpu-hours to run it.

13.2 Resource charging

We charge for used resources, both cpu and memory. The accounting system charges for used processor equivalents (PE) times used walltime, so if you ask for more than (total memory pr. node)/(total cores pr. node) - in the example below 32GB/20cores = 1.6 GB.

The concept of PE defines a processor equivalent as the resource unit 1 core and 1 unit of memory. For a node with 2 GB og memory pr core, i.e. 32 GB of memory and 16 cores - 1 PE equals to 1 core and 2GB of memory. Currently there is no common definition on the memory unit, other than the one specified above.

The best way to explain the concept of PE is by example: Assume that you have a node with 20 cpu-cores and 32 GB memory:

```
if you ask for less than 1.6GB memory per core then PE will equal the cpu count.

if you ask for 3.2GB memory per core then PE will be twice the cpu-count.

if you ask for 32 GB memory then PE=20 as you only can run one cpu per compute node.
```

The available memory and core settings for Stallo are explained here: *About Stallo*

13.3 Inspecting your quota

You can use the cost command to check how much cpu-hours are left on your allocation:

```
[user@stallo-1 ~]$ cost
Id  Name    Amount    Reserved Balance  CreditLimit Deposited Available Percentage
--- ------- --------- -------- -------- ----------- --------- --------- ----------
272 nnXXXXk 168836.65 96272.00 72564.65        0.00 290000.00  72564.65 58.22
10  nnYYYYk   4246.04     0.00  4246.04        0.00 150000.00   4246.04 2.83
```

The column meaning is

Amount: The number of hours available for new jobs.

Reserved: The number of hours reserved for currently running jobs.

Balance: Amount - Reserved.

CreditLimit: Allocated low-pri quota.

Deposited: Allocated normal quota

13.3.1 Inspecting historic use

You can view the accounting history of your projects using:

```
gstatement --hours --summarize -s YYYY-MM-DD -e YYYY-MM-DD -p nnXXXXk
```

for more detail see:

```
gstatement --man
```

CHAPTER 14

Dos and don’ts

- Never run calculations on the home disk
- Always use the SLURM queueing system
- The login nodes are only for editing files and submitting jobs
- Do not run calculations interactively on the login nodes

CHAPTER 15

Batch system

The Stallo system is a resource that is shared between many of users and to ensure fair use everyone must do their computations by submitting jobs through a batch system that will execute the applications on the available resources.

The batch system on Stallo is SLURM (Simple Linux Utility for Resource Management.)

15.1 Creating a job script

To run a job on the system you need to create a job script. A job script is a regular shell script (bash) with some directives specifying the number of CPUs, memory, etc., that will be interpreted by the batch system upon submission.

You can find job script examples in *Job script examples*.

After you wrote your job script as shown in the examples, you can start it with:

```
sbatch jobscript.sh
```

15.2 How to pass command-line parameters to the job script

It is sometimes convenient if you do not have to edit the job script every time you want to change the input file. Or perhaps you want to submit hundreds of jobs and loop over a range of input files. For this it is handy to pass command-line parameters to the job script. For an overview of the different possible parameters, see *SLURM Parameter*.

In SLURM you can do this:

```
$ sbatch myscript.sh myinput myoutput
```

And then you can pick the parameters up inside the job script:

```
#!/bin/bash
```

(continues on next page)

(continued from previous page)

```
#SBATCH ...
#SBATCH ...
...

# argument 1 is myinput
# argument 2 is myoutput
mybinary.x < ${1} > ${2}
```

For recommended sets of parameters see also *Settings for OpenMP and MPI jobs*.

15.3 Walltime

We recommend you to be as precise as you can when specifying the parameters as they will inflict on how fast your jobs will start to run. We generally have these rules for prioritizing jobs:

1. Large jobs, that is jobs with high CPUcounts, are prioritized.
2. Short jobs take precedence over long jobs.
3. Use fairshare. This means that users with many jobs running will get a decreased priority compared to other users.

To find out whether all users within one project share the same priority, run:

```
$ sshare -a -A nnNNNNk
```

For a given account (project) consider the column "RawShares". If the RawShares for the users is "parent", they all share the same fairshare priority. If it is a number, they have individual priorities.

CHAPTER 16

Job script examples

16.1 Basic examples

16.1.1 General blueprint for a jobscript

You can save the following example to a file (e.g. run.sh) on Stallo. Comment the two `cp` commands that are just for illustratory purpose (lines 46 and 55) and change the SBATCH directives where applicable. You can then run the script by typing:

```
$ sbatch run.sh
```

Please note that all values that you define with SBATCH directives are hard values. When you, for example, ask for 6000 MB of memory (`--mem=6000MB`) and your job uses more than that, the job will be automatically killed by the manager.

```
#!/bin/bash -l

##############################
#       Job blueprint        #
##############################

# Give your job a name, so you can recognize it in the queue overview
#SBATCH --job-name=example

# Define, how many nodes you need. Here, we ask for 1 node.
# Each node has 16 or 20 CPU cores.
#SBATCH --nodes=1
# You can further define the number of tasks with --ntasks-per-*
# See "man sbatch" for details. e.g. --ntasks=4 will ask for 4 cpus.

# Define, how long the job will run in real time. This is a hard cap meaning
# that if the job runs longer than what is written here, it will be
# force-stopped by the server. If you make the expected time too long, it will
```

(continues on next page)

(continued from previous page)

```
# take longer for the job to start. Here, we say the job will take 5 minutes.
#              d-hh:mm:ss
#SBATCH --time=0-00:05:00

# Define the partition on which the job shall run. May be omitted.
#SBATCH --partition normal

# How much memory you need.
# --mem will define memory per node and
# --mem-per-cpu will define memory per CPU/core. Choose one of those.
#SBATCH --mem-per-cpu=1500MB
##SBATCH --mem=5GB    # this one is not in effect, due to the double hash

# Turn on mail notification. There are many possible self-explaining values:
# NONE, BEGIN, END, FAIL, ALL (including all aforementioned)
# For more values, check "man sbatch"
#SBATCH --mail-type=END,FAIL

# You may not place any commands before the last SBATCH directive

# Define and create a unique scratch directory for this job
SCRATCH_DIRECTORY=/global/work/${USER}/${SLURM_JOBID}.stallo-adm.uit.no
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

# You can copy everything you need to the scratch directory
# ${SLURM_SUBMIT_DIR} points to the path where this script was submitted from
cp ${SLURM_SUBMIT_DIR}/myfiles*.txt ${SCRATCH_DIRECTORY}

# This is where the actual work is done. In this case, the script only waits.
# The time command is optional, but it may give you a hint on how long the
# command worked
time sleep 10
#sleep 10

# After the job is done we copy our output back to $SLURM_SUBMIT_DIR
cp ${SCRATCH_DIRECTORY}/my_output ${SLURM_SUBMIT_DIR}

# In addition to the copied files, you will also find a file called
# slurm-1234.out in the submit directory. This file will contain all output that
# was produced during runtime, i.e. stdout and stderr.

# After everything is saved to the home directory, delete the work directory to
# save space on /global/work
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}

# Finish the script
exit 0
```

16.1.2 Running many sequential jobs in parallel using job arrays

In this example we wish to run many similar sequential jobs in parallel using job arrays. We take Python as an example but this does not matter for the job arrays:

```
#!/usr/bin/env python

import time

print('start at ' + time.strftime('%H:%M:%S'))

print('sleep for 10 seconds ...')
time.sleep(10)

print('stop at ' + time.strftime('%H:%M:%S'))
```

Save this to a file called “test.py” and try it out:

```
$ python test.py

start at 15:23:48
sleep for 10 seconds ...
stop at 15:23:58
```

Good. Now we would like to run this script 16 times at the same time. For this we use the following script:

```
#!/bin/bash -l

#####################
# job-array example #
#####################

#SBATCH --job-name=example

# 16 jobs will run in this array at the same time
#SBATCH --array=1-16

# run for five minutes
#               d-hh:mm:ss
#SBATCH --time=0-00:05:00

# 500MB memory per core
# this is a hard limit
#SBATCH --mem-per-cpu=500MB

# you may not place bash commands before the last SBATCH directive

# define and create a unique scratch directory
SCRATCH_DIRECTORY=/global/work/${USER}/job-array-example/${SLURM_JOBID}
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

cp ${SLURM_SUBMIT_DIR}/test.py ${SCRATCH_DIRECTORY}

# each job will see a different ${SLURM_ARRAY_TASK_ID}
echo "now processing task id:: " ${SLURM_ARRAY_TASK_ID}
python test.py > output_${SLURM_ARRAY_TASK_ID}.txt

# after the job is done we copy our output back to $SLURM_SUBMIT_DIR
cp output_${SLURM_ARRAY_TASK_ID}.txt ${SLURM_SUBMIT_DIR}

# we step out of the scratch directory and remove it
```

(continues on next page)

(continued from previous page)

```
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}

# happy end
exit 0
```

Submit the script and after a short while you should see 16 output files in your submit directory:

```
$ ls -l output*.txt

-rw------- 1 user user 60 Oct 14 14:44 output_1.txt
-rw------- 1 user user 60 Oct 14 14:44 output_10.txt
-rw------- 1 user user 60 Oct 14 14:44 output_11.txt
-rw------- 1 user user 60 Oct 14 14:44 output_12.txt
-rw------- 1 user user 60 Oct 14 14:44 output_13.txt
-rw------- 1 user user 60 Oct 14 14:44 output_14.txt
-rw------- 1 user user 60 Oct 14 14:44 output_15.txt
-rw------- 1 user user 60 Oct 14 14:44 output_16.txt
-rw------- 1 user user 60 Oct 14 14:44 output_2.txt
-rw------- 1 user user 60 Oct 14 14:44 output_3.txt
-rw------- 1 user user 60 Oct 14 14:44 output_4.txt
-rw------- 1 user user 60 Oct 14 14:44 output_5.txt
-rw------- 1 user user 60 Oct 14 14:44 output_6.txt
-rw------- 1 user user 60 Oct 14 14:44 output_7.txt
-rw------- 1 user user 60 Oct 14 14:44 output_8.txt
-rw------- 1 user user 60 Oct 14 14:44 output_9.txt
```

16.1.3 Packaging smaller parallel jobs into one large parallel job

There are several ways to package smaller parallel jobs into one large parallel job. The preferred way is to use Job Arrays. Browse the web for many examples on how to do it. Here we want to present a more pedestrian alternative which can give a lot of flexibility.

In this example we imagine that we wish to run 5 MPI jobs at the same time, each using 4 tasks, thus totalling to 20 tasks. Once they finish, we wish to do a post-processing step and then resubmit another set of 5 jobs with 4 tasks each:

```
#!/bin/bash

#SBATCH --job-name=example
#SBATCH --ntasks=20
#SBATCH --time=0-00:05:00
#SBATCH --mem-per-cpu=500MB

cd ${SLURM_SUBMIT_DIR}

# first set of parallel runs
mpirun -n 4 ./my-binary &
mpirun -n 4 ./my-binary &
mpirun -n 4 ./my-binary &
mpirun -n 4 ./my-binary &
mpirun -n 4 ./my-binary &

wait

# here a post-processing step
```

(continues on next page)

(continued from previous page)

```
# ...

# another set of parallel runs
mpirun -n 4 ./my-binary &
mpirun -n 4 ./my-binary &
mpirun -n 4 ./my-binary &
mpirun -n 4 ./my-binary &
mpirun -n 4 ./my-binary &

wait

exit 0
```

The `wait` commands are important here - the run script will only continue once all commands started with `&` have completed.

16.1.4 Example on how to allocate entire memory on one node

```
#!/bin/bash -l

#####################################################
# Example for a job that consumes a lot of memory #
#####################################################

#SBATCH --job-name=example

# we ask for 1 node
#SBATCH --nodes=1

# run for five minutes
#              d-hh:mm:ss
#SBATCH --time=0-00:05:00

# total memory for this job
# this is a hard limit
# note that if you ask for more than one CPU has, your account gets
# charged for the other (idle) CPUs as well
#SBATCH --mem=31000MB

# turn on all mail notification
#SBATCH --mail-type=ALL

# you may not place bash commands before the last SBATCH directive

# define and create a unique scratch directory
SCRATCH_DIRECTORY=/global/work/${USER}/example/${SLURM_JOBID}
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

# we copy everything we need to the scratch directory
# ${SLURM_SUBMIT_DIR} points to the path where this script was submitted from
cp ${SLURM_SUBMIT_DIR}/my_binary.x ${SCRATCH_DIRECTORY}

# we execute the job and time it
time ./my_binary.x > my_output
```

(continues on next page)

(continued from previous page)

```
# after the job is done we copy our output back to $SLURM_SUBMIT_DIR
cp ${SCRATCH_DIRECTORY}/my_output ${SLURM_SUBMIT_DIR}

# we step out of the scratch directory and remove it
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}

# happy end
exit 0
```

16.1.5 How to recover files before a job times out

Possibly you would like to clean up the work directory or recover files for restart in case a job times out. In this example we ask Slurm to send a signal to our script 120 seconds before it times out to give us a chance to perform clean-up actions.

```
#!/bin/bash -l

# job name
#SBATCH --job-name=example

# replace this by your account
#SBATCH --account=...

# one core only
#SBATCH --ntasks=1

# we give this job 4 minutes
#SBATCH --time=0-00:04:00

# asks SLURM to send the USR1 signal 120 seconds before end of the time limit
#SBATCH --signal=B:USR1@120

# define the handler function
# note that this is not executed here, but rather
# when the associated signal is sent
your_cleanup_function()
{
    echo "function your_cleanup_function called at $(date)"
    # do whatever cleanup you want here
}

# call your_cleanup_function once we receive USR1 signal
trap 'your_cleanup_function' USR1

echo "starting calculation at $(date)"

# the calculation "computes" (in this case sleeps) for 1000 seconds
# but we asked slurm only for 240 seconds so it will not finish
# the "&" after the compute step and "wait" are important
sleep 1000 &
wait
```

16.2 OpenMP and MPI

You can download the examples given here to a file (e.g. run.sh) and start it with:

```
$ sbatch run.sh
```

16.2.1 Example for an OpenMP job

```
#!/bin/bash -l

##############################
# example for an OpenMP job #
##############################

#SBATCH --job-name=example

# we ask for 1 task with 20 cores
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=20

# exclusive makes all memory available
#SBATCH --exclusive

# run for five minutes
#               d-hh:mm:ss
#SBATCH --time=0-00:05:00

# turn on all mail notification
#SBATCH --mail-type=ALL

# you may not place bash commands before the last SBATCH directive

# define and create a unique scratch directory
SCRATCH_DIRECTORY=/global/work/${USER}/example/${SLURM_JOBID}
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

# we copy everything we need to the scratch directory
# ${SLURM_SUBMIT_DIR} points to the path where this script was submitted from
cp ${SLURM_SUBMIT_DIR}/my_binary.x ${SCRATCH_DIRECTORY}

# we set OMP_NUM_THREADS to the number of available cores
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

# we execute the job and time it
time ./my_binary.x > my_output

# after the job is done we copy our output back to $SLURM_SUBMIT_DIR
cp ${SCRATCH_DIRECTORY}/my_output ${SLURM_SUBMIT_DIR}

# we step out of the scratch directory and remove it
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}
```

(continues on next page)

(continued from previous page)

```
# happy end
exit 0
```

16.2.2 Example for a MPI job

```
#!/bin/bash -l

##########################
# example for an MPI job #
##########################

#SBATCH --job-name=example

# 80 MPI tasks in total
# Stallo has 16 or 20 cores/node and therefore we take
# a number that is divisible by both
#SBATCH --ntasks=80

# run for five minutes
#               d-hh:mm:ss
#SBATCH --time=0-00:05:00

# 500MB memory per core
# this is a hard limit
#SBATCH --mem-per-cpu=500MB

# turn on all mail notification
#SBATCH --mail-type=ALL

# you may not place bash commands before the last SBATCH directive

# define and create a unique scratch directory
SCRATCH_DIRECTORY=/global/work/${USER}/example/${SLURM_JOBID}
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

# we copy everything we need to the scratch directory
# ${SLURM_SUBMIT_DIR} points to the path where this script was submitted from
cp ${SLURM_SUBMIT_DIR}/my_binary.x ${SCRATCH_DIRECTORY}

# we execute the job and time it
time mpirun -np $SLURM_NTASKS ./my_binary.x > my_output

# after the job is done we copy our output back to $SLURM_SUBMIT_DIR
cp ${SCRATCH_DIRECTORY}/my_output ${SLURM_SUBMIT_DIR}

# we step out of the scratch directory and remove it
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}

# happy end
exit 0
```

16.2.3 Example for a hybrid MPI/OpenMP job

```
#!/bin/bash -l

#######################################
# example for a hybrid MPI OpenMP job #
#######################################

#SBATCH --job-name=example

# we ask for 4 MPI tasks with 10 cores each
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=10

# run for five minutes
#               d-hh:mm:ss
#SBATCH --time=0-00:05:00

# 500MB memory per core
# this is a hard limit
#SBATCH --mem-per-cpu=500MB

# turn on all mail notification
#SBATCH --mail-type=ALL

# you may not place bash commands before the last SBATCH directive

# define and create a unique scratch directory
SCRATCH_DIRECTORY=/global/work/${USER}/example/${SLURM_JOBID}
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

# we copy everything we need to the scratch directory
# ${SLURM_SUBMIT_DIR} points to the path where this script was submitted from
cp ${SLURM_SUBMIT_DIR}/my_binary.x ${SCRATCH_DIRECTORY}

# we set OMP_NUM_THREADS to the number cpu cores per MPI task
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

# we execute the job and time it
time mpirun -np $SLURM_NTASKS ./my_binary.x > my_output

# after the job is done we copy our output back to $SLURM_SUBMIT_DIR
cp ${SCRATCH_DIRECTORY}/my_output ${SLURM_SUBMIT_DIR}

# we step out of the scratch directory and remove it
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}

# happy end
exit 0
```

If you want to start more than one MPI rank per node you can use `--ntasks-per-node` in combination with `--nodes`:

```
#SBATCH --nodes=4 --ntasks-per-node=2 --cpus-per-task=8
```

This will start 2 MPI tasks each on 4 nodes, where each task can use up to 8 threads.

CHAPTER 17

SLURM Workload Manager

SLURM is the workload manager and job scheduler used for Stallo.

There are two ways of starting jobs with SLURM; either interactively with `srun` or as a script with `sbatch`.

Interactive jobs are a good way to test your setup before you put it into a script or to work with interactive applications like MATLAB or python. You immediately see the results and can check if all parts behave as you expected. See *Interactive jobs* for more details.

17.1 SLURM Parameter

SLURM supports a multitude of different parameters. This enables you to effectivly tailor your script to your need when using Stallo but also means that is easy to get lost and waste your time and quota.

The following parameters can be used as command line parameters with `sbatch` and `srun` or in jobscript, see *Job script examples*. To use it in a jobscript, start a newline with `#SBTACH` followed by the parameter. Replace <....> with the value you want, e.g. `--job-name=test-job`.

