HPC Documentation

Release 0.0

HPC group

Jun 14, 2017

News and Events

1 News and notifications 3

2 We have migrated to the SLURM resource manager 5

3 New software and module scheme 7

4 Schrodinger Workshop 9

5 Frequently asked questions 11

6 X11 connection problems 19

7 Contact 21

8 How to write good support requests 23

9 Support staff 27

10 Where to find us on the campus 29

11 About Stallo 31

12 Guidelines for use of computer equipment at the UiT The Arctic University of Norway 33

13 Getting an account 37

14 Logging in for the first time 39

15 CPU-hour quota and accounting 43

16 Linux command line 45

17 Dos and don'ts 49

18 Job script examples 51

19 Accounts 59

20 Batch system 61

21 Job related environment variables 63

22 Quick Guide to translate PBS/Torque to SLURM 65

23 Managing jobs 67

24 Walltime 69

25 Process count 71

26 Partitions (queues) and services 73

27 Interactive job submission 75

28 Monitoring your jobs 77

29 Which software is installed on Stallo 79

30 Old Scheme Modules 81

31 New Scheme Modules 83

32 Application guides 85

33 Storage and backup 99

34 Transferring files to/from Stallo 103

35 Lustre FS performance tips 105

36 Compilers 109

37 Environment modules 111

38 Profiling and optimization 113

39 Debugging 117

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

CHAPTER 1

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.

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).

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.

Fig. 1.1: Credit: Futurama (http://futurama.wikia.com/wiki/Slurms_MacKenzie)

CHAPTER 2

We have migrated to the SLURM resource manager

As of October 1st 2016 stallo.uit.no will have a new official queuing system, named SLURM (Simple Linux Utility for Resource Management).

To get a soft start for all, the old queuing system will still be functional and accept jobs as usual. We will only start out with 152 nodes in the SLURM part of the cluster, the rest will stay with the old torque. Slowly we will move more and more nodes from torque into SLURM, as nodes free up from running jobs.

Since the cluster has 2 queuing systems you will have to check both queues to get the full picture of the load on the system, this is of course unfortunate but it has to stay this way until the transition is fully completed.

The highmem nodes will remain in torque for now.

Jobs already submitted to torque will stay within this queue, so if you would like to move your jobs to SLURM you have to resubmit yourself.

If you use the Abel cluster at UiO you are already familiar with SLURM and should find it rather easy to switch. The new NOTUR supercomputers will also run SLURM.

The torque software has served us well for many years, however it is no longer actively maintained. We have encountered bugs in the system that diminish the utilization of the cluster, as this is not acceptable we have chosen to switch to SLURM. By this we hope that you will find the user experience across Norwegian HPC systems will be more uniform.

CHAPTER 3

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:

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>/
```

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.

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.

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.

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.

For information about how to reach the "new" packages, before we make this scheme default, see in the software section *New Scheme Modules*.

CHAPTER 4

Schrodinger Workshop

Schrodinger workshop in Tromso February 22nd and 23rd. Day one will be typically beginner stuff and day two advanced.

CHAPTER 5

Frequently asked questions

Passwords

I forgot my password - what now?

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

How do I change my password on Stallo?

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

Installing software

I need a Python package X but the one on Stallo is too old or I can not find it

I need a newer version of Scipy, Numpy, etc. Can you install it?

We often have newer versions of software packages installed which may not be visible with the default user settings. Find out more about it here: http://hpc.uit.no/en/latest/news/new_sw.html

In cases where this still doesn't solve your problem and you would like to install it 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.

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

Yes. The recommended way to achieve this is using virtual environments: http://docs.python-guide.org/en/latest/dev/virtualenvs/

Example (as an example we install the Biopython package):

```
$ module load GCC/4.9.1
$ 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
```

And you do not have to call it "venv". It is no problem to have many virtual environments in your home directory.

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
```

Running software

Why is a specific node so incredibly slow compared to others?

The node is probably swapping.

What does swapping mean and why should I care?

If the jobs consume more memory than the node physically has, the node starts to swap out memory to the disk. This typically means a significant slowdown of the calculation. And this is why you need to care about swapping: your calculation will slow down to a grinding halt. You can also crash the node which is bad for us.

How can I check whether my calculation is swapping?

Option 1 (inside the university network) is to check http://stallo-login2.uit.no/slurmbrowser/html/squeue.html. Click on "nodes" and then the node in question. On the right hand panel you see "Memory last hour". If memory is above the red mark, the node will swap.

Option 2 is to log into the node and run "top". On the top you see how much memory is consumed and whether the node is swapping.

Compute and storage quota

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.

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
```

Connecting via ssh

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

If you needs 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.

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
```

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.

"X11 connection rejected because of wrong authentication"-problem

Please look at our documentation of this problem here: *X11 connection problems*.