17.1.1 Basic settings:

Parameter	Function
–job-name=<name>	Job name to be displayed by for example `squeue`
–output=<path>	Path to the file where the job (error) output is written to
–mail-type=<type>	Turn on mail notification; type can be one of BEGIN, END, FAIL, REQUEUE or ALL
–mail-user=<email_address>	Email address to send notifications to

17.1.2 Requesting Resources

Parameter	Function
–time=<d-hh:mm:ss>	Time limit for job. Job will be killed by SLURM after time has run out. Format days-hours:minutes:seconds
–nodes=<num_nodes>	Number of nodes. Multiple nodes are only useful for jobs with distributed-memory (e.g. MPI).
–mem=<MB>	Memory (RAM) per node. Number followed by unit prefix, e.g. 16G
–mem-per-cpu=<MB>	Memory (RAM) per requested CPU core
–ntasks-per-node=<num_procs>	Number of (MPI) processes per node. More than one useful only for MPI jobs. Maximum number depends nodes (number of cores)
–cpus-per-task=<num_threads>	CPU cores per task. For MPI use one. For parallelized applications benchmark this is the number of threads.
–exclusive	Job will not share nodes with other running jobs. You will be charged for the complete nodes even if you asked for less.

17.1.3 Accounting

See also *Partitions (queues) and services*.

Parameter	Function
–account=<name>	Project (not user) account the job should be charged to.
–partition=<name>	Partition/queue in which o run the job.
–qos=devel	On stallo the *devel* QOS (quality of servive) can be used to submit short jobs for testing and debugging.

17.1.4 Advanced Job Control

Parameter	Function
–array=<indexes>	Submit a collection of similar jobs, e.g. `--array=1-10`. (sbatch command only). See official SLURM documentation
–dependency=<state:jobid>	Wait with the start of the job until specified dependencies have been satified. E.g. –dependency=afterok:123456
–ntasks-per-core=2	Enables hyperthreading. Only useful in special circumstances.

17.2 Differences between CPUs and tasks

As a new users writing your first SLURM job script the difference between `--ntasks` and `--cpus-per-taks` is typically quite confusing. Assuming you want to run your program on a single node with 16 cores which SLURM parameters should you specify?

The answer is it depends whether the your application supports MPI. MPI (message passing protocol) is a communication interface used for developing parallel computing programs on distributed memory systems. This is necessary for applications running on multiple computers (nodes) to be able to share (intermediate) results.

To decide which set of parameters you should use, check if your application utilizes MPI and therefore would benefit from running on multiple nodes simultaneously. On the other hand you have an non-MPI enables application or made a mistake in your setup, it doesn't make sense to request more than one node.

17.3 Settings for OpenMP and MPI jobs

17.3.1 Single node jobs

For applications that are not optimized for HPC (high performance computing) systems like simple python or R scripts and a lot of software which is optimized for desktop PCs.

Simple applications and scripts

Many simple tools and scripts are not parallized at all and therefore won't profit from more than one CPU core.

Parameter	Function
–nodes=1	Start a unparallized job on only one node
–ntasks-per-node=1	For OpenMP, only one task is necessary
–cpus-per-task=1	Just one CPU core will be used.
–mem=<MB>	Memory (RAM) for the job. Number followed by unit prefix, e.g. 16G

If you are unsure if your application can benefit from more cores try a higher number and observe the load of your job. If it stays at approximately one there is no need to ask for more than one.

OpenMP applications

OpenMP (Open Multi-Processing) is a multiprocessing library is often used for programs on shared memory systems. Shared memory describes systems which share the memory between all processing units (CPU cores), so that each process can access all data on that system.

Parameter	Function
–nodes=1	Start a parallel job for a shared memory system on only one node
–ntasks-per-node=1	For OpenMP, only one task is necessary
–cpus-per-task=<num_threads>	Number of threads (CPU cores) to use
–mem=<MB>	Memory (RAM) for the job. Number followed by unit prefix, e.g. 16G

17.3.2 Multiple node jobs (MPI)

For MPI applications.

Depending on the frequency and bandwidth demand of your setup, you can either just start a number of MPI tasks or request whole nodes. While using whole nodes guarantees that a low latency and high bandwidth it usually results in a longer queuing time compared to cluster wide job. With the latter the SLURM manager can distribute your task across all nodes of stallo and utilize otherwise unused cores on nodes which for example run a 16 core job on a 20 core node. This usually results in shorter queuing times but slower inter-process connection speeds.

We strongly advice all users to ask for a given set of cores when submitting multi-core jobs. To make sure that you utilize full nodes, you should ask for sets that adds up to both 16 and 20 (80, 160 etc) due to the hardware specifics of Stallo i.e. submit the job with `--ntasks=80` **if** your application scales to this number of tasks.

This will make the best use of the resources and give the most predictable execution times. If your job requires more than the default available memory per core (32 GB/node gives 2 GB/core for 16 core nodes and 1.6GB/core for 20 core nodes) you should adjust this need with the following command: `#SBATCH --mem-per-cpu=4GB` When doing this, the batch system will automatically allocate 8 cores or less per node.

To use whole nodes

Parameter	Function
–nodes=<num_nodes>	Start a parallel job for a distributed memory system on several nodes
–ntasks-per-node=<num_procs>	Number of (MPI) processes per node. Maximum number depends nodes (16 or 20 on Stallo)
–cpus-per-task=1	Use one CPU core per task.
–exclusive	Job will not share nodes with other running jobs. You don't need to specify memory as you will get all available on the node.

To distribute your job

Parameter	Function
–ntasks=<num_procs>	Number of (MPI) processes in total. Equals to the number of cores
–mem-per-cpu=<MB>	Memory (RAM) per requested CPU core. Number followed by unit prefix, e.g. 2G

17.3.3 Scalability

You should run a few tests to see what is the best fit between minimizing runtime and maximizing your allocated cpu-quota. That is you should not ask for more cpus for a job than you really can utilize efficiently. Try to run your job on 1, 2, 4, 8, 16, etc., cores to see when the runtime for your job starts tailing off. When you start to see less than 30% improvement in runtime when doubling the cpu-counts you should probably not go any further. Recommendations to a few of the most used applications can be found in *Application guides*.

17.4 Job related environment variables

Here we list some environment variables that are defined when you run a job script. These is not a complete list. Please consult the SLURM documentation for a complete list.

Job number:

```
SLURM_JOBID
SLURM_ARRAY_TASK_ID  # relevant when you are using job arrays
```

List of nodes used in a job:

```
SLURM_NODELIST
```

Scratch directory:

```
SCRATCH  # defaults to /global/work/${USER}/${SLURM_JOBID}.stallo-adm.uit.no
```

We recommend to **not** use $SCRATCH but to construct a variable yourself and use that in your script, e.g.:

```
SCRATCH_DIRECTORY=/global/work/${USER}/my-example/${SLURM_JOBID}
```

The reason for this is that if you forget to sbatch your job script, then $SCRATCH may suddenly be undefined and you risk erasing your entire /global/work/${USER}.

Submit directory (this is the directory where you have sbatched your job):

```
SUBMITDIR
SLURM_SUBMIT_DIR
```

Default number of threads:

```
OMP_NUM_THREADS=1
```

Task count:

```
SLURM_NTASKS
```

17.5 Partitions (queues) and services

SLURM differs slightly from the previous Torque system with respect to definitions of various parameters, and what was known as queues in Torque may be covered by both `--partition=...` and `--qos=...`.

We have the following partitions:

normal: The default partition. Up to 48 hours of walltime.

singlenode: If you ask for less resources than available on one single node, this will be the partition your job will be put in. We may remove the single-user policy on this partition in the future. This partition is also for single-node jobs that run for longer than 48 hours.

multinode: Request this partition if you ask for more resources than you will find on one node and request walltime longer than 48 hrs.

highmem: Use this partition to use the high memory nodes with 128 GB. You will have to apply for access to this partition by sending us an email explaining why you need these high memory nodes.

To figure out the walltime limits for the various partitions, type:

```
$ sinfo --format="%P %l"  # small L
```

As a service to users that needs to submit short jobs for testing and debugging, we have a service called devel. These jobs have higher priority, with a maximum of 4 hrs of walltime and no option for prolonging runtime.

Jobs in using devel service will get higher priority than any other jobs in the system and will thus have a shorter queue delay than regular jobs. To prevent misuse the devel service has the following limitations:

- Only one running job per user.
- Maximum 4 hours walltime.
- Only one job queued at any time, remark this is for the whole queue.

You submit to the devel-service by typing:

```
#SBATCH --qos=devel
```

in your job script.

17.6 General job limitations

The following limits are the default per user in the batch system. Users can ask for increased limits by sending a request to support@metacenter.no.

Limit	Value
Max number of running jobs	1024
Maximum cpus per job	2048
Maximum walltime	28 days
Maximum memory per job	No limit [1]

[1] There is a practical limit of 128GB per compute node used.

Remark: Even if we impose a 28 day run time limit on Stallo we only give a weeks warning on system maintenance. Jobs with more than 7 days walltime, will be terminated and restarted if possible.

See *About Stallo* chapter of the documentation if you need more information on the system architecture.

CHAPTER 18

Interactive jobs

18.1 Starting an interactive job

You can run an interactive job like this:

```
$ srun --nodes=1 --ntasks-per-node=1 --time=01:00:00 --pty bash -i
```

Here we ask for a single core on one interactive node for one hour with the default amount of memory. The command prompt will appear as soon as the job starts.

This is how it looks once the interactive job starts:

```
srun: job 12345 queued and waiting for resources
srun: job 12345 has been allocated resources
```

Exit the bash shell to end the job. If you exceed the time or memory limits the job will also abort.

Interactive jobs have the same policies as normal batch jobs, there are no extra restrictions. You should be aware that you might be sharing the node with other users, so play nice.

Some users have experienced problems with the command, then it has helped to specify the cpu account:

```
$ srun --account=<NAME_OF_MY_ACCOUNT> --nodes=1 --ntasks-per-node=1 --time=01:00:00 --
↪pty bash -i
```

18.2 Keeping interactive jobs alive

Interactive jobs die when you disconnect from the login node either by choice or by internet connection problems. To keep a job alive you can use a terminal multiplexer like `tmux`.

tmux allows you to run processes as usual in your standard bash shell

You start tmux on the login node before you get a interactive slurm session with `srun` and then do all the work in it. In case of a disconnect you simply reconnect to the login node and attach to the tmux session again by typing:

```
tmux attach
```

or in case you have multiple sessions running:

```
tmux list-session
tmux attach -t SESSION_NUMBER
```

As long as the tmux session is not closed or terminated (e.g. by a server restart) your session should continue. One problem with our systems is that the tmux session is bound to the particular login server you get connected to. So if you start a tmux session on stallo-1 and next time you get randomly connected to stallo-2 you first have to connect to stallo-1 again by:

```
ssh login-1
```

To log out a tmux session without closing it you have to press CTRL-B (that the Ctrl key and simultaneously "b", which is the standard tmux prefix) and then "d" (without the quotation marks). To close a session just close the bash session with either CTRL-D or type exit. You can get a list of all tmux commands by CTRL-B and the ? (question mark). See also this page for a short tutorial of tmux. Otherwise working inside of a tmux session is almost the same as a normal bash session.

CHAPTER 19

Managing jobs

The lifecycle of a job can be managed with as little as three different commands:

1. Submit the job with `sbatch <script_name>`.
2. Check the job status with `squeue`. (to limit the display to only your jobs use `squeue -u <user_name>`.)
3. (optional) Delete the job with `scancel <job_id>`.

You can also hold the start of a job:

scontrol hold <job_id> Put a hold on the job. A job on hold will not start or block other jobs from starting until you release the hold.

scontrol release <job_id> Release the hold on a job.

19.1 Job status descriptions in squeue

When you run `squeue` (probably limiting the output with `squeue -u <user_name>`), you will get a list of all jobs currently running or waiting to start. Most of the columns should be self-explaining, but the *ST* and *NODELIST (REASON)* columns can be confusing.

ST stands for *state*. The most important states are listed below. For a more comprehensive list, check the squeue help page section Job State Codes.

R The job is running

PD The job is pending (i.e. waiting to run)

CG The job is completing, meaning that it will be finished soon

The column *NODELIST (REASON)* will show you a list of computing nodes the job is running on if the job is actually running. If the job is pending, the column will give you a reason why it still pending. The most important reasons are listed below. For a more comprehensive list, check the squeue help page section Job Reason Codes.

Priority There is another pending job with higher priority

Resources The job has the highest priority, but is waiting for some running job to finish.

QOS*Limit This should only happen if you run your job with `--qos=devel`. In developer mode you may only have one single job in the queue.

launch failed requeued held Job launch failed for some reason. This is normally due to a faulty node. Please contact us via support@metacenter.no stating the problem, your user name, and the jobid(s).

Dependency Job cannot start before some other job is finished. This should only happen if you started the job with `--dependency=...`

DependencyNeverSatisfied Same as *Dependency*, but that other job failed. You must cancel the job with `scancel JOBID`.

CHAPTER 20

Monitoring your jobs

20.1 SLURM commands

To monitor your jobs, you can use of of those commands. For details run them with the --help option:

`scontrol show jobid -dd <jobid>` lists detailed information for a job (useful for troubleshooting).

`sacct -j <jobid> --format=JobID,JobName,MaxRSS,Elapsed` will give you statistics on completed jobs by jobID. Once your job has completed, you can get additional information that was not available during the run. This includes run time, memory used, etc.

From our monitoring tool Ganglia, you can watch live status information on Stallo:

- Load situation
- Job queue

20.2 CPU load and memory consumption of your job

Stallo has only limited resources and usually a high demand. Therefore using the available resources as efficient as possible is paramount to have short queueing times and getting most out of your quota.

20.2.1 Accessing slurmbrowser

In order to find out the CPU load and memory consumption of your running jobs or jobs which have finished less than 48 hours ago, please use the job browser (accessible from within the UNINETT network).

For users from outside of UNINETT, you have to setup a ssh tunnel to stallo with:

```
ssh -L8080:localhost:80 stallo-login2.uit.no
```

After that you can access slurmbrowser at http://localhost:8080/slurmbrowser/html/squeue.html

20.2.2 Detecting inefficient jobs

You can filter for a slurm job ID, account name or user name with the search bar in the upper left corner.

For single- or multinode jobs the `AvgNodeLoad` is an important indicator if your jobs runs efficiently, at least with respect to CPU usage. If you use the whole node, the average node load should be close to number of CPU cores of that node (so 16 or 20 on Stallo). In some cases it is totally acceptable to have a low load if you for instance need a lot of memory but in general either CPU or memory load should be high. Otherwise you are wasting your quota and experience probably longer than necessary queuing times.

If you detect inefficient jobs you either look for ways to improve the resource usage of your job or ask for less resources in your SLURM script.

20.3 Understanding your job status

When you look at the job queue through the job browser, or you use the `squeue` command, you will see that the queue is divided in 3 parts: Active jobs, Idle jobs, and Blocked jobs.

Active jobs are the jobs that are running at the moment. Idle jobs are next in line to start running, when the needed resources become available. Each user can by default have only one job in the Idle Jobs queue.

Blocked jobs are all other jobs. Their state can be *Idle*, *Hold*, or *Deferred*. *Idle* means that they are waiting to get to the Idle queue. They will eventually start when the resources become available. The jobs with the *Hold* state have been put on hold either by the system, or by the user. F.e. if you have one job in the Idle queue, that is not very important to you, and it is blocking other, more urgent, jobs from starting, you might want to put that one job on hold. Jobs on hold will not start until the hold is released. *Deferred* jobs will not start. In most cases, the job is deferred because it is asking for a combination of resources that Stallo can not provide.

Please contact the support staff, if you don't understand why your job has a hold or deferred state.

20.4 Summary of used resources

Slurm will append a summary of used resources to the `slurm-xxx.out` file. The fields are:

- Task and CPU usage stats
  - `AllocCPUS`: Number of allocated CPUs
  - `NTasks`: Total number of tasks in a job or step.
  - `MinCPU`: Minimum CPU time of all tasks in job (system + user).
  - `MinCPUTask`: The task ID where the mincpu occurred.
  - `AveCPU`: Average CPU time of all tasks in job (system + user)
  - `Elapsed`: The jobs elapsed time in format [DD-[HH:]]MM:SS.
  - `ExitCode`: The exit code returned by the job script. Following the colon is the signal that caused the process to terminate if it was terminated by a signal.
- Memory usage stats
  - `MaxRSS`: Maximum resident set size of all tasks in job.
  - `MaxRSSTask`: The task ID where the maxrss occurred.
  - `AveRSS`: Average resident set size of all tasks in job.
  - `MaxPages`: Maximum number of page faults of all tasks in job.

  - `MaxPagesTask`: The task ID where the maxpages occurred.
  - `AvePages`: Average number of page faults of all tasks in job.
- Disk usage stats
  - `MaxDiskRead`: Maximum number of bytes read by all tasks in job.
  - `MaxDiskReadTask`: The task ID where the maxdiskread occurred.
  - `AveDiskRead`: Average number of bytes read by all tasks in job.
  - `MaxDiskWrite`: Maximum number of bytes written by all tasks in job.
  - `MaxDiskWriteTask`: The task ID where the maxdiskwrite occurred.
  - `AveDiskWrite`: Average number of bytes written by all tasks in job.

CHAPTER 21

Running MPI jobs

There are two available MPI implementations on Stallo:

- OpenMPI provided by the `foss` module, e.g. `module load foss/2016b`
- Intel MPI provided by the `intel` module, e.g. `module load intel/2016b`

There are several ways of launching an MPI application within a SLURM allocation, e.g. `srun`, `mpirun`, `mpiexec` and `mpiexec.hydra`. Unfortunately, the best way to launch your program depends on the MPI implementation (and possibly your application), and choosing the wrong command can severly affect the efficiency of your parallel run. Our recommendation is the following:

21.1 Intel MPI

With Intel MPI, we have found that `mpirun` can incorrectly distribute the MPI ranks among the allocated nodes. We thus recommend using `srun`:

```
$ srun --mpi=pmi2 ./my_application
```

21.2 OpenMPI

With OpenMPI, `mpirun` seems to be working correctly. Also, it seems that `srun` fails to launch your application in parallel, so here we recommend using `mpirun`:

```
$ mpirun ./my_application
```

NOTE: If you're running on the `multinode` partition you automatically get the `--exclusive` flag, e.i. you get allocated (and charged for) **full** nodes, even if you explicitly ask for less resources per node. This is not the case for the `normal` partition.

CHAPTER 22

Which software is installed on Stallo

To find out what software packages are available, type:

```
module avail
```

22.1 Changes in application software

News about planned/unplanned downtime, changes in hardware, and important changes in software will be published on the HPC UiT twitter account https://twitter.com/hpc_uit and on the login screen of stallo. For more information on the different situations see *news*.

The easiest way to check which software and versions available is to use the `module` command. List all software available:

```
module avail
```

List all version of a specific software:

```
module avail software-name
```

CHAPTER 23

Missing or new software

If there is any software missing on this list that you would like to have installed on Stallo, or you need help to install your own software, please feel free to contact the support personal about this: support@metacenter.no.

If there is software missing on Stallo that you would like to have, there are different options to install it: * Use our installation framework EasyBuild * Use a package manager like Conda or especially for bioinformatic software Bioconda * Compile it yourself and install into your home directory * Ask us to install for you

In this document we will show some ways of installing a new software. If you run into problems, need help or have question, please feel free to contact the support personal about this: support@metacenter.no.

23.1 Installing new software using EasyBuild

Our building and installation framework, EasyBuild can be used by you to install software into your own home directory. This is especially interesting for you if you need a software but it is probably not used by other users of Stallo, so a systemwide installation doesn't make much sense.

To see if your software is available via the EasyBuild system have a look into the easyconfig github repository. if you can find a version of your software there. Alternatively use the search function of EasyBuild:

```
eb -S REGEX
```

Let's assume you want to install the genome aligner Kraken2. We find a easyconfig file with the name Kraken2-2.0.7-beta-foss-2018b-Perl-5.28.0.eb in the easyconfig repository.

Before we start the installation process we first have to load the necessary modules:

```
ml purge
ml EasyBuild
```

Before we actually build we can check the building process by adding *–dry-run-short* or *-D* to the build command:

```
eb Kraken2-2.0.7-beta-foss-2018b-Perl-5.28.0.eb --dry-run-short
```

As you probably see, a long list of packages is printed with some packages being marked as missing. To automatically install all necessary dependencies we can add *–robot* and then run the installation:

```
eb Kraken2-2.0.7-beta-foss-2018b-Perl-5.28.0.eb --robot
```

Before we can load your newly installed module, we have to add the path to the Lmod search path. You might have to do this from time to time, or even every time you use your software, as your local cache is regularly updated.

```
ml use ~/.local/easybuild/modules/all
ml Kraken2/2.0.7
```

23.2 Conda/ Bioconda

Many software packages, especially if they are python based, can be easily installed using the Conda package manager. For many bioinformatics software, Bioconda has become a nice solution.

A small tutorial can be found in the *Python* section of this documentation.

CHAPTER 24

Software Module Scheme

Since a HPC cluster is shared among many users, and also holds a significant size in contrast to most desktop compute machinery around, the amount of installed software spans many applications in many different versions and quite a few of them are installed typically non-standard places for easier maintenance (for admin crew), practical and security reasons. It is not possible (nor desirable) to use them all at the same time, since different versions of the same application may conflict with each other. Therefore, it is practical to provide the production environment for a given application outside of the application itself. This is done using a set of instructions and variable settings that are specific for the given application called an application module. This also simplifies control of which application versions are available in a specific session.

We are also using the Easybuild system for installing software on Stallo (see https://easybuilders.github.io/easybuild/).

The main command for using this system is the *module* command. You can find a list of all its options by typing:

```
module --help
```

We use the lmod module system; for more info see https://lmod.readthedocs.io/en/latest/ on Stallo. Below we listed the most commonly used options, but also feel free to ivestigate options in this toolset more thoroughly on developers site.

24.1 Which modules are currently loaded?

To see the modules currently active in your session, use the command:

```
module list
```

24.2 Which modules are available?

In order to see a complete list of available modules, issue the command:

```
module avail
```

The resulting list will contain module names conforming to the following pattern:

- name of the module
- /
- version
- (default) if default version

24.3 How to load a module

In order to make, for instance, the NetCDF library available issue the command:

```
module load netCDF
```

This will load the default NetCDF version. To load a specific version, just include the version after a slash:

```
module load netCDF/4.3.3.1-intel-2016a
```

24.4 How to unload a module

Keeping with the above example, use the following command to unload the NetCDF module again:

```
module unload netcdf
```

24.5 Which version of a module is the default?

In order to see, for instance, which version of NetCDF is loaded by the module, use:

```
module show netcdf
```

This will show the version and path of the module.

24.6 How do I switch to a different version of a module?

Switching to another version is similar to loading a specific version. As an example, if you want to switch from the default (current) Intel compilers version to version 2016b (aka 2016.3), type:

```
module switch intel intel/2016b
```

Beware: by default on Stallo, you are not able to have two conflicting modules - f.e. openmpi and impi - loaded at the same time. This is only partially true for these new packages. With the new packages it is also more common that when you load one package it will automatically load several other packages that it needs to function. We recommend doing ml after every load and unloading any conflicting packages.

CHAPTER 25

Python, R, Matlab and Perl

Scripting languages often support modules or libraries for additional functionality or convenience functions. We encourage users to install modules locally for only the current user.

25.1 Python

You can install many python and non-python packages yourself using conda or especially for bioinformatics software bioconda.

Conda enables you to easily install complex packages and software. Creating multiple enviroments enables you to have installations of the same software in different versions or incompatible software collections at once. You can easily share a list of the installed packages with collaborators or colleagues, so they can setup the same enivironment in a matter of minutes.

25.1.1 Setup

First you load the miniconda module which is like a python and r package manager. Conda makes it easy to have multiple environments for example one python2 and one python3 based parallel to each other without interfering.

Start by removing all preloaded modules which can complicate things. We then display all installed version and load the newest one (4.6.14):

```
ml purge
ml avail Miniconda
ml Miniconda3/4.6.14
```

To install packages we first have to add the package repository to conda (we only have to do this once) and set up a new conda environment which we will call e.g. “python3” where we also specify which python version we want and which packages should be installed (matplotlib numpy):

```
conda config --add channels defaults
conda config --add channels conda-forge

conda create --name python3 python=3 matplotlib numpy
```

If you want install bioinformatics packages you should also add the bioconda channel:

```
conda config --add channels bioconda
```

In case you want to install the conda environment in another directory than the home, you can add *–prefix PATH*. This enables multiple users of a project to share the conda environment by installing it into their project folder instead of the users home.

To suppress the warning that a newer version of conda exists which is usually not important for most users and will be fixed by us by installing a new module:

```
conda config --set notify_outdated_conda false
```

25.1.2 Daily usage

To load this environment you have to use the following commands either on the command line or in your job script:

```
ml purge
ml Miniconda3/4.6.14
conda activate python3
```

Then you can use all software as usual.

To deactivate the current environment:

```
conda deactivate
```

If you need to install additional software or packages, we can search for it with:

```
conda search SOMESOFTWARE
```

and install it with:

```
conda install -n python3 SOMESOFTWARE
```

If the python package you are looking for is not available in conda you can use **pip** like usually from within a conda environment to install additional python packages:

```
pip install SOMEPACKAGE
```

To update the a single package with conda:

```
conda update -n python3 SOMESOFTWARE
```

or to update all packages:

```
conda update -n python3 --all
```

25.1.3 Share your environment

To export a list of all packages/programs installed with conda in a certain environment (in this case "python3"):

```
conda list --explicit --name python3 > package-list.txt
```

To setup a new environment (let's call it "newpython") from ab exported package list:

```
conda create --name newpython --file package-list.txt
```

25.1.4 Additional Conda information

Cheatsheet and built-in help

See this cheatsheet for an overview over the most important conda commands.

In case you get confused by the conda commands and command line options you can get help by adding *–help* to any conda command or have a look at the conda documentation.

Miniconda vs. Anaconda

Both Miniconda and Anaconda are distributions of the conda repository managment system. But while Miniconda brings just the managment system (the conda command), Anaconda comes with a lot of built-in packages.

Both are installed on Stallo but we advise the use of Miniconda. By explicitly installing packages into your own enviroment the chance for unwanted effects and errors due to wrong or incomaptible versions is reduced. Also you can be sure that everything that happens with your setup is controlled by yourself.

25.1.5 Virtual Environments (deprecated)

We recommend using Conda , but there is another way to install modules locally is using virtual environments

As an example we install the Biopython package (and here we use the `Python/3.6.4-intel-2018a` module as an example):

```
$ module load Python/3.6.4-intel-2018a
$ virtualenv venv
$ source venv/bin/activate
$ pip install biopython
```

Next time you log into the machine you have to activate the virtual environment:

```
$ source venv/bin/activate
```

If you want to leave the virtual environment again, type:

```
$ deactivate
```

And you do not have to call it "venv". It is no problem to have many virtual environments in your home directory. Each will start as a clean Python setup which you then can modify. This is also a great system to have different versions of the same module installed side by side.

If you want to inherit system site packages into your virtual environment, do this instead:

```
$ virtualenv --system-site-packages venv
$ source venv/bin/activate
$ pip install biopython
```

25.2 R

25.2.1 Load R

Using R on Stallo is quite straightforward. First check which versions are available:

```
ml avail -r '^R/'
```

To load a version:

```
ml R/3.5.0-iomkl-2018a-X11-20180131
```

Now you can use R from the command line just as you would on your local computer.

25.2.2 Install Packages

To install R packages use `install.packages()`. First open the R command line and then install a package e.g. "tidyverse":

```
R
install.packages("tidyverse")
```

Note: The first time you install new packages, R will ask you whether it should install these packages into your home folder. Confirm both questions with `y` and then choose a close download mirror

25.3 MATLAB

25.3.1 Load MATLAB

To use MATLAB simply load the module at the start of your jobscript or type them on the command line:

```
ml purge
ml avail matlab # To display all installed versions
ml MATLAB/R2018a-foss-2017a # or any other version you want
```

25.3.2 Interactice Shell

On the login nodes you can start a normal MATLAB session with an graphical user interface (GUI). You can use this to visualize and look at data. Just type `matlab`.

But remember NOT to run calculations on the login nodes as this might slow down the system for all stallo users. If this happens we will kill the process without prior warning.

You can also start an interactive matlab shell on the command line without graphical user interface (headless) with:

```
matlab -nodesktop -nodisplay -nosplash
```

See `matlab -h` for all command line options. If you are on a compute node `matlab` always starts a headless matlab shell.

25.3.3 Running MATLAB Scripts

You can run a matlab script by:

```
matlab -r -nodisplay -nosplash -r 'run("SCRIPT.m")'
```

In some instances it might be necessary to use an absolute file path to the script.

25.3.4 Tips

- You can reduce the memory usage by starting matlab without java support, just add `-nojvm`.
- To get a graphical interface when starting `matlab` on a login node, you need to activate X11 forwarding for your ssh connection to stallo. If you connect to stallo from a linux machine use `ssh -X` to tunnel graphical output to your computer.

25.4 Perl

We will use Perl 5.28 and use the standard paths. This follows the general instruction given here: https://metacpan.org/pod/local::lib.

```
$ module load Perl/5.28.0-GCCcore-7.3.0
$ mkdir my_perl_installs   # or however you want to call this temporary folder
$ cd my_perl_installs

# Check the newest version on metacpan.org and search for local::lib
$ wget https://cpan.metacpan.org/authors/id/H/HA/HAARG/local-lib-2.000024.tar.gz

$ tar xzf local-lib-2.000024.tar.gz
$ cd local-lib-2.000024
$ perl Makefile.PL --bootstrap
$ make test
$ make install
$ echo 'eval "$(perl -I$HOME/perl5/lib/perl5 -Mlocal::lib)"' >> ~/.bashrc
$ source ~/.bashrc
```

Now, the module local::lib is installed and the `~/.bashrc` changed such that Perl should now recognize your local folder as module folder. All future modules will be installed to `~/perl5`.

If you want to install, for example, the module Math::Vector::Real, just call cpan:

```
$ cpan Math::Vector::Real
```

Remember to load the right Perl version first (`module load ...`). The first time you call cpan, it will ask you to do some configurations. Just press enter (let it do its configurations).

CHAPTER 26

Linux command line

26.1 New on Linux systems?

This page contains some tips on how to get started using the Stallo cluster on UiT if you are not too familiar with Linux/Unix. The information is intended for both users that are new to Stallo and for users that are new to Linux/UNIX-like operating systems. Please consult the rest of the user guide for information that is not covered in this chapter.

For details about the hardware and the operating system of stallo, and basic explanation of Linux clusters please see the *About Stallo* part of this documentation.

To find out more about the storage and file systems on the machines, your disk quota, how to manage data, and how to transfer file to/from Stallo, please read the *Transferring files to/from Stallo* section.

26.1.1 ssh

The only way to connect to Stallo is by using ssh. Check the *Logging in for the first time* in this documentation to learn more about ssh.

26.1.2 scp

The machines are stand-alone systems. The machines do not (NFS-)mount remote disks. Therefore you must explicitly transfer any files you wish to use to the machine by scp. For more info, see *Transferring files to/from Stallo*.

26.1.3 Running jobs on the machine

You must execute your jobs by submitting them to the batch system. There is a dedicated section *Batch system* in our user guide that explain how to use the batch system. The pages explains how to write job scripts, and how to submit, manage, and monitor jobs. It is not allowed to run long or large memory jobs interactively (i.e., directly from the command line).

26.1.4 Common commands

pwd Provides the full pathname of the directory you are currently in.

cd Change the current working directory.

ls List the files and directories which are located in the directory you are currently in.

find Searches through one or more directory trees of a file system, locates files based on some user-specified criteria and applies a user-specified action on each matched file. e.g.:

```
find . -name 'my*' -type f
```

The above command searches in the current directory (.) and below it, for files and directories with names starting with my. "-type f" limits the results of the above search to only regular files, therefore excluding directories, special files, pipes, symbolic links, etc. my* is enclosed in single quotes (apostrophes) as otherwise the shell would replace it with the list of files in the current directory starting with "my".

grep Find a certain expression in one or more files, e.g:

```
grep apple fruitlist.txt
```

mkdir Create new directory.

rm Remove a file. Use with caution.

rmdir Remove a directory. Use with caution.

mv Move or rename a file or directory.

vi/vim or emacs Edit text files, see below.

less/ more View (but do not change) the contents of a text file one screen at a time, or, when combined with other commands (see below) view the result of the command one screen at a time. Useful if a command prints several screens of information on your screen so quickly, that you don't manage to read the first lines before they are gone.

| Called "pipe" or "vertical bar" in English. Group 2 or more commands together, e.g.:

```
ls -l | grep key | less
```

This will list files in the current directory (ls), retain only the lines of *ls* output containing the string "key" (grep), and view the result in a scrolling page (less).

26.1.5 More info on manual pages

If you know the UNIX-command that you would like to use but not the exact syntax, consult the manual pages on the system to get a brief overview. Use 'man [command]' for this. For example, to get the right options to display the contents of a directory, use 'man ls'. To choose the desired options for showing the current status of processes, use 'man ps'.

26.1.6 Text editing

Popular tools for editing files on Linux/UNIX-based systems are 'vi' and 'emacs'. Unfortunately the commands within both editors are quite cryptic for beginners. It is probably wise to spend some time understanding the basic editing commands before starting to program the machine.

vi/vim Full-screen editor. Use 'man vi' for quick help.

emacs Comes by default with its own window. Type 'emacs -nw' to invoke emacs in the active window. Type 'Control-h i' or follow the menu 'Help->manuals->browse-manuals-with-info' for help. 'Control-h t' gives a tutorial for beginners.

26.1.7 Environment variables

The following variables are automatically available after you log in:

```
$USER      your account name
$HOME      your home directory
$PWD       your current directory
```

You can use these variables on the command line or in shell scripts by typing $USER, $HOME, etc. For instance: 'echo $USER'. A complete listing of the defined variables and their meanings can be obtained by typing 'printenv '.

You can define (and redefine) your own variables by typing:

```
export VARIABLE=VALUE
```

26.1.8 Aliases

If you frequently use a command that is long and has for example many options to it, you can put an alias (abbreviation) for it in your `~/.bashrc` file. For example, if you normally prefer a long listing of the contents of a directory with the command 'ls -laF | more', you can put following line in your `~/.bashrc` file:

```
alias ll='ls -laF | more'
```

You must run 'source ~/.bashrc' to update your environment and to make the alias effective, or log out and in :-). From then on, the command 'll' is equivalent to 'ls -laF | more'. Make sure that the chosen abbreviation is not already an existing command, otherwise you may get unexpected (and unwanted) behavior. You can check the existence and location of a program, script, or alias by typing:

```
which [command]
whereis [command]
```

26.1.9 ~/bin

If you frequently use a self-made or self-installed program or script that you use in many different directories, you can create a directory ~/bin in which you put this program/script. If that directory does not already exist, you can do the following. Suppose your favorite little program is called 'myscript' and is in your home ($HOME) directory:

```
mkdir -p $HOME/bin
cp myscript $HOME/bin
export PATH=$PATH:$HOME/bin
```

PATH is a colon-separated list of directories that are searched in the order in which they are specified whenever you type a command. The first occurrence of a file (executable) in a directory in this PATH variable that has the same name as the command will be executed (if possible). In the example above, the 'export' command adds the ~/bin directory to the PATH variable and any executable program/script you put in the ~/bin directory will be recognized as a command. To add the ~/bin directory permanently to your PATH variable, add the above 'export' command to your ~/.bashrc file and update your environment with 'source ~/.bashrc'.

Make sure that the names of the programs/scripts are not already existing commands, otherwise you may get unexpected (and unwanted) behaviour. You can check the contents of the PATH variable by typing:

```
printenv PATH
echo $PATH
```

CHAPTER 27

General comments about licenses

This is an attempt to describe the different licenses we have on Stallo and for the entire NOTUR system when it comes to chemistry and material science codes.

Here, I will only address the codes with a certain access limitation. For free, open source codes without any limitations, I do not really see the point in commenting. But, the reader should be aware that "free" not allways means "unrestricted access".

Also, due to a choice of policy, I am only adressing the issues of academic access. For commercial licenses, the user or user groups needs to investigate consequences and prices themselves. For any given code, the license model and related issues is more in depth described as a part of the standard documentation.

There is in general 6 different categories of access limitations for software on Stallo:

1. Licenses paid entirely by the users.
2. Licenses paid partially by the users and partly by national bodies.
3. Licenses paid partly or entirely by institutions.
4. Licenses paid entirely by national bodies.
5. Licenses available only on a machine or a service (machine-license).
6. Free licenses that requires an agreement with the vendor before using the code.

Category 1 is what is the "bring your own license" group. Either the vendor does not provide computer centre license, or pricing is somewhat outside the boundaries of acceptance for NOTUR. VASP is in this category, where NOTUR does administer a certain level of access based on feedback from the VASP vendor. Other examples is Petrel, Molpro (previously).

Category 2 is a joint funding model, where users have to add to the total pool of finance in order to provide a certain level of license tokens available. For instance, the Schrodinger small molecule drug discovery suite license is split into a "national" and a "local" part, with one common license server and unlimited tokens available for everyone. NOTUR provides support and manages the license tokens on behalf of the community, up until now as a part of DS Chem (predecessor to the Application Managment project). The license is monitored, in order to ensure a minimum level of fairness. If a user crosses a certain level of usage without representing one of the groups contributing to funding - access will be denied until finance formalities has been settled.

Category 3 is the group of software where your respective host institution has implication for access or not. Codes like COMSOL, MATLAB, STATA, ANSYS, STAR-CCM+, IDL, Mathematica are all in this category.

Category 4 is a lot more complex than you might anticipate. Codes like ADF and Turbomole are in this category. ADF is currently entirely funded by Sigma2, but how long this will continue is unknown. For ADF, we have a national license - meaning that anyone can request a download and install the code suite as long as they are members on a Norwegian Grant degreeing institution. Turbomole has currently the form of a national machine type license, as a part of prevoiously mentioned DS Chem. As long as on person has the responsibility of installing Turbomole on all the national academic high performance compute clusters, we only have to by one common license for all these machines. Currently, the national body has paid for a two group Crystal license, in order to map interest and need. We have (or at least had when the license was bought) an agreement that if the interest came to the level where we needed to grant access to more people, we could transform the license into a machine type license by only paying the difference in license fee. Gaussian has the appearance of a national license, and is paid by NOTUR - but it really is 4 site licenses making a total of "access license for those who comes from one of the four original norwegian academic hpc host sites - NTNU, UiB, UiO, UiT". It also means that others that do hold a valid Gaussian license will be allowed access. This licensing is limited to main version (G09, G16 etc) and may explain limitations on the more recent versions. The intel tools and compiler suites for use on the national academic hpc cluster is also in this category.