Jobs and queue system

I am not able to submit jobs longer than two days

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

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*.

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.

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
```

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. Currently only the 20 core nodes have been enabled for the SLURM batch system, these have no local disk and thus no swap space. 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, it seems to be a sort of buffer within SLURM no allowing you to consume absolutely all memory available (system needs some to work). 2000MB seems to work fine, but not 2 GB for 16 cores/node.

The solution we want to push in general, see *First time you run a Gaussian job?*, is this:

Specify number of tasks:

```
#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.

Step memory limit

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

For slurm, the memory flag seems to be 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 pr core. With memory intensive applications like comsol or VASP, your job will likely be terminated. The solution to this problem is to, like we have said elsewhere, 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, the user has asked for more time than is allowed for the QOS associated with the job.

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.

CPU v.s. core

In this documentation we are frequently using the term *CPU*, which in most cases are equivalent to the more precise term *processor core* / *core*. The *multi core age* is here now *:-)*

Running many short tasks

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

Background

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.

Without going into any more details, let's look at the solution.

Running tasks in parallel within one job

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 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)

  # 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

X11 connection problems

Over the last 6-12 months we have had an increasing number of users experiencing problems when connecting to stallo with X11 forwarding enabled. We have put a lot of effort in investigating and trying to solve this but have so far not been able to come up with a solution. We have some solutions that seem to temporarily solve the problem for some users.

We are now asking for help on this because we are not able to solve it alone and for the impacted users this is a serious problem.

Here we will collect all information we have including the solutions we know have worked for some.

Description of the problem

The errormessages reported, os-es and browsers we know it apers on.

List of solutions we have tried

Espen's solution on his Mac:

For Mac OS X Sierra (10.12) it is related to the fact that the xauth binary has a non-standard placement, causing ssh to not finding it when logging in with X11 tunelling enabled (ssh -Y).

Be ware that for the OS X platform, this also inflects the usage of the safari browser when logging in to stallo-gui, since the browser seems to read the default path setup from the system. Using the Chrome browser solves the X11 problem on stallo-gui for OSX.

The overall sollution for Espen on his Mac was the following:

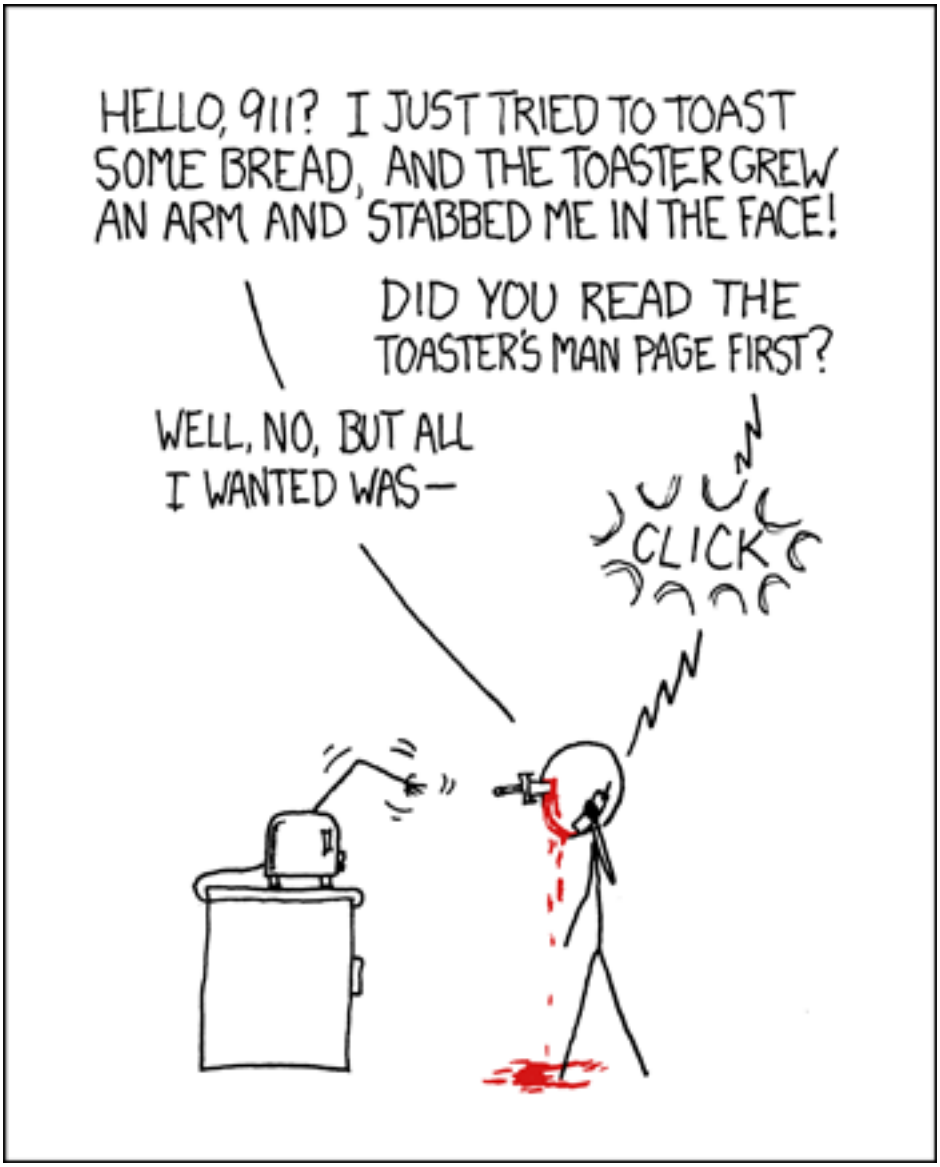
- sudo emacs -nw /etc/ssh/ssh_config
- Go to the line that starts with "# Host *" (line 20 in my config)
- Added "XAuthLocation /usr/X11/bin/xauth" under ForwardAgent and ForwardX11.