Category 5 is the group of software that is only available on a single installation/machine. On Stallo we have NBO6 and the nbo6 plug in for ADF. On Abel they have an installation of Molpro which is both machine type and site type licensed.

Category 6 In this group, we have Molcas, ORCA. (and maybe Dalton and Dirac). Some vendors, even if they offer their software free for academic usage in general, or for limited areas (Molcas is free for academics from Nordic institutions), they still want to have a certain control over the software distribution.

and, of course, category 7 is the entire group of software where this is not really a problem;-).

You will find more info in the application guides or in the respective module files. The latter info you get access to by typing:

CHAPTER 28

New software and module scheme

On the 9th-10th of May 2017, the default module setup changed. It is likely that you will have problems loading modules and jobs waiting to start might crash. Jobs already running should not be affected.

What this means: We have many new software packages installed, that were not visible by default, though they were already accessible. On the afore mentioned days we made these packages visible by default. There are several big changes that you will experience, and sadly this might break your job scripts and cause you error messages when trying to load modules. Jobs that are waiting to start might crash at start because they can not load modules, but already running jobs should run without problems. We apologize for these problems and we believe that this way of switching is still a better alternative than shutting Stallo down.

If you have trouble or questions, please, check this manual and contact us.

Here are the main changes you will experience:

28.1 Change of software module names

You will still be able to use the "old" software you used before, but many names of sw packages will change. The name changes are in using mixed case instead of just small case, f.e. "python" is will become "Python", "openmpi" becomes "OpenMPI", and so on. The version of the modules has also changed. In most modules the version now contains some basic information on what it was compiled with - f.e. intel/2016a or foss/2016a (Free Open Source Software, commonly referred to as GNU) - and in some cases also information on some dependency - f.e. Python/2.7.12. In the new installation, the name "intel" has a somewhat different meaning from the old installations: loading intel/13.0 will give you the intel module, and you will have icc and ifort available to you; loading intel/2016a will load several packages including icc, ifort, imkl and impi. More information on this coming soon.

If you are looking for a package, try:

```
$ module avail <package_name>
```

This command is case insensitive, and lists you all modules, which you can load for this package (both from the new and the old installations). If you get too many results because the name of your package is part of other names (f.e. "R", "intel", or "Python") add a "/" at the end of the name:

```
$ module avail <package_name>/
```

28.2 Behavior of potentially conflicting modules

Another important change is behavior of potentially conflicting modules. In the "old setup" you are not able to have two conflicting modules - f.e. openmpi and impi - loaded at the same time. This is only partially true for the "new scheme" and you will need to pay more attention to this. With the new packages, it is also more common that when you load one package it will automatically load several other packages that it needs to function. Check what you have currently loaded with:

```
$ module load
```

or `ml` for short, and unload conflicting modules, or do:

```
$ module purge
```

to remove all loaded packages and start over.

28.3 StdEnv and StdMod

By default, you will have StdEnv and StdMod loaded. StdEnv sets up the paths and other variables for you to have a working module environment. It has a "sticky" tag and will not be unloaded with a `module purge` or a `module unload StdEnv`. We strongly recommend keeping it loaded. StdMod loads the default system modules. You are welcome to unload it, if you wish, and both `module unload StdMod` or `module purge` will unload it.

28.4 Lmod warning "rebuild your saved collection"

Lmod allows a user to save a bundle of modules as a collection using `module save <collection_name>` and `module restore <collection_name>`. This enables you to quickly get the same list of modules loaded if you tend to use the same modules over and over. With a new module scheme came a different system MODULEPATH. For this reason, if you have some module collections saved, you will experience the following warning: "Lmod Warning: The system MODULEPATH has changed: please rebuild your saved collection." To solve this you need to remove your old collections and create them again. We apologize for the inconvenience.

28.5 If you are already using "mod_setup.sh" or "/home/easybuild/"

If you have already started using the new modules by loading the "notur" module and sourcing "mod_setup.sh", simply remove these two steps and continue working as normal. The same goes for any references to /home/easybuild/. We kept these options working for now to reduce the number of potentially crashing jobs, but we plan to remove them in the near future, so we recommend removing those lines from any new jobs you launch. You do not need the module "varset" any more, StdEnv has taken over its functionality. If the only reason you were using the "notur" module was to access the newer installations, you do not need it either. If you were using the "notur" module to run ADF or other software, then you still need to continue using it.

CHAPTER 29

Application guides

For a general explanation on how to make an application available for use, the module system, and information about changes in application software see *Software Module Scheme*.

29.1 The Amsterdam Density Functional modeling suite

This page contains general information about the ADF modeling suite installed on Stallo:

29.1.1 Related information

First time you run an ADF job?

This page contains info aimed at first time users of ADF on Stallo, but may also be usefull to more experienced users. Please look carefully through the provided examples. Also note that the job-script example is rather richly commented to provide additional and relevant info.

If you want to run this testjob, download the copies of the scripts and put them into your test job folder (which I assume you have created in advance).

ADF input example

```
title Caffeine for scale testing
integration  6.0
units
length angstrom
end
symmetry  Nosym
atoms cartesian
C     1.179579     0.000000    -0.825950
C     2.359623     0.000000     0.016662
```

(continues on next page)

(continued from previous page)

```
C      2.346242      0.000000      1.466600
N      1.092573      0.000000      2.175061
C      0.000000      0.000000      1.440000
N      0.000000      0.000000      0.000000
N      3.391185      0.000000      1.965657
C      4.217536      0.000000      1.154419
N      3.831536      0.000000      0.062646
C      4.765176     -0.384157     -0.964164
C      1.058378     -0.322767      3.578004
O     -1.345306      0.000000      1.827493
C     -1.260613     -0.337780     -0.608570
O      1.192499      0.000000     -2.225890
H     -1.997518     -0.535233      0.168543
H     -1.598963      0.492090     -1.227242
H     -1.138698     -1.225644     -1.227242
H      0.031688     -0.271264      3.937417
H      1.445570     -1.329432      3.728432
H      1.672014      0.388303      4.129141
H      4.218933     -0.700744     -1.851470
H      5.400826      0.464419     -1.212737
H      5.381834     -1.206664     -0.604809
H      5.288201      0.000000      1.353412
end

BASIS
Type TZ2P
End

geometry
end
scf
end
unrestricted

charge 0 0

xc
gga PW91
end

noprint frag sfo

endinput
```

This file can also be downloaded here: `Caffeine input for ADF`.

Place this file in a job folder of choice, say ADFFIRSTJOB in your home directory on Stallo.

then, copy the job-script as seen here:

```
#!/bin/bash -l

################### ADF Job Batch Script Example ###################
# Section for defining queue-system variables:
#-------------------------------------
# This script asks for a given set of cores nodes and cores. Stallo has got 16 or 20␣
↪cores/node,
```

(continues on next page)

(continued from previous page)

```
# asking for something that adds up to both is our general recommodation (80, 160␣
↪etc), you would
# then need to use --ntasks instead of --nodes and --ntasks-per-node. (replace both).
# Runtime for this job is 59 minutes; syntax is hh:mm:ss.
# Memory is set to the maximum advised for a full node, 1500MB/core - giving a total
# of 30000MB/node and leaving some for the system to use. Memory
# can be specified pr core, virtual or total pr job (be carefull).
#-------------------------------------
# SLURM-section
#SBATCH --job-name=adf_runex
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=20
#SBATCH --time=00:59:00
#SBATCH --mem-per-cpu=1500MB # Be ware of memory needs, might be a lot higher if you␣
↪are running Zora basis, for example.
#SBATCH --output=adf_runex.log
#SBATCH --mail-type=ALL
#SBATCH --exclusive
#SBATCH --partition=multinode

######################################
# Section for defining job variables and settings:

input=caffeine # Name of input without extention
ext=adf # We use the same naming scheme as the software default, also for extention
cores=$SLURM_NTASKS  # Number of cores potentially used by mpi engine in submit␣
↪procedure

# We load all the default program system settings with module load:

module --quiet purge
module load ADF/adf2017.108

# Now we create working directory and temporary scratch for the job(s):
# Necessary variables are defined in the notur and the software modules.

export SCM_TMPDIR=/global/work/$USER/$SLURM_JOBID
mkdir -p $SCM_TMPDIR

# Preparing and moving inputfiles to tmp:

submitdir=$SLURM_SUBMIT_DIR

cp ${input}.${ext} $SCM_TMPDIR
cd $SCM_TMPDIR

# In case necessary, set SCM_IOBUFFERSIZE
#export SCM_IOBUFFERSIZE=1024 # Or higher if necessary.

######################################
# Section for running the program and cleaning up:

# Running the program:

time adf -n $cores < ${input}.${ext} > adf_$input.out

# Cleaning up and moving files back to home/submitdir:
```

(continues on next page)

(continued from previous page)

```
# Make sure to move all essential files specific for the given job/software.

cp adf_$input.out $submitdir/adf_$input.out
cp TAPE21 $submitdir/$input.t21

# To zip some of the output might be a good idea!
#gzip $input.t21
#mv $input.t21.gz $submitdir/

# Investigate potentially other files to keep:
echo $(pwd)
echo $(ls -ltr)

# ALWAYS clean up after yourself. Please do uncomment the following line
#cd $submitdir
#rm -r $tempdir/*

echo "Job finished at"
date
################### Job Ended ###################
exit 0
```

(Can also be downloaded from here: `job_adf.sh`)

Place this script in the same folder and type:

```
sbatch job_adf.sh
```

You may now have submitted your first adf job.

These files are also available on Stallo

```
module load ADF/adf2017.108
cd <to whatever you call your test folder> # for instance ADFFIRSTJOB
cp -R /global/hds/software/notur/apprunex/ADF/* .
```

To verify that the jobs has worked fine, check the outputfile. If it says EXIT and print Bond Energies at the end of the file, you are likely on the safe side.

Check the Bond Energy in adf_caffeine.out. The value should ideally be close to -156.75317227 eV. When this is the case, you may alter variables in the shell script as much as you like to adapt to your own jobs.

NB: ADF is installed as precompiled binaries, they come with their own mpi (intel MPI). So if you are not using the provided runscript example and/or loading the newer modules - please make sure that you load an intelmpi module and also preferably a rather recent intel compiler.

Also note that, on Stallo, we hold the nbo6 plug in license that allows users of adf to produce files that can be read with the nb06 software.

Good luck with your chemistry!

Amsterdam Density Functional program

Information regarding the quantum chemistry code ADF on Stallo

General Information

Description

According to the vendor, ADF (Amsterdam Density Functional) is a DFT program particularly strong in understanding and predicting structure, reactivity, and spectra of molecules. It is a Fortran program for calculations on atoms and molecules (in gas phase or solution). It can be used for the study of such diverse fields as molecular spectroscopy, organic and inorganic chemistry, crystallography and pharmacochemistry.

The underlying theory is the Kohn-Sham approach to Density-Functional Theory (DFT). The software is a DFT-only first-principles electronic structure calculations program system, and consists of a rich variety of packages.

Online documentation from vendor

- Documentation: https://www.scm.com/doc/ADF

Usage

The ADF/BAND suite of software is currently installed as precompiled binaries on Stallo. We install the intel-mpi version, since it has proven to collaborate the better with our mpi setup. We generally advise to run ADF on more than one node, unless you do know that your particular problem does not make the code scale well.

Use

```
$ module avail ADF
```

to see which versions of ADF which are available. Use

```
$ module load ADF/<version> # i.e adf2017.108
```

to get access to any given version of ADF.

The periodic DFT code Band

Information regarding the periodic DFT code Band on Stallo. You may also be interested in this: *First time you run a Band job?*.

General Information

Description

BAND is an atomic-orbital based DFT program for periodic systems (crystals, slabs, chains and molecules).

The Amsterdam Density Functional Band-structure program - BAND - can be used for calculations on periodic systems, i.e. polymers, slabs and crystals, and is supplemental to the molecular ADF program for non-periodic systems. It employs density functional theory in the Kohn-Sham approach. BAND is very similar to ADF in the chosen algorithms, although important differences remain.

BAND makes use of atomic orbitals, it can handle elements throughout the periodic table, and has several analysis options available. BAND can use numerical atomic orbitals, so that the core is described very accurately. Because of the numerical orbitals BAND can calculate accurate total energies. Furthermore it can handle basis functions for arbitrary l-values.

Online info from vendor

- Documentation: https://www.scm.com/doc/BAND

Usage

The ADF/BAND suite of software is currently installed as precompiled binaries on Stallo. We install the intel-mpi version, since it has proven to collaborate the better with our mpi setup. We generally advise to run Band on more than one node, unless you do know that your particular problem does not make the code scale well.

Use

```
$ module avail ADF
```

to see which versions of Band which are available. Use

```
$ module load ADF/<version> # i.e adf2017.108
```

to get access to any given version of Band.

First time you run a Band job?

This page contains info aimed at first time users of Band on Stallo, but may also be usefull to more experienced users. Please look carefully through the provided examples. Also note that the job-script example is rather richly commented to provide additional and relevant info.

If you want to run this testjob, download the copies of the scripts and put them into your test job folder (which I assume you have created in advance).

Band input example

```
TITLE Untitled

NumericalQuality Normal

UNITS
    length Angstrom
    angle Degree
END

ATOMS
    Si -0.67875 -0.67875 -0.67875
    Si 0.67875 0.67875 0.67875
END

Lattice
  0.0   10.860 10.860
 10.860  0.0   10.860
 10.860 10.860  0.0
End

DOS
Enabled True
```

(continues on next page)

(continued from previous page)

```
End

BasisDefaults
BasisType DZ
Core Large
End

XC
LDA SCF VWN
END

GeoOpt
UseVariableTrustRadius false
End

BZSTRUCT
    Enabled True
END

end input
```

This file can also be downloaded here: `Silicone input for Band`.

Place this file in a job folder of choice, say BANDFIRSTJOB in your home directory on Stallo.

then, copy the job-script as seen here:

```
#!/bin/bash -l

#################### ADF Job Batch Script Example ####################
# Section for defining queue-system variables:
#------------------------------------
# This script asks for a given set of cores nodes and cores. Stallo has got 16 or 20␣
↪cores/node,
# asking for something that adds up to both is our general recommodation (80, 160␣
↪etc), you would
# then need to use --ntasks instead of --nodes and --ntasks-per-node. (replace both).
# Runtime for this job is 59 minutes; syntax is hh:mm:ss.
# Memory is set to the maximum advised for a full node,
# of 30000MB/node and leaving some for the system to use. Memory
# can be specified pr core, virtual or total pr job (be carefull).
#------------------------------------
# SLURM-section
#SBATCH --job-name=band_runex
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=20
#SBATCH --time=00:59:00
#SBATCH --mem-per-cpu=1500MB # Be ware of memory needs!
#SBATCH --output=band_runex.log
#SBATCH --mail-type=ALL
#SBATCH --exclusive
#SBATCH --partition=multinode

######################################
# Section for defining job variables and settings:
```

(continues on next page)

(continued from previous page)

```
input=silicone # Name of input without extention
ext=adf # We use the same naming scheme as the software default, also for extention
cores=40  # Number of cores potentially used by mpi engine in submit procedure

# We load all the default program system settings with module load:

module --quiet purge
module load ADF/adf2017.108

# Now we create working directory and temporary scratch for the job(s):
# Necessary variables are defined in the notur and the software modules.

export SCM_TMPDIR=/global/work/$USER/$SLURM_JOBID
mkdir -p $SCM_TMPDIR

# Preparing and moving inputfiles to tmp:

submitdir=$SLURM_SUBMIT_DIR
tempdir=$SCM_TMPDIR

cd $submitdir
cp ${input}.${ext} $tempdir
cd $tempdir

# In case necessary, set SCM_IOBUFFERSIZE
#export SCM_IOBUFFERSIZE=1024 # Or higher if necessary.

#######################################
# Section for running the program and cleaning up:

# Running the program:

time band  -n $cores < ${input}.${ext} > band_$input.out

# Cleaning up and moving files back to home/submitdir:
# Make sure to move all essential files specific for the given job/software.

cp band_$input.out $submitdir/band_$input.out
cp TAPE21 $submitdir/$input.t21

# To zip some of the output might be a good idea!
#gzip $input.t21
#mv $input.t21.gz $submitdir/

# Investigate potentially other files to keep:
echo $(pwd)
echo $(ls -ltr)

# ALWAYS clean up after yourself. Please do uncomment the following line
#cd $submitdir
#rm  $tempdir/*
#rmdir $tempdir

echo "Job finished at"
date
################## Job Ended ##################
exit 0
```

(Can also be downloaded from here: `job_band.sh`)

Place this script in the same folder and type:

```
sbatch job_band.sh
```

You may now have submitted your first adf job.

These files are also available on Stallo

```
module load ADF/adf2017.108
cd <to whatever you call your test folder> # for instance BANDFIRSTJOB
cp -R /global/hds/software/notur/apprunex/ADF/* .
```

To verify that the jobs has worked fine, check the outputfile. If it says EXIT and print Energy of Formation at the end of the file, you are likely on the safe side.

#Check the ENERGY OF FORMATION in band_silicone.out. The value should ideally be close to -4.2467 eV. When this is the case, you may alter variables in the shell script as much as you like to adapt to your own jobs.

NB: The ADF modeling suite is installed as precompiled binaries, they come with their own mpi (intel MPI). So if you are not using the provided runscript example and/or loading the newer modules - please make sure that you load an intelmpi module and also preferably a rather recent intel compiler.

Good luck with your chemistry!

Information for advanced users

Scaling behaviour

Since ADF is a very complex code, able to solve a vast range of chemistry problems - giving a unified advice regarding scaling is difficult. We will try to inspect scaling behaviour related to most used areas of application. For a standard geometry optimization, it seems to scale well in the region of 4-6 full nodes (60-100 cores) at least. For linear transit we would currently stay at no more than 4 full nodes or less currently.Unless having tests indicating otherwise, users who want to run large jobs should allocate no more than the prescribed numbers of processors. More information will come.

Memory allocation

On Stallo there are 32 GB and 128GB nodes. Pay close attention to memory usage of job on the nodes where you run, and if necessary redistribute the job so that it uses less than all cores on the node until the limit of 32 GB/core. More than that, you will need to ask for access to the highmem-queue. As long as you do not ask for more than 2 GB/core, using the pmem-flag for torque does in principle give no meaning.

How to restart an ADF job

1. In the directory where you started your job, rename or copy the job-output t21 file into $SCM_TMPDIR/TAPE21.
2. In job.inp file, put RESTART TAPE21 just under the comment line.
3. Submit job.inp file as usual.

This might also be automized, we are working on a solution for restart after unexpected downtime.

How to run ADF using fragments

This is a brief introduction to how to create fragments necessary for among other things, BSSE calculations and proper broken symmetry calculations.

Running with fragments:

- Download and modify script for fragment create run, e.g. this template: Create.TZP.sh (modify ACCOUNT, add desired atoms and change to desired basis and desired functional)
- Run the create job in the same folder as the one where you want to run your main job(s) (sbatch Create.TZP.sh).
- Put the line cp $init/t21.* . in your ADF run script (in your $HOME/bin directory)
- In job.inp, specify correct file name in FRAGMENT section, e.g. "H t21.H_tzp". * Submit job.inp as usual.