- Closed the file and closed all terminal windows.
- Opened a terminal window and logged in to Stallo. Problem gone.
- Since he had multiple user accounts on Stallo, he also had to add "X11Forwarding yes" to your sshd_config file.

CHAPTER 7

Contact

If you need help, please file a support request via support-uit@notur.no, and our local team of experts will try to assist you as soon as possible.



Credit: http://xkcd.com/293/

Alternatively you can use http://support.notur.no to submit and track your requests. The necessary user name and password are generated the first time you send an email to support-uit@notur.no.

Postal address

```
HPC group
IT-avdelingen Teo2 4. etg.
UiT Norges Arktiske Universitet
9037 Tromsø
```

CHAPTER 8

How to write good support requests

Writing good support requests is not only good for the support team, it is also better for you because we will understand your request sooner and will be able to resolve it faster. Below is a list of good practices.

Never send support requests to staff members directly

Always send them to support-uit@notur.no and staff members will pick them up there. On support-uit@notur.no they get tracked and have higher visibility. Some staff members work on the support line only part time. Sending the request to support-uit@notur.no makes sure that somebody will pick it up.

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.

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.

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.

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.

Be specific

The better you describe the problem the less we have to guess and ask. Make the problem reproducible. See also next point.

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.

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.

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.

CHAPTER 9

Support staff

Our vision

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

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.

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.

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. On January 1st 2005 a new limited company within Uninett (Sigma) was established to administrate the national HPC program. The four Universities will act as Resource Sites as before.

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.

Our team

- Radovan Bast
- Roy Dragseth
- Steinar Henden
- Dan Jonsson
- Elena Malkin
- Espen Tangen
- Giacomo Tartari

CHAPTER 10

Where to find us on the campus

CHAPTER 11

About Stallo

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 144 x HP Apollo 8000 gen8 servers	1 x HP BL460 gen8 blade servers 1 x. HP SL230 gen8 servers 1 x. HP apollo 8000 gen8 servers
# CPU's / # Cores	608 / 4864 656 / 6560 288 / 5760	2 / 16 2 / 20 2 / 20
Processors	608 x 2.60 GHz Intel Xeon E5 2670 656 x 2.80 GHz Intel Xeon E5 2680 288 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 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	13
Infrastructure racks	2
Storage racks	3

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

Stallo - a Linux cluster

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

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.

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.

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

Since 2003, the HPC-group at has been one of five international development sites for the Linux operation 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.

Stallo - Norse 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 12

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

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.

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.

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.

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.

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.

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).

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.

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.

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.

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.

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.

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.

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 13

Getting an account

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 Notur can be found at https://www.notur.no/use-notur-resources

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:

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.

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 wan 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-uit@notur.no.

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 14

Logging in for the first time

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 njord.hpc.ntnu.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 [user name]
```

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>
```

Log in with an ssh-key

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.

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*).

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.

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.

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 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.
```

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 15

CPU-hour quota and accounting

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.

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*

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

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 16

Linux command line

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 storage section.

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.

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*.

Running jobs on the machine

You must execute your jobs by submitting them to the batch system. There is a dedicated section jobs 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).

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).

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'.

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.

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
```

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]
```

~/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 17

Dos and don’ts

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

CHAPTER 18

Job script examples

Help! I don't know what OpenMP or MPI means!

OpenMP and MPI are parallelization frameworks. If you want to run many similar jobs that each use one core at a time, scroll down to job arrays.

Example for an OpenMP job

```
#!/bin/bash

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

#SBATCH --job-name=example

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

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

# short partition should do it
#SBATCH --partition short

# 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 of available cores
export OMP_NUM_THREADS=${SLURM_TASKS_PER_NODE}

# 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
```

Save it to a file (e.g. run.sh) and submit it with:

```
$ sbatch run.sh
```

Example for a MPI job

```
#!/bin/bash

############################
# 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

# short partition should do it
#SBATCH --partition short

```

```
# 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 ./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
```

Save it to a file (e.g. run.sh) and submit it with:

```
$ sbatch run.sh
```

Example for a hybrid MPI OpenMP job

```
#!/bin/bash

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

#SBATCH --job-name=example

# we ask for 2 MPI tasks with 20 cores each
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=20

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

# short partition should do it
#SBATCH --partition short
```

```
# 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 ./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
```

Save it to a file (e.g. run.sh) and submit it with:

```
$ sbatch run.sh
```

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
```

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

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:

```
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

#####################
# 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

# short partition should do it
#SBATCH --partition short

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

# happy end
exit 0
```

Sumbit 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
```

Observe that they all started (approximately) at the same time:

```
$ grep start *txt

output_1.txt:start at 14:43:58
output_10.txt:start at 14:44:00
output_11.txt:start at 14:43:59
output_12.txt:start at 14:43:59
output_13.txt:start at 14:44:00
output_14.txt:start at 14:43:59
output_15.txt:start at 14:43:59
output_16.txt:start at 14:43:59
output_2.txt:start at 14:44:00
output_3.txt:start at 14:43:59
output_4.txt:start at 14:43:59
output_5.txt:start at 14:43:58
output_6.txt:start at 14:43:59
output_7.txt:start at 14:43:58
output_8.txt:start at 14:44:00
output_9.txt:start at 14:43:59
```

Example on how to allocate entire memory on one node

```
#!/bin/bash

#######################################################
# example for a job where we consume lots 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

# short partition should do it
#SBATCH --partition short

# 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
```

CHAPTER 19

Accounts

Account information

Information on available CPU-hours in your accounts:

```
$ sbank balance statement -u
```

How to set the account in your job script

You can set it like this:

```
#SBATCH --account=nn1234k
```

CHAPTER 20

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.)

If you are already used to Torque/Maui (the previous queue system used on Stallo), but not SLURM, you might find this *Quick Guide to translate PBS/Torque to SLURM* useful.

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*.

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.

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

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

```
...

# argument 1 is the input file
# argument 2 is the output file
mybinary.x < ${1} > ${2}
```

CHAPTER 21

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
```

CHAPTER 22

Quick Guide to translate PBS/Torque to SLURM

User commands	PBS/Torque	SLURM
Job submission	qsub [filename]	sbatch [filename]
Job deletion	qdel [job_id]	scancel [job_id]
Job status (by job)	qstat [job_id]	squeue --job [job_id]
Full job status (by job)	qstat -f [job_id]	scontrol show job [job_id]
Job status (by user)	qstat -u [username]	squeue --user=[username]

Environment variables	PBS/Torque	SLURM
Job ID	$PBS_JOBID	$SLURM_JOBID
Submit Directory	$PBS_O_WORKDIR	$SLURM_SUBMIT_DIR
Node List	$PBS_NODEFILE	$SLURM_JOB_NODELIST

Job specification	PBS/Torque	SLURM
Script directive	#PBS	#SBATCH
Job Name	-N [name]	--job-name=[name] OR -J [name]
Node Count	-l nodes=[count]	--nodes=[min[-max]] OR -N [min[-max]]
CPU Count	-l ppn=[count]	--ntasks-per-node=[count]
CPUs Per Task		--cpus-per-task=[count]
Memory Size	-l mem=[MB]	--mem=[MB] OR --‘mem-per-cpu=[MB]
Wall Clock Limit	-l walltime=[hh:mm:ss]	--time=[min] OR --time=[days-hh:mm:ss]
Node Properties	-l nodes=4:ppn=8:[property]	--constraint=[list]
Standard Output File	-o [file_name]	--output=[file_name] OR -o [file_name]
Standard Error File	-e [file_name]	--error=[file_name] OR -e [file_name]
Combine stdout/stderr	-j oe (both to stdout)	(Default if you don’t specify --error)
Job Arrays	-t [array_spec]	--array=[array_spec] OR -a [array_spec]
Delay Job Start	-a [time]	--begin=[time]

CHAPTER 23

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.

CHAPTER 24

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.

CHAPTER 25

Process count

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.

For single node jobs, just ask for less than 16 cores with less than 1.6GB of memory/core.

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*.

CHAPTER 26

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:

short: For testing and debugging. Up to 1 hour of walltime.

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.