Running with fragments is only necessary when you want to run a BSSE calculations or manipulate charge and/or atomic mass for a given atom (for instance modeling heavy isotope labelled samples for frequency calculations).

General Information

29.1.2 Description

The DFT code ADF is used in many areas of chemistry and materials science, and consists of the following parts:

- The pure DFT code based on Slater type orbitals; ADF.
- The periodic DFT code BAND shares a lot of functionality with ADF.d for treating large periodic molecular systems.
- DFTB and MOPAC are fast approximate methods to study large molecules and big periodic systems, employing DFT-based and semi-empirical data, respectively.
- Reaction dynamics in large complex systems can be studied with bond order based ReaxFF.
- COSMO-RS uses quantum mechanical data from ADF to predict thermodynamic properties of solutions and mixtures (LogP, VLE, pKa, . . .)

29.1.3 Online info from vendor

- Homepage: https://www.scm.com
- Documentation: https://www.scm.com/doc

The support people in NOTUR, do not provide trouble shooting guides anymore, due to a national agreement that it is better for the community as a whole to add to the community info/knowledge pool where such is made available. For ADF/BAND we advise to search in general documentation, sending emails to support (either metacenter or scm) or trying the ADF mailing list (see https://www.scm.com/Support for more info).

29.1.4 License and access policy

The license of ADF/Band is commercial.

NOTUR holds a national license of the ADF program system, making usage of ADF/BAND available for all academic scientists in Norway.

We have a national license for the following packages:

- ADF & ADFGUI

- BAND &BANDGUI
- CRS
- DFTB & DFTBGUI
- GUI
- REAXFF & REAXFFGUI
- NBO6 (on Stallo only, but machine license available for all users of Stallo).

Please note that this is an academic type license; meaning that research institutes not being part of Norwegian Universities must provide their own license to be able to use and publish results obtained with this code on NOTUR installlations.

29.1.5 Citation

When publishing results obtained with the referred software referred, please do check the developers web page in order to find the correct citation(s).

29.2 Dalton

29.2.1 Related information

First time you run a Dalton job?

This page contains info aimed at first time users of Dalton on Stallo. Please look carefully through the provided examples. Also note that the job-script example is rather richly commented to provide additional and relevant info.

If you want to run this testjob, copy the input example and the job script example shown below into your test job folder (which I assume you have created in advance).

Dalton needs one molecule file and one input file. For simplicity, we have tarred them together and you may download them here: `Dalton files`

Molcas runscrip example

```
#!/bin/bash -l

# Write the account to be charged here
# (find your account number with `cost`)
#SBATCH --account=nnXXXXk

#SBATCH --job-name=daltonexample

# we ask for 20 cores
#SBATCH --ntasks=20

# run for five minutes
#              d-hh:mm:ss
#SBATCH --time=0-00:05:00

# 500MB memory per core
```

(continues on next page)

(continued from previous page)

```
# this is a hard limit
#SBATCH --mem-per-cpu=500MB

# Remove percentage signs to turn on all mail notification
#%%%SBATCH --mail-type=ALL

# You may not place bash commands before the last SBATCH directive!

# Load Dalton
# You can find all installed Dalton installations with:
#   module avail dalton
module load Dalton/2013.2

# Define the input files
molecule_file=caffeine.mol
input_file=b3lyp_energy.dal

# Define and create a unique scratch directory
SCRATCH_DIRECTORY=/global/work/${USER}/daltonexample/${SLURM_JOBID}
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

# Define and create a temp directory
TEMP_DIR=${SCRATCH_DIRECTORY}/temp
mkdir -p ${TEMP_DIR}

# copy input files to scratch directory
cp ${SLURM_SUBMIT_DIR}/${input_file} ${SLURM_SUBMIT_DIR}/${molecule_file} ${SCRATCH_
↪DIRECTORY}

# run the code
dalton -N ${SLURM_NTASKS} -t ${TEMP_DIR} ${input_file} ${molecule_file}

# copy output and tar file to submit dir
cp *.out *.tar.gz ${SLURM_SUBMIT_DIR}

# we step out of the scratch directory and remove it
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}

# happy end
exit 0
```

You can also download the runscript file here: `Dalton run script`

The runscript example and the input are also on Stallo

Type:

```
cd <whatever_test_folder_you_have> # For instance testdalton
cp -R /global/hds/software/notur/apprunex/dalton/* .
```

When you have all the necessary files in the correct folders, submit the job by typing:

```
sbatch job_dalton.sh
```

Good luck.

29.2.2 General information

Description

Dalton is an ab initio quantum chemistry computer program. It is capable of calculating various molecular properties using the Hartree-Fock, MP2, MCSCF and coupled cluster theories. Dalton also supports density functional theory calculations.

Online information from vendor

- Homepage: http://daltonprogram.org/
- Documentation: http://daltonprogram.org/documentation/
- For download: http://daltonprogram.org/download/

License and access policy

Dalton is licensed under the GPL 2. That basically means that everyone may freely use the program.

Citation

When publishing results obtained with the referred software referred, check the following page to find the correct citation(s): http://daltonprogram.org/citation/

29.2.3 Usage

You load the application by typing:

```
module load Dalton/2016-130ffaa0-intel-2017a
```

For more information on available versions of Dalton, type:

```
module avail Dalton
```

29.3 Gaussian and GaussView

29.3.1 Gaussian job example

To run this example create a directory, step into it, create the input file and submit the script with:

```
$ sbatch gaussian-runscript.sh
```

Input example

Direct download link: `input example`

```
%chk=example
%mem=500MB
#p b3lyp/cc-pVDZ opt

benzene molecule example

0 1
C         0.00000        1.40272        0.00000
C         0.00000       -1.40272        0.00000
C         1.21479        0.70136        0.00000
C         1.21479       -0.70136        0.00000
C        -1.21479        0.70136        0.00000
C        -1.21479       -0.70136        0.00000
H         0.00000        2.49029        0.00000
H         0.00000       -2.49029        0.00000
H         2.15666        1.24515        0.00000
H         2.15666       -1.24515        0.00000
H        -2.15666        1.24515        0.00000
H        -2.15666       -1.24515        0.00000
```

Runscript example

Direct download link: `runscript example`

```
#!/bin/bash -l

#SBATCH --account=nn....k

#SBATCH --job-name=example
#SBATCH --output=example.log

# Stallo nodes have either 16 or 20 cores: 80 is a multiple of both
#SBATCH --ntasks=80

# make sure no other job runs on the same node
#SBATCH --exclusive

# syntax is    DD-HH:MM:SS
#SBATCH --time=00-00:30:00

# allocating 30 GB on a node and leaving a bit over 2 GB for the system
#SBATCH --mem-per-cpu=1500MB

##############################################################################
# no bash commands above this line
# all sbatch directives need to be placed before the first bash command

# name of the input file without the .com extention
input=example
```

(continues on next page)

(continued from previous page)

```
# flush the environment for unwanted settings
module --quiet purge
# load default program system settings
module load Gaussian/g16_B.01

# set the heap size for the job to 20 GB
export GAUSS_LFLAGS2="--LindaOptions -s 20000000"

# split large temporary files into smaller parts
lfs setstripe -c 8 .

# create scratch space for the job
export GAUSS_SCRDIR=/global/work/${USER}/${SLURM_JOB_ID}
tempdir=${GAUSS_SCRDIR}
mkdir -p ${tempdir}

# copy input and checkpoint file to scratch directory
cp ${SLURM_SUBMIT_DIR}/${input}.com ${tempdir}
if [ -f "${SLURM_SUBMIT_DIR}/${input}.chk" ]; then
    cp ${SLURM_SUBMIT_DIR}/${input}.chk ${tempdir}
fi

# run the code
cd ${tempdir}
time g16.ib ${input}.com > ${input}.out

# copy output and checkpoint file back
cp ${input}.out ${SLURM_SUBMIT_DIR}
cp ${input}.chk ${SLURM_SUBMIT_DIR}

# remove the scratch directory after the job is done
# cd ${SLURM_SUBMIT_DIR}
# rm -rf /global/work/${USER}/${SLURM_JOB_ID}

exit 0
```

29.3.2 Memory and number of cores

This page contains info about special features related to the Gaussian install made on Stallo, but also general issues related to Gaussian only vaguely documented elsewhere.

Choosing the right version

To see which versions of Gaussien are available, use:

```
$ module avail Gaussian/
```

To load a specific version of Gaussian, use for instance:

```
$ module load Gaussian/g16_B.01
```

Gaussian over Infiniband

First note that the Gaussian installation on Stallo is the Linda parallel version, so it scales somewhat initially. On top of this, Gaussian is installed with a little trick, where the loading of the executable is intercepted before launched, and an alternative socket library is loaded. This enables the running Gaussian natively on the Infiniband network giving us two advantages:

- The parallel fraction of the code scales to more cores.
- The shared memory performance is significantly enhanced (small scale performance).

But since we do this trick, we are greatly depending on altering the specific node address into the input file: To run gaussian in parallel requires the additional keywords `%LindaWorkers` and `%NProcshared` in the `Link 0` part of the input file. This is taken care of by a wrapper script around the original binary in each individual version folder. This is also commented in the job script example. Please use our example when when submitting jobs.

We have also taken care of the rsh/ssh setup in our installation procedure, to avoid .tsnet.config dependency for users.

Parallel scaling

Gaussian is a rather large program system with a range of different binaries, and users need to verify whether the functionality they use is parallelized and how it scales.

Due to the preload Infiniband trick, we have a somewhat more generous policy when it comes to allocating cores/nodes to Gaussian jobs but before computing a table of different functionals and molecules, we strongly advice users to first study the scaling of the code for a representative system.

Please do not reuse scripts inherited from others without studying the performance and scaling of your job. If you need assistance with this, please contact the user support.

We do advice people to use up to 256 cores (`--tasks`). We have observed acceptable scaling of the current Gaussian install beyond 16 nodes for the jobs that do scale outside of one node (i.e. the binaries in the $gXXroot/linda-exe folder). Linda networking overhead seems to hit hard around this amount of cores, causing us to be somewhat reluctant to advice going beyond 256 cores.

Since we have two different architectures with two different core counts on Stallo, the `--exclusive` flag is important to ensure that the distribution of jobs across the whole system are done in a rather flexible and painless way.

Memory allocation

Gaussian takes care of memory allocation internally.

That means that if the submitted job needs more memory per core than what is in average available on the node, it will automatically scale down the number of cores to mirror the need. This also means that you always should ask for full nodes when submitting Gaussian jobs on Stallo! This is taken care of by the `--exclusive` flag and commented in the job script example.

The `%mem` allocation of memory in the Gaussian input file means two things:

- In general it means memory/node – for share between `nprocshared`, and additional to the memory allocated per process. This is also documented by Gaussian.
- For the main process/node it also represents the network buffer allocated by Linda since the main Gaussian process takes a part and Linda communication process takes a part equally sized – thus you should never ask for more than half of the physical memory on the nodes, unless they have swap space available - which you never should assume. The general `%mem` limit will always be half of the physical memory pool given in MB instead of GB - 16000MB instead of 16GB since this leaves a small part for the system. That is why we would actually advise to use 15GB as maximum `%mem` size.

Large temporary outputs on Stallo

As commented here *Storage and backup* there is an issue related to very large temporary files on Stallo. Please read up on it at act accordingly. This issue is also commented in the job script example.

29.3.3 GaussView

Gaussview is a visualization program that can be used to open Gaussian output files and checkpoint files (.chk) to display structures, molecular orbitals, normal modes, etc. You can also set up jobs and submit them directly.

Official documentation: https://gaussian.com/gaussview6/

GaussView on Stallo

To load and run GaussView on Stallo, load the relevant Gaussian module, and then call GaussView:

```
$ module avail Gaussian
$ module load Gaussian/g16_B.01
$ gview
```

Gaussian is a popular computational chemistry program. Official documentation: http://gaussian.com/man

29.3.4 License and access

The license is commercial/proprietary and constitutes of 4 site licenses for the 4 current host institutions of NOTUR installations; NTNU, UiB, UiO, UiT. Only persons from one of these institutions have access to the Gaussian Software system installed on Stallo. Note that users that do not come from one of the above mentioned institutions still may be granted access, but they need to document access to a valid license for the version in question first.

- To get access to the code, you need to be in the `gaussian` group of users.
- To be in the `gaussian` group of users, you need be qualified for it - see above.

29.3.5 Citation

For the recommended citation, please consult http://gaussian.com/citation/.

29.4 Molcas

29.4.1 Related information

First time you run a Molcas job?

This page contains info aimed at first time users of Molcas on Stallo. Please look carefully through the provided examples. Also note that the job-script example is rather richly commented to provide additional and relevant info.

If you want to run this testjob, copy the input example and the job script example shown below into your test job folder (which I assume you have created in advance).

Molcas comes with one input file and one geometry file. For simplicity, we have tarred them together and you may download them here: `Ethane-input`

Molcas runscrip example

```
#!/bin/bash -l

################### VASP Job Batch Script Example ###################
# Section for defining queue-system variables:
#-------------------------------------
# This script asks for a given set of cores nodes and cores. Stallo has got 16 or 20␣
↪cores/node,
# asking for something that adds up to both is our general recommodation (80, 160␣
↪etc), you would
# then need to use --ntasks instead of --nodes and --ntasks-per-node. (replace both).
# Runtime for this job is 59 minutes; syntax is hh:mm:ss.
# Memory is set to the maximum advised for a full node, 1500MB/core - giving a total
# of 30000MB/node and leaving some for the system to use. Memory
# can be specified pr core, virtual or total pr job (be carefull).
#-------------------------------------
# SLURM-section
#SBATCH --job-name=molcas_runex
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
##SBATCH --ntasks=20
#SBATCH --time=00:59:00
#SBATCH --mem-per-cpu=1500MB
#SBATCH --output=molcas_runex.log
#SBATCH --mail-type=ALL
#SBATCH --exclusive

#######################################
# Section for defining job variables and settings:


proj=ehtane # Name of project/folder
input=C2H6 # Name of job input

# We load all the default program system settings with module load:

module --quiet purge
module load Molcas/molcas82-intel-2015a
# You may check other available versions with "module avail Molcas"

# Now we create working directory and temporary scratch for the job(s):
# Necessary variables are defined in the notur and the software modules.

export MOLCAS_WORKDIR=/global/work/$USER/$SLURM_JOBID
mkdir -p $MOLCAS_WORKDIR

# Preparing and moving inputfiles to tmp:

submitdir=$SLURM_SUBMIT_DIR
tempdir=$MOLCAS_WORKDIR

cd $submitdir
cp ${input}.* $tempdir
cd $tempdir

#######################################
```

(continues on next page)

(continued from previous page)

```
# Section for running the program and cleaning up:

# Running the program:
time molcas  Project=${proj} -f ${input}*.input

# Cleaning up and moving files back to home/submitdir:
# Make sure to move all essential files specific for the given job/software.

cp * $submitdir

# To zip some of the output might be a good idea!
#gzip $resultszip
#mv $resultzip.gz $SUBMITDIR/

# Investigate potentially other files to keep:
echo $(pwd)
echo $(ls -ltr)

# ALWAYS clean up after yourself. Please do uncomment the following line
#cd $submitdir
#rm  $tempdir/*
#rmdir $tempdir

echo "Job finished at"
date
################### Job Ended ###################
exit 0
```

NB: Note that some of Molcas´s various modules are not mpi scalable. Consult the vendor-provided manual for scaling issues. Our example is based on a single-node all cores job.

You can also download the runscript file here: `Molcas run script`

The runscript example and the input are also on Stallo

Type:

```
module load Molcas/molcas82-intel-2015a
cd <whereevertestfolderyouhave> # For instance testmolcas
cp -R /global/hds/software/notur/apprunex/Molcas/* .
```

When you have all the necessary files in the correct folders, submit the job by typing:

```
sbatch job_molcas.sh
```

To verify that nothing has gone wrong, check the status file. If it concludes “happy landing” you are on the safe side, most probably. Good luck.

29.4.2 General information

Description

Molcas is an ab initio quantum chemistry software package developed by scientists to be used by scientists. The basic philosophy is is to be able to treat general electronic structures for molecules consisting of atoms from most of

the periodic table. As such, the primary focus of the package is on multiconfigurational methods with applications typically connected to the treatment of highly degenerate states.

Key aspects of Molcas:

- SCF/DFT, CASSCF/RASSCF, CASPT2/RASPT2
- Fast, accurate, and robust code

Online information from vendor

- Homepage: http://molcas.org/
- Documentation: http://molcas.org/documentation/manual/
- For download: http://molcas.org/download.html

License and access policy

Molcas comes with a non-free computer center license, though it is free for academic employees from the Nordic region. The HPC group @ UiT have an agreement with Molcas as a part of the old application management project, that we do make a central install of the code - but users who wants access needs to document that they have made an agreement with Molcas. This proof of agreement should then be mailed (e or non-e) to support@metacenter.no to be granted membership of the molcas group on Stallo.

For the future, we are considering establishing the same policy as for *ORCA*, especially since the vendor has done a substantial job at decomplicating the install procedure using CMake.

Citation

When publishing results obtained with the referred software referred, please do check the developers web page in order to find the correct citation(s).

29.4.3 Usage

You load the application by typing:

```
module load Molcas/molcas82-intel-2015a
```

For more information on available versions of Molcas, type:

```
module avail Molcas
```

29.5 ORCA

Some users have requested an installation of the ORCA quantum chemistry program on Stallo. Unfortunately, ORCA does not come with a computer center license, which basically means that we cannot easily install it globally on Stallo. However, the code is free of charge and available for single users or at research group level, provided compliance with the *ORCA End User License Agreement*. This means that interested users can download the precompiled executables from the ORCA web page themselves, and run the code on Stallo.

- Homepage: https://orcaforum.kofo.mpg.de

29.5.1 Installation

The following has been tested for version 4.2.1 on Stallo:

1) Go to the ORCA forum page and register as user
2) Wait for activation e-mail (beware of spam filters…)
3) Log in and click on **Downloads** in the top menu
4) Download the Linux x86-64 complete archive and place it somewhere in your home folder
5) Extract the tarball: `$ tar xf orca_4_2_1_linux_x86-64_openmpi314.tar.xz`
6) The `orca` executable is located in the extracted archive and should work out of the box

29.5.2 Usage

In order to run ORCA in parallel the OpenMPI module must be loaded, and it is important to use the **full path** to the executable

```
$ module load OpenMPI/3.1.3-GCC-8.2.0-2.31.1
$ /full/path/to/orca orca.inp >orca.out
```

Note that the calculation should not be launched by `mpirun`, but you should use the input keyword `nprocs`. A very simple example input file is provided below. Please refer to the ORCA manual (can be downloaded from the same download page on the ORCA forum) for details on how to run the code for your application.

ORCA input example

```
! HF def2-TZVP

%pal
nprocs 4    # parallel execution
end

* xyz 0 1
C  0.0  0.0  0.0
O  0.0  0.0  1.13
*
```

29.6 Schrödinger Product Suites

A description of the Schrödinger product suites that are available for norwegian academic users

29.6.1 Related information

Running Schrodinger jobs on Stallo

This page contains info about how to submit Schrodinger jobs on Stallo, based on two sets of examples - one for docking and one for molecular dynamics - provided to us by users.

Pay enhanced attention to the *ssh -Y c1-2* in examples below; this represents the adviced behaviour on how to run jobs on Stallo for your benefit solely!