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

```
$ sinfo --format="%P %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.

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-uit@notur.no.

Limit	Value
Max number of running jobs	1024
Maximum cpus per job	2048
Maximum walltime	28 days
Maximum memory per job	No limit:sup:*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 27

Interactive job submission

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.

CHAPTER 28

Monitoring your jobs

For details run the command with the --help option.

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

sacct -j <jobid> –format=JobID,JobName,MaxRSS,Elapsed To get 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

CPU load and memory consumption of your job

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).

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.

CHAPTER 29

Which software is installed on Stallo

To find out what software packages are available, type:

```
module avail
```

Missing or new software

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

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 30

Old Scheme Modules

Software installations on the clusters span many applications, and many different version numbers of the same application. 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. In order to simplify the control of which application versions are available in a specific session, there is a system for loading and unloading 'modules' which activate and deactivate the relevant parts of your user session.

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

```
module --help
```

Below we listed the most commonly used options.

Which modules are currently loaded?

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

```
module list
```

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

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/3.6.2
```

How to unload a module

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

```
module unload netcdf
```

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.

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 14.0, type:

```
module switch intel intel/14.0
```

CHAPTER 31

New Scheme Modules

We are working on a new scheme for installing, updating and maintaining software packages, including making modules and module files schemes. **All new software is and will be installed according to this scheme.** For several reasons we can not make these installations as the default setting for all users just yet, but we found it beneficial that you at least have access to them. Please note, that we might still do some changes without prior notice. In particular, the name of the packages and the behavior of lmod commands such as *module load*, *module avail*,*module spider*, etc. is likely to change. If you have trouble finding something, or have some questions,please, check this manual and contact us.

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.

If you would like to use these modules, execute the following 2 commands:

```
ml notur
source mod_setup.sh
```

Now work as normal. The module *varset* needs to be kept loaded for some variables to be set correctly and things to work. To see which modules you now have available do

```
module avail
```

You should be able to see and load both the "old" software packages and the "new" ones. For example, you should be able to load both intel/13.0 and intel/2016a, but note the difference in the behavior of these 2 packages.

Feedback

Feedback is appreciated. If you would like to give us feedback or if you have questions, please send an email to our usual address support-uit@notur.no.

Tips

If you want to have your default settings back without logging out and back in, do:

```
ml notur
source undo_mod_setup.sh
```

If you want to have this setup as your default setup:

```
ml notur
source new_default.sh
```

These settings are then saved as your default and you will have them when you log in. The module *varset* needs to be kept loaded for some variables to be set correctly and things to work.

If you want to go back to your old defaults:

```
ml notur
source undo_new_default.sh
```

CHAPTER 32

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 applications.

The GAUSSIAN program system

A description of the quantum chemistry program system Gaussian

General Information:

Gaussian is a computational chemistry software program system initially released in 1970. Gaussian has a rather low user threshold and a tidy user setup, which, together with a broad range of possibilities and a graphical user interface (gaussview), might explain its popularity in academic institutions.

Online info from vendor:

- Homepage: http://www.gaussian.com
- Documentation: http://www.gaussian.com/g_tech/g_ur/g09help.htm

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.

Additional online info about Gaussian on Stallo:

GaussView for Gaussian

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.

GaussView on Stallo

The GaussView version for Gaussian 09 is release 5.0.8 To load and run GaussView on Stallo, you first have to load the relevant gaussian module, and then call gaussview:

```
$ module load Gaussian/09.d01
$ gview
```

More information about GaussView:

- General info about the program: http://www.gaussian.com/g_prod/gv5.htm
- Online reference: http://www.gaussian.com/g_tech/gv5ref/gv5ref_toc.htm

First time you run a Gaussian job?

This page contains info aimed at first time users of Gaussian 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, copy the input example and the job script example shown below into your test job folder (which I assume you have created in advance).

Gaussian input example:

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

No comment specified. Default Comment: From NewCaffeine.xyz

0 1
 N      3.831536     0.000000     0.062646
 C      2.359623     0.000000     0.016662
 C      2.346242     0.000000     1.466600
 N      3.391185     0.000000     1.965657
 C      4.217536     0.000000     1.154419
 N      1.092573     0.000000     2.175061
 C      0.000000     0.000000     1.440000
 N      0.000000     0.000000     0.000000
 C      1.179579     0.000000    -0.825950
 C     -1.305083     0.000000    -0.608570
 O     -1.345306     0.000000     1.827493
 O      1.192499     0.000000    -2.225890
 C      1.079284     0.000000     3.614999
 C      4.767291     0.000000    -1.031868
 H     -2.067983     0.000000     0.168543
 H     -1.417118     0.889165    -1.227242
 H     -1.417118    -0.889165    -1.227242
 H      0.049258     0.000000     3.968508
 H      1.589271    -0.889165     3.982721
 H      1.589271     0.889165     3.982721
 H      4.222791     0.000000    -1.974970
```

```
H      5.393373      0.889165     -0.974180
H      5.393373     -0.889165     -0.974180
H      5.288201      0.000000      1.353412
```

NB: Pay special attention to the %mem defined in the beginning of the file. Also note that there are no mentioning of Linda and shared memory cores. This is adressed here:

About the Gaussian install on Stallo

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

Gaussian on Stallo:

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

- The parallel faction of gaussian executables scales a lot "longer" (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 nodal adress 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, and is commented in jobscript example file. Please do use this script or similar when submitting jobs; it will benefit all of us.

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

About memory allocation for Gaussian:

- In general, 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 automaticall scale down the number of cores to mirror the need. This also means that you allways should ask for full nodes when submitting Gaussian 09 jobs!

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

- In general it means memory/node – for share between nprocshared, and aditional to the memory allocated pr. process. This is also documented by Gaussian.
- For the mother superior (mother process/node) it also represents the network buffer allocated by Linda. So basically G09 takes a part and Linda takes a part the same size – 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. Basically, the general %mem limit will allways 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.

For core-count, node-count and amounts of memory on Stallo, see :ref:‘about_stallo´.

Scalability and parallel performance of Gaussian on Stallo:

Due to the nifty trick mentioned above, we have a somewhat more generous policy when it comes to allocating cores/nodes to Gaussian jobs:

1. We do advice people to use up to 16 nodes, at least 12 full nodes, with the current install on Stallo. We have proved acceptale scaling of the current Gaussian install beyond 16 nodes with 16 cores on each. So in essence; at least up to 16 nodes. Enjoy.
2. Linda networking overhead seems to hit hard around this amount of cores; causing us to be somewhat reluctant to advice going to more than this. On the other hand, this will give you a maximum of 320 cores with the most recent hardware available on Stallo.

We have been looking into issues related to this, and expect this situation to change when the new Gaussian Major release comes out late 2016/ early 2017.

Gaussian runscrip example:

```
#!/bin/bash -l
################### Gaussian Job Batch Script Example ###################
# Section for defining queue-system variables:
#-------------------------------------
# This script asks for a given set nodes and cores/node. Stallo has got 20 cores/node,
# asking for full nodes will make your life easier!)
# Runtime for this job is 59 minutes; syntax is dd-hh:mm:ss.
# Memory is set to 1500MB; but does really not matter since you are using full node.
# Though it is worth noting that memory settings for slurm seems to be a hard limit,
# it can be specified pr core or total pr job/node (be carefull with the latter).
#-------------------------------------
# SLURM-section
#SBATCH --job-name=g09_runex
#SBATCH -N 2
#SBATCH --ntasks-per-node=20
#SBATCH --time=00-00:59:00
#SBATCH --mem-per-cpu=1500MB
#SBATCH --output=g09_runex.log
#SBATCH --mail-type=ALL
#SBATCH --partition=gaussian
#####################################
# Section for defining job variables and settings:
#-------------------------------------
# Area for defining variables:

input=caffeine # Name of input without extention
extention=com # We use the same naming scheme as the software default

#-------------------------------

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

module load notur
module load Gaussian/09.d01

# Check other available versions with "module avail gaussian"

#-------------------------------
```

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

export GAUSS_SCRDIR=/global/work/$USER/$SLURM_JOB_ID

mkdir -p $GAUSS_SCRDIR
echo " The job will use scratch directory ${GAUSS_SCRDIR}." # Message written to log␣
↪for safety measure.

# Preparing and moving inputfiles to tmp:

cd $SUBMITDIR
cp $input.com $GAUSS_SCRDIR

# Checking for old job files to do restart from:
if [ -f $input.chk ]
then
        echo "Copying chk-file to scratch."
        cp $input.chk $GAUSS_SCRDIR
else
        echo "No chk file found."
        echo "Starting the gaussian job without a checkpointfile."
fi

cd $GAUSS_SCRDIR

# Preparation of inputfile is done by G09.prep.slurm in folder $g09tooldir
# If you want to inspect it, cd $g09tooldir after loading the gaussian module

G09.prep $input

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

# Running the program:

time g09 < $input.com > gaussian_$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 gaussian_$input.out $SUBMITDIR
cp $input.chk $SUBMITDIR

# To zip some of the output might be a good idea!

# Investigate potentially other files to keep:
echo `pwd`
echo `ls -ltr`



# ALLWAYS clean up after yourself. Please do uncomment the following line
#cd $SUBMITDIR
#rm  $GAUSS_SCRDIR/*
#rmdir $GAUSS_SCRDIR
```

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

NB: Note that we for Gaussian advice to specify both nodes and core/node. This is due to the special way Gaussian sets up parallel jobs!