A more thorough explanation to this, is that maestro starts a distribution and surveilance process that creates the jobs that enters the shared resources allocation (aka batch) system. If this process dies, the underlying jobs dies disregarding their computational status. This could have been solved by just running this on the login node, but imagine how it would have been with 1000 simultanious users sharing two login nodes and 50 of those ran 20-40 simultaneous perl and python processes each on the login nodes. So, please do as told.

Submitting jobs through the maestro interface

Log in to Stallo with either x11-tunnelling enabled or through stallo-gui.uit.no.

To get download the jobscript example(s), type the following:Direct log in via ssh:

```
$ module load Schrodinger/2017-2-intel-2016a
$ cp -r /global/hds/software/notur/apprunex/Schrodinger/* ~
# Be sure this do not overwrite any folder or info you may want to keep in your home.
```

Note: This suite is quite extensive in features, and we generally advice you to either join tutorial courses, talk to experts in your proximity or read the vendor-provided documentation if you have absolutely no knowledge about how to use this suite of software. Here we only provide a couple of rough startup examples to get you up running.

Then;

If you want to test the molecular dynamics package

Do the following:

```
$ ssh -Y c1-2
$ module load Schrodinger/2017-2-intel-2016a
$ cd example_md
$ maestro -SGL
```

and start a Molecular Dynamics task from the Tasks menu bar. Load model system from file, choose desmond_md_example.cms

Set all settings to your satisfaction.

Open the job settings; you should only be presented the following options

1. localhost (only for job-setups)
2. batch-normal (2 days (=48hrs) of walltime/1000 cpus)
3. batch-long (21 days (504 hrs) of walltime/1000cpus)

Make sure that only one option is highlighted (only one of the batch-XXX´s). Press the run button and supervise running.

(This test case is provided us by Vladimir Pomogaev.)

If you want to test the docking package

Do the following:

```
$ ssh -Y c1-2
$ module load Schrodinger/2017-2-intel-2016a
$ cd example_docking
$ maestro -SGL
```

and start a Docking task from the Tasks menu bar.

Choose abl1-kinase-t316i_glide-grid.zip as grid

Use ligands from file; if you want to run a very short test - choose abl1-kinase-t316i_minimized.mae, for a longer test browse for SD and choose Asinex-50000-3D-minimized.sdf set of ligands. Set all settings to your satisfaction. For job settings, see example above.

(This test case is provided us by Bjorn Dalhus.)

If you want to run vsw routines from the command line

The Schrodinger suite is shipped with scripts that connects the software installation with the system batch resource allocation setup, making it possible to submit glide jobs from the linux command line.

Examples of valid command line submissions using the vsw-tool on Stallo:

```
vsw *.inp -DRIVERHOST batch-normal -host_glide batch-normal:100 -NJOBS 1 -adjust
```

If you are worried that you will not refind your files for emergency startup:

Note the following details:

1. When the Schrodinger module is loaded on Stallo, the Schrodinger software folder is set in path, making $SCHRODINGER unecessary.
2. The Schrodinger setup on Stallo writes to the scratch file system by default, potentially making both the -LOCAL and the -SAVE flags uneccesary.
3. We do not recommend the -REMOTEDRIVER flag due to the risk of loosing jobs related to the admin process running out allocated time.

Installing Schrodinger on your local client machine

This page contains info about how to download and install the Schrodinger suite on your local client machine.

Getting an account on www.schrodinger.com

You need to create an account (free) and log in with the password you create. Then you download the relevant software for your machine and follow install procedures that comes with the software.

Getting access to the national license

Since we have a joint funding of license tokens between user community and national bodies, members of the funding research groups are entiteled to get access to the license from their client machines. The easiest way to do this is by downloading the following license file:

```
SERVER Lisens-Schrodinger.uit.no 0050569f5819 27008
VENDOR SCHROD PORT=53000
USE_SERVER
```

(Can also be downloaded from here: `License_Schrodinger.txt`)

This should be sufficient to make your personal copy of Schrodinger Small Molecule Drug Discovery suite work on your machine.

Note: It is a known bug that the configure setup of Schrodinger reports no license found even if the program suite works well. Please ignore.

General Information

NOTUR is, together with the user community, funding a national license of Schrödinger´s Small Molecule Drug Discovery Suite and PyMol. This implies that users are allowed to download and install their own copies of the software, and using the license server for providing license tokens. More about that in another document: *Installing Schrodinger on your local client machine*.

29.6.2 Online info from vendor

- Homepage: https://www.schrodinger.com/

About the suites:

- Small Molecule Drug Discovery Suite: https://www.schrodinger.com/suites/small-molecule-drug-discovery-suite
- PyMol: https://www.schrodinger.com/pymol

For documentation, you need to create an account (free) and log in. Documentation is also available in the software home folder:

```
$ module load Schrodinger/{version}
$ cd $SCRODINGER/docs
```

Remeber to point a pdf-reader or html reader to the documentation if you plan to read it on Stallo.

Support people in NOTUR do not provide trouble shooting guides anymore. For Schrödinger suites, the vendor company is giving good support on a system level. Problems related to running schrodinger on a NOTUR facility should be adressed to us.

29.6.3 Citation

When publishing results obtained with the referred software, please do check the developers web page in order to find the correct citation(s).

29.6.4 License and access policy

The licenses of Schrödinger Product Suites are commercial.

NOTUR is, together with the user community, funding a national license of Schrödinger´s Small Molecule Drug Discovery Suite and PyMol. The licenses are administered by a license server based on flexlm. The adress for this license setup is available upon request to support@metacenter.no.

The outtake of license tokens is monitored on regular basis, and we try to catch those who seems to use the suite for regular scientific production and ask them to contribute financially to the overall deal. So far, this policy have worked fairly well.

All members of the co-funding research groups have unlimited access to the Schrödinger´s Small Molecule Drug Discovery Suite and PyMol license tokens.

Please note that this is an academic type license; meaning that research institutes not being part of Norwegian Universities must provide their own license to be able to use and publish results obtained with this code on NOTUR installlations.

Usage

The most commot usage of schrodinger on Stallo is through the Maestro gui. Log in with x11-tunneling enabled, or through the web-interface http://stallo-gui.uit.no/. Load the module of choice:

Use

```
$ module avail Schrodinger
```

to see which versions of Schrodinger are available.

Use

```
$ module load Schrodinger/<version> # i.e 2017-2-intel-2016a
```

to get access to Schrodinger. The batch resource allocation system is integrated with the gui through a schrodinger.hosts file which is centrally administered.

Please do not hold a local copy!

For examples on how to submit Schrodinger jobs on Stallo, look here *Running Schrodinger jobs on Stallo*.

Finding available licenses

This should in principle be obsolete for users, since we are promised unlimited licenses in the national system. But still, for the curious souls:

If you want to know about avaible licenses; do the following

(after loading the schrodinger module)

```
$ licadmin STAT
```

This command will give you information about license status for the national Schrodinger suite licenses.

Access to PyMOL

For users that wants/needs access to PyMOL, please fill out the following form: https://skjema.uio.no/pymol-access.

Please not that this strategy replaces old habits of sending personal emails in this regard.

29.7 The TURBOMOLE program system

Information regarding the quantum chemistry program system TURBOMOLE.

29.7.1 Related information

TURBOMOL runscript example

```
#!/bin/bash -l
#SBATCH --job-name=turbomole
#SBATCH --account=nnXXXXk
#SBATCH --time=1:00:00
```

(continues on next page)

(continued from previous page)

```
#SBATCH --mem-per-cpu=1000MB
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=20
#SBATCH --mail-type=ALL
#SBATCH --partition=multinode

# For TURBOMOLE v7.2:
# --nodes           : scale by the size of the calculation
# --ntasks-per-node : always a singel task per node
# --cpus-per-task   : always full nodes

# load modules
module --quiet purge
module load TURBOMOLE/7.2

# TURBOMOLE needs this variable
export TURBOTMPDIR=/global/work/${USER}/${SLURM_JOBID}
workdir=${TURBOTMPDIR}
submitdir=${SLURM_SUBMIT_DIR}

# copy files to work
mkdir -p $workdir
cd $workdir
cp $submitdir/* $workdir

# set parallel environment
export PARNODES=${SLURM_NTASKS}
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

# run jobex calculation
dscf > dscf.out

# copy files back
mv * $submitdir

# clean up
rm -r $workdir

exit 0
```

TURBOMOL example on Stallo

The above runscript can be found on Stallo along with the necessary input files to perform a simple DFT calculation. To get access to the examples directory, load the `notur` module

```
$ module load notur
$ cp -r /global/hds/software/notur/apprunex/TURBOMOLE .
```

From this directory you can submit the example script (you need to specify a valid account in the script first)

```
$ sbatch job_turbo.sh
```

The calculation should take less than 2 min, but may spend much more than this in the queue. Verify that the output file `dscf.out` ends with

```
**** dscf : all done ****
```

Please refer to the TURBOMOLE manual for how to setup different types of calculations.

29.7.2 General Information

Description

TURBOMOLE is a quantum chemical program package, initially developed in the group of Prof. Dr. Reinhart Ahlrichs at the University of Karlsruhe and at the Forschungszentrum Karlsruhe. In 2007, the TURBOMOLE GmbH (Ltd) was founded by the main developers of the program, and the company took over the responsibility for the coordination of the scientific development of the program, to which it holds all copy and intellectual property rights.

Online info from vendor

- Homepage: http://www.turbomole-gmbh.com
- Documentation: http://www.turbomole-gmbh.com/turbomole-manuals.html

License information

Sigma2 holds a national TURBOMOLE license covering all computing centers in Norway.

Citation

When publishing results obtained with this software, please check with the developers web page in order to find the correct citation(s).

29.7.3 TURBOMOLE on Stallo

Usage

To see which versions of TURBOMOLE are available

```
$ module avail turbomole
```

Note that the latest (as of May 2018) version 7.2 comes in three different parallelization variants

- TURBOMOLE/7.2
- TURBOMOLE/7.2.mpi
- TURBOMOLE/7.2.smp

where the latter two correspond to the old separation between distributed memory (mpi) and shared memory (smp) implementations that some users may know from previous versions of the program. We recommend, however, to use the new hybrid parallelization scheme (new in v7.2) provided by the `TURBOMOLE/7.2` module.

In order to run efficiently in hybrid parallel the SLURM setup must be correct by specifying CPUs rather than tasks per node

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=20
```

i.e. do NOT specify `--ntasks=40`. Also, some environment variables must be set in the run script

```
export TURBOTMPDIR=/global/work/${USER}/${SLURM_JOBID}
export PARNODES=${SLURM_NTASKS}
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
```

See the TURBOMOLE example for more details.

NOTE: The new hybrid program (v7.2) will launch a separate server process in addition to the requested parallel processes, so if you run e.g. on 20 cores each on two nodes you will find one process consuming up to 2000% CPU on each node plus a single process on the first node consuming very little CPU.

NOTE: We have found that the hybrid program (v7.2) runs efficiently only if *full* nodes are used, e.i. you should always ask for `--cpus-per-node=20`, and scale the number of nodes by the size of the calculation.

29.8 VASP (Vienna Ab initio Simulation Package)

Information regarding the Vienna Ab initio Simulation Package (VASP) installed on Stallo.

29.8.1 Related information

First time you run a VASP job?

This page contains info aimed at first time users of VASP on Stallo, but may also be usefull to more experienced users. Please look carefully through the provided examples. Also note that the job-script example is rather richly commented to provide additional and relevant info.

If you want to run this testjob, download the copies of the scripts and put them into your test job folder (which I assume you have created in advance).

VASP input example

Download the tarred job `CeO2job-files`.

move this file to your test job folder on Stallo and type

```
tar -zxf CeO2.tar.gz
```

Then; download the job-script as seen here:

```
#!/bin/bash -l

################## VASP Job Batch Script Example ##################
# Section for defining queue-system variables:
#-------------------------------------
# This script asks for a given set of nodes and cores. cores. Stallo has got
# 16 or 20 cores/node, so you need to know what you want.
# Runtime for this job is 59 minutes; syntax is hh:mm:ss.
# Memory si set to the maximum amound adviced when asking for a full node, it
```

(continues on next page)

(continued from previous page)

```
# adds up to 30 000MB, leaving a small part for the system to use. Memory
# can be specified pr core, virtual or total pr job (be carefull).
#-------------------------------------
# SLURM-section
#SBATCH --job-name=vasp_runex
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=20
#SBATCH --time=02:00:00
##SBATCH --mem-per-cpu=1500MB
#SBATCH --output=vasp_runex.log
#SBATCH --mail-type=ALL
#SBATCH --exclusive
#SBATCH --partition=multinode

######################################
# Section for defining job variables and settings:

# When you use this test script, make sure, your folder structure is as follows:
# ./job_vasp.sh
# ./CeO2job/INCAR
# ./CeO2job/KPOINTS
# ./CeO2job/POTCAR
# ./CeO2job/POSCAR

proj=CeO2job # Name of job folder
input=$(ls ${proj}/{INCAR,KPOINTS,POTCAR,POSCAR}) # Input files from job folder

# We load all the default program system settings with module load:

module --quiet purge
module load VASP/5.4.1.plain-intel-2016a
# You may check other available versions with "module avail VASP"

# Now we create working directory and temporary scratch for the job(s):
# Necessary variables are defined in the notur and the software modules.

export VASP_WORKDIR=/global/work/$USER/$SLURM_JOB_ID

mkdir -p $VASP_WORKDIR

# Preparing and moving inputfiles to tmp:

submitdir=$SLURM_SUBMIT_DIR

cp $input $VASP_WORKDIR
cd $VASP_WORKDIR

######################################
# Section for running the program and cleaning up:

# Running the program:

time mpirun vasp_std

# Cleaning up and moving files back to home/submitdir:
# Make sure to move all essential files specific for the given job/software.
```

(continues on next page)

(continued from previous page)

```
cp OUTCAR $submitdir/${proj}.OUTCAR

# To zip some of the output might be a good idea!
#gzip results.gz OUTCAR
#mv $results.gz $submitdir/

# Investigate potentially other files to keep:
echo $(pwd)
echo $(ls -ltr)

# ALWAYS clean up after yourself. Please do uncomment the following line
# If we have to, we get really grumpy!
#cd $submitdir
#rm -r $VASP_WORKDIR/*

echo "Job finished at"
date
################### Job Ended ###################
exit 0
```

Download it here `job_vasp.sh`.

These files are also available on Stallo

```
module load VASP/5.4.1.plain-intel-2016a
cd <to whatever you call your test folder> # for instance testvasp
cp -R /global/hds/software/notur/apprunex/VASP/* .
sbatch job_vasp.sh
```

and you are up running. Happy hunting.

About the VASP install on Stallo

This page contains information related to the installation of VASP on Stallo. Some of this is relevant also for self-compilation of the code, for those who want to give this a try.

VASP on Stallo

Note that the VASP installation on Stallo mainly follows the standard syntax introduced by the vASP team with their new installation scheme. Based on their system, we have added two binaries - as commented under.

If you do

```
module avail VASP
```

on Stallo, you will notice that for version 5.4.1 and onwards, there is a dramatic increase in the available binaries. And this might appear confusing.

First; All versions of VASP is compiled with the support for maximally-localised Wannier functions and the Wannier90 program and also the MPI flag in FPP (-DMPI)

Second; Each release of VASP is compiled in two different versions; “tooled” and “plain”.

- VASP/x.y.z.tooled is a version where all necessary support for Texas transition state tools (vTST) and the explicit solvation model (VSPsol) and BEEF is added.
- VASP/x.y.z.plain is the version without this support/ additions. (Relatively unmodified source).

The reason for this is that we are uncertain of effects on the tooles on calculated numbers, due to reproducibility, we have chosen to hold the different versions separate.

Then it starts getting interesting: For each version, there are 15 different binaries - consisting of 3 groups of 5.

- unmodified group; binaries without any additional name to them; vasp_std
- noshear: vasp_std_noshear
- abfix: vasp_std_abfix

The unmodified group is compiled on basis on the unmodified set of fortran files that comes with the code. The noshear version is compiled with a modified version of the file constr_cell_relax.F file where shear forces are not calculated. The abfix version is compiled with a modified version of the constr_cell_relax.F file where two lattice vectors are fixed.

There are 5 different binaries in each group, all compiled with the same FPP settings (mentioned below):

- vasp_std
- vasp_gam
- vasp_ncl

These are familiar from the standard build system that is provided with the code. In addtion to these, we have

- vasp_tbdyn
- vasp_gam_tbdyn

FPP settings for each binary

1. vasp_std is compiled with the following additional FPP flag(s): -DNGZhalf
2. vasp_gam is compiled with the following additional FPP flag(s): -DNGZhalf -DwNGZhalf
3. vasp_ncl is compiled with the following additional FPP flag(s): no modifcations in FPP settings
4. vasp_tbdyn is compiled with the following additional FPP flag(s): -DNGZhalf -Dtbdyn
5. vasp_gam_tbdyn is compiled with the following additional FPP flag(s): -DNGZhalf -DwNGZhalf -Dtbdyn

We would be happy to provide a copy of our build scripts (patches) upon request.

About memory allocation for VASP

VASP is known to be potentially memory demanding. Quite often, you might experience to use less than the full number of cores on the node, but still all of the memory.

For core-count, node-count and amounts of memory on Stallo, see *About Stallo*.

There are two important considerations to make:

First: Make sure that you are using the SBATCH –exclusive flag in your run script. Second: How to allocate all the memory:

```
#!/bin/bash -l

#################################################
# Example for a job that consumes a lot of memory #
#################################################

#SBATCH --job-name=example

# we ask for 1 node
#SBATCH --nodes=1

# run for five minutes
#               d-hh:mm:ss
#SBATCH --time=0-00:05:00

# total memory for this job
# this is a hard limit
# note that if you ask for more than one CPU has, your account gets
# charged for the other (idle) CPUs as well
#SBATCH --mem=31000MB

# turn on all mail notification
#SBATCH --mail-type=ALL

# you may not place bash commands before the last SBATCH directive

# define and create a unique scratch directory
SCRATCH_DIRECTORY=/global/work/${USER}/example/${SLURM_JOBID}
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

# we copy everything we need to the scratch directory
# ${SLURM_SUBMIT_DIR} points to the path where this script was submitted from
cp ${SLURM_SUBMIT_DIR}/my_binary.x ${SCRATCH_DIRECTORY}

# we execute the job and time it
time ./my_binary.x > my_output

# after the job is done we copy our output back to $SLURM_SUBMIT_DIR
cp ${SCRATCH_DIRECTORY}/my_output ${SLURM_SUBMIT_DIR}