These files are also available on Stallo:

To find the jobscript example(s), type the following:

```
module load notur
cd $runex/Gaussian
ls
```

Copy the contents of this folder to your home directory.

Then, submit the job by typing:

```
sbatch job_g09.sh
```

Compare the energy of g09_caffeine.out with the correct energy of HF=-680.4153661 for this calculation. The energy should ideally be identical or close to identical. After that, you may alter the variables in the shell script as much as you like to adapt to your own jobs. Good luck.

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).

License information:

The license of GAUSSIAN is commercial/proprietary.

The license of Gaussian contitutes of 4 site licenses for the 4 current host institutions of NOTUR installations; NTNU, UiB, UiO, UiT. In principle, only person from one of these instituions have access to the Gaussian Software system installed on Stallo.

- 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 either to be a member of the abovementioned institutions or provide proof of holding a license on your own.

Usage

Since Gaussian is a rather large and versatile program system with a range of different binaries, we would in general advice users to check whether their jobs are parallelized or not before submitting jobs. It would in general, unless every step is entirely well paralellized, allways be more efficient to split a complex many-step job into smaller paralell and serial parts/jobs so that also the overall utilization of hardware is improved. I you are in doubt whether or not your

job will scale outside one node (=shared memory), go to the Gaussian application home folder and check if there is an *.exel version of the executable(s) you will be using. If yes, your job will generally work ok in parallell up to approx 300 cores (this is for the more advanced users).

We generally wants users to run as many nodes as possible to limit the walltime length of running jobs.

Use

```
$ module avail gaussian
```

to see which versions of Gaussian are available. Use

```
$ module load Gaussian/<version> # i.e 09.d01
```

to get access to any given version of Gaussian.

The first time you run a Gaussian job?

Get the information you need here:

About the Gaussian version(s) installed on Stallo

Since the installs we have made on Stallo are somewhat different from installs of Gaussian elsewhere, we have gathered some information about this here:

Here we also adress issues related to running Gaussian in parallel, number of cores to use, memory allocation and special issues taken care of at install.

VASP (Vienna Ab initio Simulation Package)

VASP is a package for performing ab-initio quantum-mechanical molecular dynamics (MD) using pseudopotentials and a plane wave basis set.

To use VASP you need to be a member of the vasp group, contact the HPC staff to get access. Vasp is a commercial software that requires a license for all who wants to run on Stallo, to get access you would need to prove that the group you are a member of holds a valid licence.

You load the application by typing:

```
$ module load VASP
```

This command will give you the default version, which is currently 5.3.2

For more information on available versions, type:

```
$ module avail vasp
```

For more information, see: http://www.vasp.at

Amsterdam Density Functional program system

A description of the quantum chemistry and material science package ADF/BAND

General Information:

The ADF software is a DFT-only first-principles electronic structure calculations program system, and consists of a rich variety of packages.

Online info from vendor:

Homepage: http://www.scm.com Documentation: http://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 notur or scm) or trying the ADF mailing list (see http://www.scm.com/Support for more info).

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).

License information:

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.

is software for first-principles electronic structure calculations. The program suite shows good scaling behaviour, and we have tested in the 80-160 core region but not beyond. For statements from vendor related to scaling, please visit their home page as shown on top of this page.

(link to general info about license policy notur when available; currently contact: www.sigma2.no)

Usage

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 are available. Use

```
$ module load ADF
```

or

```
$ module load ADF/<version> # i.e 2014.10 (default)
```

to get access to ADF.

First run of ADF/BAND:

To get download the jobscript example(s), type the following:

Copy the contents of this folder to your home directory.

Then, submit the job by typing:

Compare the energy of adf_caffeine.out with the Bond Energy = -156.75317227 eV in adf_example_correct_stallo.output. The energy should ideally be identical or close to identical. When this is the case, you may alter variables in the shell script as much as you like to adapt to your own jobs. Good luck.

- 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, please make sure that the default openmpi is swapped with the default intel-MPI module.
- On Stallo, we holds the nbo6 plug in license that allows users of adf to produce files that can be read with the nb06 software.

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
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
```

ADF runscrip example:

```
#!/bin/bash -l
#################### ADF Job Batch Script Example ####################
# Section for defining queue-system variables:
#------------------------------------
# This script asks for a given set of cores. Stallo has got 16 or 20 cores/node,
# asking for something that adds up to both is our general recommodation (80, 160 etc)
# Runtime for this job is 59 minutes; syntax is hh:mm:ss.
# Memory not set since you are using full node,
# it can be specified pr core, virtual or total pr job (be carefull).
#------------------------------------
# SLURM-section
#SBATCH --job-name=adf_runex
#SBATCH --ntasks=40
#SBATCH --time=00:59:00
#SBATCH --mem-per-cpu=1GB
#SBATCH --output=adf_runex.log
#SBATCH --mail-type=ALL
#SBATCH --exclusive
#SBATCH --partition=multinode
#####################################
# Section for defining job variables and settings:
#------------------------------------
# Area for defining variables:

input=caffeine # Name of input without extention
```

```
extention=adf # We use the same naming scheme as the software default
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 load notur
module load ADF/2014.10

# Check other available versions with "module avail adf"

#------------------------------

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

mkdir -p $SCM_TMPDIR

# Preparing and moving inputfiles to tmp:

cd $SUBMITDIR
cp $input.adf $SCM_TMPDIR
cd $SCM_TMPDIR

# In case necessary, set SCM_IOBUFFERSIZE
#export SCM_IOBUFFERSIZE=1024

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

# Running the program:

time adf  -n $cores < $input.adf > adf_$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 adf_$input.out $SUBMITDIR/adf_$input.out
cp TAPE21 $SUBMITDIR/$input.t21

mkdir $USERWORK/$input.${BATCHNUM}.res
mv $SCM_TMPDIR/* $USERWORK/$input.${BATCHNUM}.res

# 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`

# ALLWAYS clean up after yourself. Please do uncomment the following line
#cd $SUBMITDIR
#rm  $SCM_TMPDIR/*
#rmdir $SCM_TMPDIR
```

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

Schrödinger Product Suites

A description of the Schrödinger product suites that are available for norwegian academic users

General Information:

NOTUR is, together with the user community, funding a national license of Schrödinger´s Small Molecule Drug Discovery Suite and PyMol.

Online info from vendor:

- Homepage: http://www.schrodinger.com/

About the suites:

- Small Molecule Drug Discovery Suite: http://www.schrodinger.com/smdd/
- PyMol: http://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.

Citation

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

License information:

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-uit(at)notur.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.

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:

One important note: All new installs will be made through the new software scheme on Stallo. Please confer the documentation for *New Scheme Modules*.

Use

```
$ module avail Schrodinger
```

to see which versions of Schrodinger are available.

Use

```
$ module load Schrodinger/<version> # i.e 2015.4
```

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 run_schrodinger.

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 soles:

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.

CHAPTER 33

Storage and backup

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)

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 default size of the home directory's for each user is 300 GB. If more space is needed for permanent storage users have to apply for it. Please contact the system administrators, support-uit@notur.no, for more information about this.

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.

Work/scratch areas

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-uit@notur.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.

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-uit@notur.no

Archiving data

Archiving is not provided. However you may apply for archive space on Norstore.

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.

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
```

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.

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.

Compression of data

Infrequently accessed files must be compressed to reduce file system usage.

Tools like gzip, bzip2 and zip are in the PATH and are available on all nodes. The manual page for these tools are very detailed, use them for further help:

```
$ man gzip
```

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 34

Transferring files to/from Stallo

All Notur systems are stand-alone systems, and in general we do not (NFS-)mount remote disks on them. Therefore you must either explicitly transfer any files you wish to use by using either sftp or scp, or you can mount your home directory('s) on the Notur systems *on your own computer*.

Transferring data to/from the system

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
```

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.

High-performance tools

OpenSSH with HPN

The default *ssh* client and server on stallo login nodes is the *openssh* package with applied HPN patches. By using a hpnssh client on the other end of the data transfer throughput will be increased.

To use this feature you must have a HPN patched openssh version. You can check if your ssh client has HPN patches by issuing:

```
ssh -V
```

if the output contains the word "hpn" followed by version and release then you can make use of the high performance features.

Transfer can then be speed up either by disabling data encryption, AFTER you have been authenticated or logged into the remote host (NONE cipher), or by spreading the encryption load over multiple threads (using MT-AES-CTR cipher).

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.
```

Subversion and rsync

The tools subversion and rsync is also available for transferring files.

CHAPTER 35

Lustre FS performance tips

How you can adjust Lustre depending on your application IO requirements.

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.

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.

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).

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).

storage/files/stallo-parallelIO-write.png

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.

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 Best Practices from NICS (http://www.nics.tennessee.edu/I-O-Best-Practices).

CHAPTER 36

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.

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.

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.

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.

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 37

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 applications section for an introduction on how to use modules.

CHAPTER 38

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.

Allinea Performance Reports

Allinea 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

```

```
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.

Performance tuning by Compiler flags

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.

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.

Vtune

Intel Vtune Amplifier is a versatile serial and parallel profiler, with features such as stack sampling, thread profiling and hardware event sampling.

Totalview

Totalview TotalView is a source- and machine-level debugger for multi-process, multi-threaded programs.

CHAPTER 39

Debugging

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.

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