# we step out of the scratch directory and remove it
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}

# happy end
exit 0
```

29.8.2 General information

Description

VASP is a complex package for performing ab-initio quantum-mechanical molecular dynamics (MD) simulations using pseudopotentials or the projector-augmented wave method and a plane wave basis set. The approach implemented in VASP is based on the (finite-temperature) local-density approximation with the free energy as variational quantity and an exact evaluation of the instantaneous electronic ground state at each MD time step.

VASP uses efficient matrix diagonalisation schemes and an efficient Pulay/Broyden charge density mixing. These techniques avoid all problems possibly occurring in the original Car-Parrinello method, which is based on the simultaneous integration of electronic and ionic equations of motion.

The interaction between ions and electrons is described by ultra-soft Vanderbilt pseudopotentials (US-PP) or by the projector-augmented wave (PAW) method. US-PP (and the PAW method) allow for a considerable reduction of the number of plane-waves per atom for transition metals and first row elements. Forces and the full stress tensor can be calculated with VASP and used to relax atoms into their instantaneous ground-state.

There are various type plug ins and added functionality to VASP. On Stallo, we have added support for the Texas transition state tools (vTST), the VASPsol implicit solvation model made by Hennig and Mathew from University of Florida and the Bayesian Error Estimation Functionals from the SUNCAT center, Standford.

Online information from vendor

- Homepage: https://www.vasp.at
- Documentation: https://cms.mpi.univie.ac.at/wiki/index.php/The_VASP_Manual

License and access policy

The VASP license is proprietary and commercial. It is based on group license policy, and for NOTUR systems VASP packages falls in the "bring your own license" category. See *General comments about licenses*.

The Vasp program is not distributed via site licences. However, HPC-staff in NOTUR have access to the VASP code to be able to support any research groups that have a valid VASP license.

VASP is a commercial software package that requires a license for all who wants use it. To run VASP:

1. Your group must have a valid licence. To acquire a licence, please consult this link: https://www.vasp.at/index.php/faqs/71-how-can-i-purchase-a-vasp-license.
2. We need to get a confirmation from a VASP representative to confirm that you have access to the license. Your group representative needs to contact Dr. Doris Vogtenhuber (doris.vogtenhuber@univie.ac.at) from the VASP team and ask her to send a confirmation email to us (support@metacenter.no) to confirm that you have a valid licence. Once we get a confirmation email we will grant you access to run VASP.

The support people in NOTUR, do not provide trouble shooting guides anymore, due to a national agreement that it is better for the community as a whole to add to the community info/knowledge pool where such is made available. Also, HPC staff from UiT does not provide any support to VASP 4 anymore, basically due to age of the code.

Citation

When publishing results obtained with the referred software referred, please do check the developers web page in order to find the correct citation(s).

29.8.3 Usage

You load the application by typing:

```
$ module load VASP
```

This command will give you the default version.

For more information on available versions, type:

```
$ module avail VASP
```

Experiencewise, VASP is a rather memory intensive code. Users are advised to read up on the general job script example(s) for SLURM and Stallo, and also how to specify memory in SLURM.

About the VASP version(s) installed on Stallo

Since the installation we made on Stallo is rather taylor made, we have gathered information about this here: *About the VASP install on Stallo*.

Here we also adress compilation relevant information for those who would like to do it themselves.

29.9 The COMSOL Multiphysics software suite

Information regarding the computational physics program system COMSOL

29.9.1 Related information

First time you run an COMSOL job?

This page contains info aimed at first time users of COMSOL on Stallo, but may also be usefull to more experienced users. Please look carefully through the provided examples. Also note that the job-script example is rather richly commented to provide additional and relevant info.

If you want to run this testjob, download the copies of the scripts and put them into your test job folder (which I assume you have created in advance).

COMSOL input example

The file used in this example can also downloaded here: `Small test for COMSOL`.

Place this file in a job folder of choice, say COMSOLFIRSTJOB in your home directory on Stallo.

then, copy the job-script as seen here:

```
#!/bin/bash -l

# Note that this is a setup for the highmem 128GB nodes. To run on the regular
# nodes, change to the normal partition and reduce the memory requirement.
#####################################
#SBATCH --job-name=comsol_runex
#SBATCH --nodes=2
#SBATCH --cpus-per-task=16      # Highmem nodes have 16 cores
#SBATCH --mem-per-cpu=7500MB    # Leave some memory for the system to work
#SBATCH --time=0-01:00:00       # Syntax is DD-HH:MM:SS
#SBATCH --partition=highmem     # For regular nodes, change to "normal"
#SBATCH --mail-type=ALL         # Notify me at start and finish
#SBATCH --exclusive             # Not necessary, implied by the specs above
##SBATCH --account=nnXXXXk      # Change to your account and uncomment
#####################################

# define input
```

(continues on next page)

(continued from previous page)

```
inp=$1  # First input argument: Name of input without extention
std=$2  # Second input argument: Type of study
ext=mph # We use the same naming scheme as the software default extention

# define directories
submitdir=${SLURM_SUBMIT_DIR}
workdir=/global/work/${USER}/${SLURM_JOBID}

# create work directory
mkdir -p ${workdir}

# move input to workdir
cp ${inp}.${ext} ${workdir}

# load necessary modules
module purge --quiet
module load COMSOL/5.3-intel-2016a

# run calculation in workdir
cd ${workdir}
time comsol -nn ${SLURM_NNODES}\
            -np ${SLURM_CPUS_PER_TASK}\
            -mpirsh /usr/bin/ssh\
            -mpiroot $I_MPI_ROOT\
            -mpi intel batch\
            -inputfile ${inp}.${ext}\
            -outputfile ${inp}_out.${ext}\
            -study ${std}\
            -mlroot /global/apps/matlab/R2014a

# move output back
mv *out* $submitdir

# cleanup
cd $submitdir
rm -r ${workdir}

exit 0
```

(Can also be downloaded from here: `run_comsol.sh`)

Before you can submit a job, you need to know what "type" of study you want to perform (please read more about that on the vendor support page). For example purpose, I have chosed study 4 - named std04; and this is the second variable argument to the run script (see comments in script).

Place this script in the same folder and type:

```
sbatch run_comsol.sh comsol_smalltest std04
```

You may now have submitted your first COMSOL job.

Good luck with your (MULTI)physics!

General Information

29.9.2 Description

COMSOL Multiphysics® is a general-purpose software platform, based on advanced numerical methods, for modeling and simulating physics-based problems. It is highly visual, based on graphical interfaces - so users are adviced to use either the client installation on their client machine or the remote graphics solution on Stallo.

29.9.3 Online info from vendor

- Homepage: https://www.comsol.com
- Documentation/support: https://www.comsol.com/support

29.9.4 License information

The license of COMSOL is commercial/proprietary.

The installation of COMSOL on Stallo is for UiT users only. If you need/want access to COMSOL, you need to contact UiT support.

29.9.5 Citation

When publishing results obtained with the referred software referred, please do check the developers web page in order to find the correct citation(s).

Additional online info about COMSOL on Stallo:

29.9.6 Usage

COMSOL is a rather memory intensive code, so we would generally advice users to either run their jobs on the highmem-nodes (with 8GB/core of memory) or less than full nodes on the slim nodes.

Use

```
$ module avail COMSOL
```

to see which versions of COMSOL are available. Use

```
$ module load COMSOL/<version> # i.e 5.3-intel-2016a
```

to get access to any given version of COMSOL.

The license of comsol is installed as a floating network license type, meaning that there is a server checking out license tokens upon request. If you want to know whether there are available license tokens or not, load the module as above. Then type

```
$ lmstat -c $LMCOMSOL_LICENSE_FILE -a
```

Happy calculations!

CHAPTER 30

Storage and backup

30.1 Available file system

Stallo has a "three folded" file system:

- Global accessible home area (user area): /home (64 TB)
- Global accessible work or scratch area: /global/work (1000 TB)
- Local accessible work or scratch area on each node: /local/work (~450 GB)

30.2 Home area

The file system for user home directories on Stallo. It is a 64 TB global file system, which is accessible from both the login nodes and all the compute nodes. The size of the home directory's for each user is 300 GB. It is not possible to extend the size. If you need more space, consider using /global/work (see below).

The home area is for "permanent" storage only, so please do not use it for temporary storage during production runs. Jobs using the home area for scratch files while running may be killed without any warning.

30.3 Work/scratch areas

Warning: Due to stallo coming close to its storage limits, starting from July 2020 /global/work will be subject to a auto cleanup affecting all files older than 21 days.

Starting from first of July we will move all files which haven't been accessed for more than 21 days to a trash folder from where they will be deleted in due time.

We ask you to classify your data in /global/work and move all files you need to keep to your home folder or other storage options like NIRD. In order to save storage space consider archiving your data in compressed form, see *Compression of data*.

In case you miss important files that have been moved, please write an email to migration@metacenter.no as we keep the files for some time and can restore them if needed.

There are two different work/scratch areas available on Stallo:

- 1000 TB global accessible work area on the cluster, accessible from both the login nodes and all the compute nodes as /global/work. This is the recommended work area, both because of size and performance! Users can stripe files themselves as this file system is a Lustre file system.
- In addition, each compute node has a small work area of approximately 450 GB, only locally accessible on each node. This area is accessible as /local/work on each compute node. In general we do not recommend to use /local/work, both because of (the lack of) size and performance, however for some users this may be the best alternative.

These work areas should be used for all jobs running on Stallo.

There is no backup of files stored on the work areas. If you need permanent storage of large amounts of data, please contact the system administrators: support@metacenter.no

Disk quota is not enforced on work/scratch areas. Please use common courtesy and keep your work/scratch partitions clean. Move all files you do not need on Stallo elsewhere or delete them. Since overfilled work/scratch partitions can cause problems, files older than 14 days are subject for deletion without any notice.

Files on /local/work/ belonging to users other than the one that runs a job on the node will be deleted.

30.4 Backup

There is no real backup of the data on Stallo. However we do keep daily snapshots of /home and /project for the last 7 days. The /home snapshots are kept at /global/hds/.snapshot/

There is no backup of files stored on the /global/work and /local/work areas. If you need permanent storage of large amounts of data, or if you need to restore some lost data, please contact the system administrators: support@metacenter.no

30.5 Archiving data

Archiving is not provided. However you may apply for archive space on Norstore.

30.6 Closing of user account

User accounts on Stallo are closed on request from Uninett Sigma or the project leader. The account is closed in a way so that the user no longer can log in to Stallo.

If the user has data needed by other people in the group all data on /home/ is preserved.

30.7 Privacy of user data

General privacy

There is a couple of things you as a user, can do to minimize the risk of your data and account on Stallo being read/accessed from the outside world.

1. Your account on Stallo is personal, do not give away your password to anyone, not even the HPC staff.
2. If you have configured ssh-keys on your local computer, do not use passphrase-less keys for accessing Stallo.

By default a new account on Stallo is readable for everyone on the system. That is both /home/ and /global/work/

This can easily be change by the user using the command chmod The chmod have a lot "cryptic" combinations of options (click here for a more in depth explanation). the most commonly used is:

- only user can read their home directory:

```
chmod 700 /home/$USER
```

- User and their group can read and execute files on the home directory:

```
chmod 750 /home/$USER
```

- User and all others including the group can read and execute the files:

```
chmod 755 /home/$USER
```

- everybody can read, execute, and WRITE to directory:

```
chmod 777 /home/$USER
```

30.8 Management of lage files (> 200GB)

Some special care needs to be taken if you want to create very large files on the system. With large we mean file sizes over 200GB.

The /global/work file system (and /global/home too) is served by a number of storage arrays that each contain smaller pieces of the file system, the size of the chunks are 2TB (2000GB) each. In the default setup each file is contained within one storage array so the default filesize limit is thus 2TB. In practice the file limit is considerably smaller as each array contains a lot of files.

Each user can change the default placement of the files it creates by striping files over several storage arrays. This is done with the following command:

```
lfs setstripe -c 4 .
```

After this has been done all new files created in the current directory will be spread over 4 storage arrays each having 1/4th of the file. The file can be accessed as normal no special action need to be taken. When the striping is set this way it will be defined on a per directory basis so different directories can have different stripe setups in the same file system, new subdirectories will inherit the striping from its parent at the time of creation.

We recommend users to set the stripe count so that each chunk will be approx. 200-300GB each, for example

File size	Stripe count	Command
500-1000GB	4	`lfs setstripe -c 4 .`
1TB - 2TB	8	`lfs setstripe -c 8 .`

Once a file is created the stripe count cannot be changed. This is because the physical bits of the data already are written to a certain subset of the storage arrays. However the following trick can used after one has changed the striping as described above:

```
$ mv file file.bu
$ cp -a file.bu file
$ rm file.bu
```

The use of `-a` flag ensures that all permissions etc are preserved.

30.9 Management of many small files (> 10000)

The file system on Stallo is designed to give good performance for large files. This have some impact if you have many small files.

If you have thousands of files in one directory. Basic operations like 'ls' becomes very slow, there is nothing to do about this. However directories containing many files may cause the backup of the data to fail. It is therefore highly recommended that if you want backup of the files you need to use 'tar' to create on archive file of the directory.

30.10 Compression of data

Data which is not accessed frequently like results of finished projects should be compressed in order to reduce storage space.

We recommend `xz` and `tar` to compress single files or whole folder structures. To compress a single file:

```
$ xz file
```

To decompress:

```
$ xz --decompress file
```

To create a archive multiple files or folder:

```
$ tar cfJv archive.tar.xz files
```

It is recommended to use the file suffix `.tar.xz` to make it clear that archive was compressed with `xz`.

To extract a archive (use `-C folder` to extract the files in folder):

```
$ tar xvf archive.tar.xz
```

30.11 Binary data and endianness

Stallo is like all desktop PCs a little endian computer.

At the moment in NOTUR the only big endian machine is njord.hpc.ntnu.no so Fortran sequential unformatted files create on Njord cannot be read on Stallo.

The best work around for this is to save your file in a portable file format like netCDF or HDF5.

Both formats are supported on stallo, but you have to load its modules to use them:

```
$ module load netCDF
```

Or:

```
$ module load HDF5
```

CHAPTER 31

Transferring files to/from Stallo

Only ssh type of access is open to stallo. Therefore to upload or download data only scp and sftp can be used.

To transfer data to and from stallo use the following address:

```
stallo.uit.no
```

This address has nodes with 10Gb network interfaces.

Basic tools (scp, sftp)

Standard scp command and sftp clients can be used:

```
ssh stallo.uit.no
ssh -l <username> stallo.uit.no

sftp stallo.uit.no
sftp <username>@stallo.uit.no
```

31.1 Mounting the file system on you local machine using sshfs

For linux users:

```
sshfs [user@]stallo.uit.no:[dir] mountpoint [options]
```

eg.:

```
sshfs steinar@stallo.uit.no:  /home/steinar/stallo-fs/
```

Windows users may buy and install expandrive.

31.1.1 High-performance tools

31.2 NONE cipher

This cipher has the highest transfer rate. Keep in mind that data after authentication is NOT encrypted, therefore the files can be sniffed and collected unencrypted by an attacker. To use you add the following to the client command line:

```
-oNoneSwitch=yes -oNoneEnabled=yes
```

Anytime the None cipher is used a warning will be printed on the screen:

```
"WARNING: NONE CIPHER ENABLED"
```

If you do not see this warning then the NONE cipher is not in use.

MT-AES-CTR

If for some reason (eg: high confidentiality) NONE cipher can't be used, the multithreaded AES-CTR cipher can be used, add the following to the client command line (choose one of the numbers):

```
-oCipher=aes[128|192|256]-ctr
```

or:

```
-caes[128|192|256]-ctr.
```

31.3 Subversion and rsync

The tools subversion and rsync is also available for transferring files.

CHAPTER 32

Lustre FS performance tips

32.1 How you can adjust Lustre depending on your application IO requirements.

32.2 Introduction

Lustre is a scalable, high performance, parallel I/O file system. More information about Lustre can be found on Wikipedia .

When using additional library for I/O (i.e. MPIIO, HDF5), reading and writing can be done in parallel from several nodes into single-shared file.

At the time of writing this guide Lustre is available on 2 NOTUR machines:

- Hexagon (/work)
- Stallo (/global/work)

All tests were performed on fully loaded machines, mean values of three repeats were used.

Lustre terminology and setup:

MDS is the MetaData Server which handles the information about files and directories. OSS is a object storage server that store file data on one or more object storage targets (OSTs). OST is the Object Storage Target, which is responsible for writing or reading the actual data to and from disk.

32.3 Striping

Lustre file striping defines the number of OSTs a file is written across. At the time of writing hexagon has 2 stripes with 1MB size. Stallo has by default 1 stripe (no striping). You can manage striping with the following tools/commands:

lfs setstripe - a command to change striping parameters. lfs getstripe - a command to get striping information. llapi - a set of C commands to manipulate striping parameters from C programs (llapi_file_create, llapi_file_get_stripe).

Note that the changed striping can only take effect for newly created files or files that are copied (not moved) into the directory.

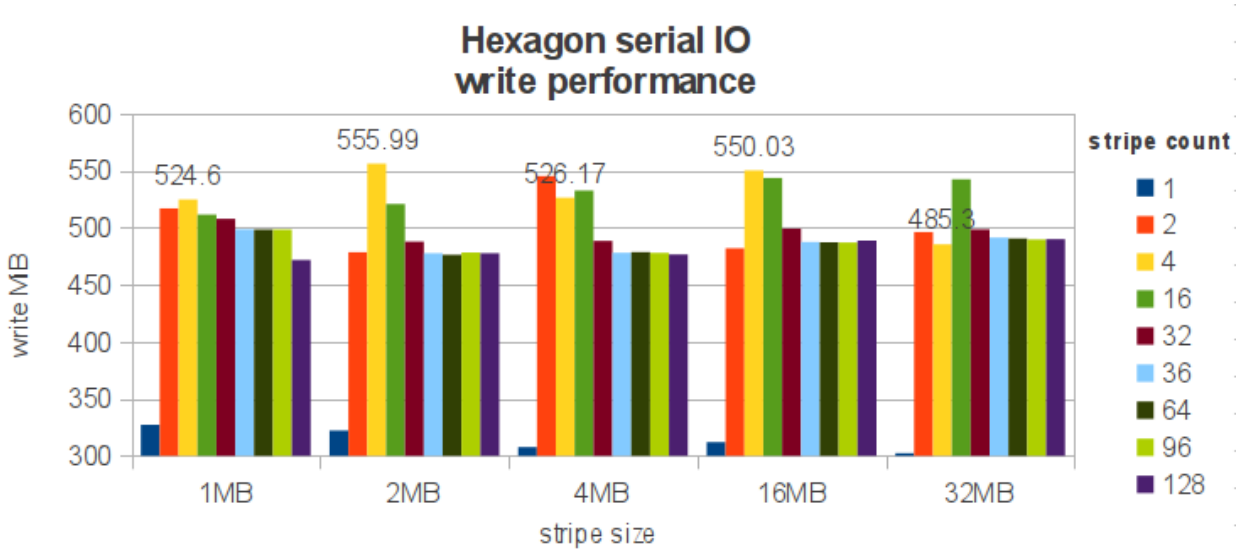
Examples:

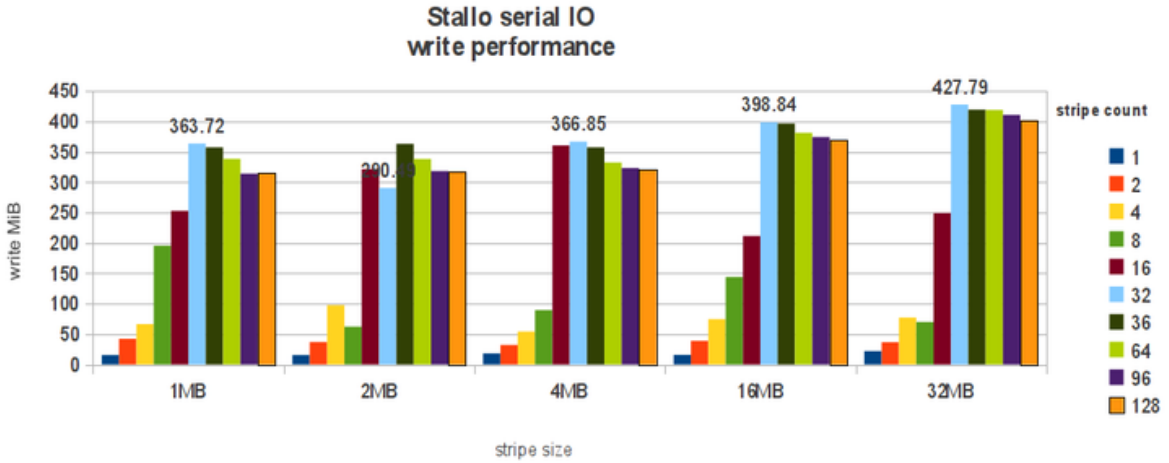
lfs setstripe –size 2M “dir” # will set stripe size for “dir” to 2M.

lfs setstripe –count 12 “dir” # will set that each file inside “dir” will be striped across 12 OSTs.

32.4 Serial IO

This is when a file or set of files are accessed by one process. Below are charts representing Hexagon and Stallo serial IO performance with different number of OSTs and different chunk sizes. It is true for both machines that to get better IO performance you have to stripe file across several OSTs, where for:



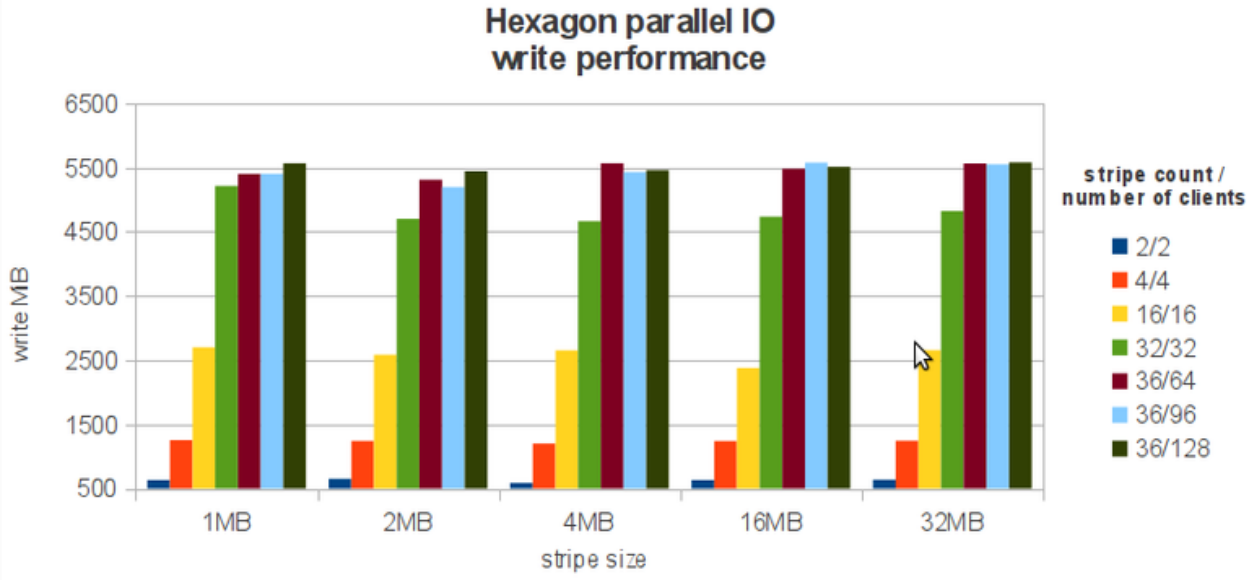


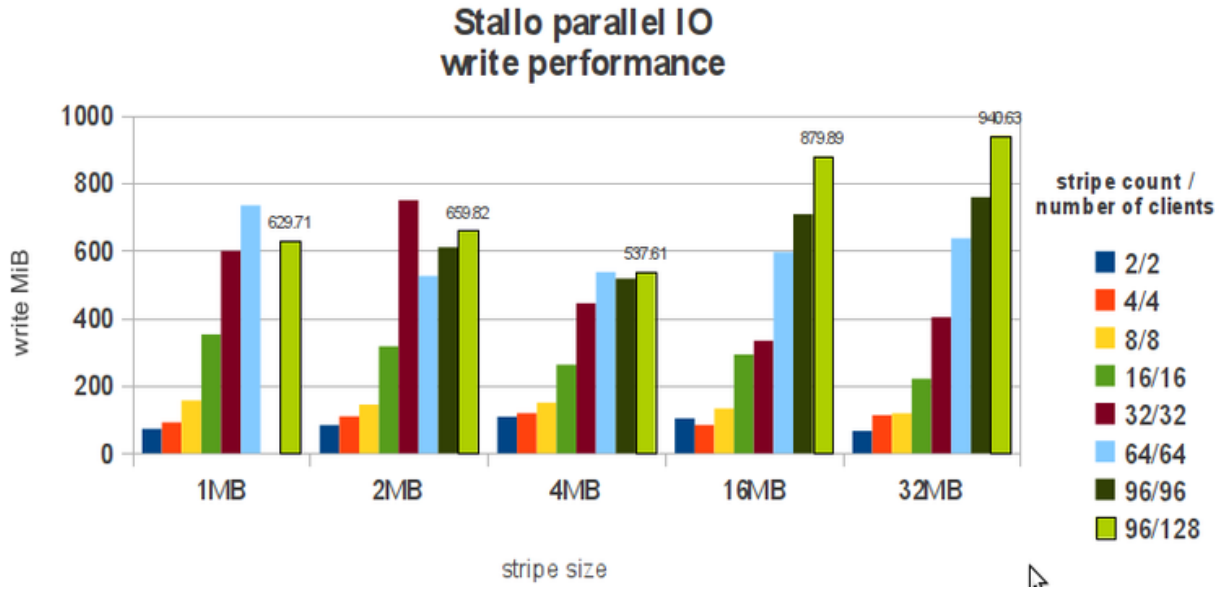
Hexagon optimal is using 2-4 OSTs, depending on the stripe size. Increasing chunk size is not much affecting hexagon. This can be related to the interconnect, where 1MB transfer size is a minimal size to get optimal performance.

Stallo, by using 8 OSTs you will speedup your data IO from default 25MIB/s to almost 200MIB/s! Maximum performance you will get with bigger chunk size and enough number of OSTs (32MB chunk and 32OSTs will give you 428MIB/s).

32.5 Parallel IO

Many processes writing into single-shared file. You will need to write at offsets or use parallel IO library, like MPIIO, HDF5. On both machines the same number of stripes as the number of clients have been used (up to the maximum number of OSTs).





The general rule, like "many clients – do stripe" works on both machines. Specific: - Hexagon. One to one ratio of clients to OSTs works fine (up to the maximum number of OSTs). Increasing chunk size is not affecting performance.

- Stallo, to get most out of the file system you will have to increase the chunk size to 32MB and when you have up to 96 clients stripe over as many OSTs as you have clients, when you are over 96 clients, keep number of OSTs to 96 to avoid contention.

32.6 General Tips

General striping recommendation:

- Many clients and many files: Do NOT stripe.
- Many clients one file: Do stripe.
- Some clients and few large files: Do stripe.

In addition:

- Use parallel IO (HDF5, MPIIO), this is the only way to get full advantage of the Lustre filesystem.
- Open files read-only whenever is possible.
- Keep small files on the same OST.

It is highly recommended to read I/O Lustre tips from NICS (http://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips).

CHAPTER 33

Compilers

The default development environment on Stallo is provided by Intel Cluster Studio XE. In general users are advised to use the Intel compilers and MKL performance libraries, since they usually give the best performance.

33.1 Fortran compilers

The recommended Fortran compiler is the Intel Fortran Compiler: ifort. The gfortran compiler is also installed, but we do not recommend it for general usage.

33.2 Usage of the Intel ifort compiler

For plain Fortran codes (all Fortran standards) the general form for usage of the Intel ifort compiler is as follows:

```
$ ifort [options] file1 [file2 ...]
```

where options represents zero or more compiler options, and fileN is a Fortran source (.f .for .ftn .f90 .fpp .F .FOR .F90 .i .i90), assembly (.s .S), object (.o), static library (.a), or an other linkable file. Commonly used options may be placed in the ifort.cfg file.

The form above also applies for Fortran codes parallelized with OpenMP (www.openmp.org, Wikipedia); you only have to select the necessary compiler options for OpenMP.

For Fortran codes parallelized with MPI the general form is quite similar:

```
$ mpif90 [options] file1 [file2 ...]
```

The wrapper mpif90 is using the Intel ifort compiler and invokes all the necessary MPI machinery automatically for you. Therefore, everything else is the same for compiling MPI codes as for compiling plain Fortran codes.

33.3 C and C++ compilers

The recommended C and C++ compilers are the Intel Compilers; icc (C) and icpc (C++). The gcc and g++ compilers are also installed, but we do not recommend them for general usage due to performance issues.

33.4 Usage of the Intel C/C++ compilers

For plain C/C++ codes the general form for usage of the Intel icc/icpc compilers are as follows:

```
$ icc  [options] file1 [file2 ...]  # for C
$ icpc [options] file1 [file2 ...]  # for C++
```

where options represents zero or more compiler options, fileN is a C/C++ source (.C .c .cc .cpp .cxx .c++ .i .ii), assembly (.s .S), object (.o), static library (.a), or other linkable file. Commonly used options may be placed in the icc.cfg file (C) or the icpc.cfg (C++).

The form above also applies for C/C++ codes parallelized with OpenMP (www.openmp.org, Wikipedia); you only have to select the necessary compiler options for OpenMP.

For C/C++ codes parallelized with MPI the general form is quite similar:

```
$ mpicc [options] file1 [file2 ...]  # for C when using OpenMPI
$ mpiCC [options] file1 [file2 ...]  # For C++ when using OpenMPI
```

Both mpicc and mpiCC are using the Intel compilers, they are just wrappers that invoke all the necessary MPI machinery automatically for you. Therefore, everything else is the same for compiling MPI codes as for compiling plain C/C++ codes.

CHAPTER 34

Environment modules

The user's environment, and which programs are available for immediate use, is controlled by the `module` command. Many development libraries are dependant on a particular compiler versions, and at times a specific MPI library. When loading and/or unloading a compiler, `module` automatically unloads incompatible modules and, if possible, reloads compatible versions.

Currently, not all libraries in all combinations of all compilers and MPI implementations are supported. By using the default compiler and MPI library, these problems can be avoided. In the future, we aim to automate the process so that all possible (valid) permutations are allowed.

Read the *Software Module Scheme* section for an introduction on how to use modules.

CHAPTER 35

Profiling and optimization

In general, in order to reach performances close to the theoretical peak, it is necessary to write your algorithms in a form that allows the use of scientific library routines, such as BLACS/LAPACK.

35.1 Arm Performance Reports

Arm Performance Reports offers a nice and convenient way to get an overview profile for your run very quickly. It will introduce a typically negligible runtime overhead and all you need to do is to load the `perf-reports` module and to launch your "normal" execution using the `perf-report` launcher.

Here is an example script:

```
#!/usr/bin/env bash

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
#SBATCH --time=0-00:10:00

module load perf-reports/5.1

# create temporary scratch area for this job on the global file system
SCRATCH_DIRECTORY=/global/work/$USER/$SLURM_JOBID
mkdir -p $SCRATCH_DIRECTORY

# run the performance report
# all you need to do is to launch your "normal" execution
# with "perf-report"
cd $SCRATCH_DIRECTORY
perf-report mpiexec -n 20 $SLURM_SUBMIT_DIR/example.x

# perf-report generates summary files in html and txt format
# we copy result files to submit dir
cp *.html *.txt $SLURM_SUBMIT_DIR
```

(continues on next page)

(continued from previous page)

```
22
23 # clean up the scratch directory
24 cd /tmp
25 rm -rf $SCRATCH_DIRECTORY
```

What we do there is to profile an example binary located in `$SLURM_SUBMIT_DIR/example.x`.

The profiler generates summary files in html and txt format and this is how an example html summary can look (open it in your browser):

Summary: wave_c is Compute-bound in this configuration

Compute	99.9%	Time spent running application code. High values are usually good. This is **very high**; check the CPU performance section for advice.
MPI	0.1%	Time spent in MPI calls. High values are usually bad. This is **very low**; this code may benefit from a higher process count.
I/O	0.0%	Time spent in filesystem I/O. High values are usually bad. This is **negligible**; there's no need to investigate I/O performance.

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

As very little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU

A breakdown of the 99.9% CPU time:

Scalar numeric ops	16.0%
Vector numeric ops	0.0%
Memory accesses	25.2%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

No time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

MPI

A breakdown of the 0.1% MPI time:

Time in collective calls	100.0%
Time in point-to-point calls	0.0%
Effective process collective rate	8.92 kB/s
Effective process point-to-point rate	0.00 bytes/s

Most of the time is spent in collective calls with a very low transfer rate. This suggests load imbalance is causing synchronization overhead; use an MPI profiler to investigate.

I/O

A breakdown of the 0.0% I/O time:

Time in reads	0.0%
Time in writes	0.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	0.00 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

Threads

A breakdown of how multiple threads were used:

Computation	0.0%
Synchronization	0.0%
Physical core utilization	5.0%
System load	5.1%

No measurable time is spent in multithreaded code.

Physical core utilization is low. Try increasing the number of processes to improve performance.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	66.8 MB
Peak process memory usage	74.9 MB
Peak node memory usage	3.0%

The peak node memory usage is very low. Running with fewer MPI processes and more data on each process may be more efficient.

35.2 Performance tuning by Compiler flags

35.2.1 Quick and dirty

Use `ifort/icc -O3`. We usually recommend that you use the `ifort/icc` compilers as they give superior performance on Stallo. Using `-O3` is a quick way to get reasonable performance for most applications. Unfortunately, sometimes the compiler break the code with `-O3` making it crash or give incorrect results. Try a lower optimization, `-O2` or `-O1`, if this doesn't help, let us know and we will try to solve this or report a compiler bug to INTEL. If you need to use `-O2` or `-O1` instead of `-O3` please remember to add the `-ftz` too, this will flush small values to zero. Doing this can have a huge impact on the performance of your application.

35.2.2 Profile based optimization

The Intel compilers can do something called profile based optimization. This uses information from the execution of the application to create more effective code. It is important that you run the application with a typical input set or else the compiler will tune the application for another usage profile than you are interested in. With a typical input set one means for instance a full spatial input set, but using just a few iterations for the time stepping.

1. Compile with `-prof-gen`.
2. Run the app (might take a long time as optimization is turned off in this stage).
3. Recompile with `-prof-use`. The simplest case is to compile/run/recompile in the same catalog or else you need to use the `-prof-dir` flag, see the manual for details.

35.3 Vtune

Intel Vtune Amplifier is a versatile serial and parallel profiler, with features such as stack sampling, thread profiling and hardware event sampling.

35.4 Totalview

Totalview is a source- and machine-level debugger for multi-process, multi-threaded programs.

CHAPTER 36

Debugging

36.1 Debugging with TotalView

TotalView is a graphical, source-level, multiprocess debugger. It is the primary debugger on Stallo and has excellent capabilities for debugging parallel programs. When using this debugger you need to turn on X-forwarding, which is done when you login via ssh. This is done by adding the -Y on newer ssh version, and -X on older:

```
$ ssh -Y username@stallo.uit.no
```

The program you want to debug has to be compiled with the debug option. This is the "-g" option, on Intel and most compilers. The executable from this compilation will in the following examples be called "filename".

First, load the totalview module to get the correct environment variables set:

```
$ module load TotalView
```

To start debugging run:

```
$ totalview MyProg
```

Which will start a graphical user interface.

Once inside the debugger, if you cannot see any source code, and keep the source files in a separate directory, add the search path to this directory via the main menu item File->Search path.

Source lines where it is possible to insert a breakpoint are marked with a box in the left column. Click on a box to toggle a breakpoint.

Double clicking a function/subroutine name in a source file should open the source file. You can go back to the previous view by clicking on the left arrow on the top of the window.

The button "Go" runs the program from the beginning until the first breakpoint. "Next" and "Step" takes you one line / statement forward. "Out" will continue until the end of the current subroutine/function. "Run to" will continue until the next breakpoint.

The value of variables can be inspected by right clicking on the name, then choose "add to expression list". The variable will now be shown in a pop up window. Scalar variables will be shown with their value, arrays with their dimensions and type. To see all values in the array, right click on the variable in the pop up window and choose "dive". You can now scroll through the list of values. Another useful option is to visualize the array: after choosing "dive", open the menu item "Tools->Visualize" of the pop up window. If you did this with a 2D array, use middle button and drag mouse to rotate the surface that popped up, shift+middle button to pan, Ctrl+middle button to zoom in/out.

Running totalview on an interactive node:

```
$ mkdir -p /global/work/$USER/test_dir
$ cp $HOME/test_dir/a.out /global/work/$USER/test_dir
$ cd /global/work/$USER/test_dir
$ module load TotalView
$ totalview a.out
```

Replace [#procs] with the core-count for the job.

A window with name "New Program" should pop up. Under "Program" write the name of the executable. Under "Parallel" choose "Open MPI" and "Tasks" is the number of cores you are using ([#procs]).

You can also start Totalview with:

```
$ mpirun -tv a.out
```

The users guide and the quick start quide for Totalview can be found on the RogueWave documentation page.

36.2 Debugging with Valgrind

- A memory error detector
- A thread error detector
- A MPI error detector
- A cache and branch-prediction profiler
- A call-graph generating cache profiler
- A heap profiler
- Very easy to use

Valgrind is the best thing for developers since the invention of pre-sliced bread!

Valgrind works by emulating one or more CPUs. Hence it can intercept and inspect your unmodified, running program in ways which would not be otherwise possible. It can for example check that all variables have actually been assigned before use, that all memory references are within their allowed space, even for static arrays and arrays on the stack.

What makes Valgrind so extremely powerful is that it will tell exactly where in the program a problem, error or violation occurred. It will also give you information on how many allocates/deallocates the program performed, and whether there is any unreleased memory or memory leaks at program exit. In fact, it goes even further and will tell you on what line the unreleased/leaking memory was allocated. The cache profiler will give you information about cache misses and where they occur.

The biggest downside to Valgrind is that it will make your program run much slower. How much slower depends on what kind of, and how much, information you request. Typically the program will run 10-100 times slower under Valgrind.

Simply start Valgrind with the path to the binary program to be tested:

```
$ module load Valgrind
$ valgrind /path/to/prog
```

This runs Valgrind with the default "tool" which is called memcheck, which checks memory consistency. When run without any extra flags, Valgrind will produce a balanced, not overly detailed and informative output. If you need a more detailed (but slower) report, run Valgrind with:

```
$ valgrind --leak-check=full --track-origins=yes --show-reachable=yes /path/to/prog
```

Of course, if you want to get all possible information about where in the program something was inconsistent you must compile the program with debugging flags switched on.

If you have a multi-threaded program (e.g. OpenMP, pthreads), and you are unsure if there might be possible deadlocks or data races lurking in your program, the Valgrind thread checker is your best friend. The thread checking tool is called helgrind:

```
$ export OMP_NUM_THREADS=2
$ valgrind --tool=helgrind /path/to/prog
```

For more information on using Valgrind please refer to the man pages and the Valgrind manual which can be found on the Valgrind website: http://www.valgrind.org