Using Seastar

A manual for using HPC-systems especially targetted to the beowulf cluster "Seastar" of the CMT research group of the University of Antwerp

Nikolas Garofil

February 26, 2016

On a sunny afternoon at the UA,
I was threatened: "Explain Seastar !!! Okay ?!?"
So I started writing documentation,
with everything but it's location,
'cause that is TOP-secret, so I'll never say !

1 Disclaimer

This manual is distributed in the hope that it will be useful, but *without any warranty*, without even the implied warranty of not setting the printer on fire by printing it out, destroying your computer by reading it with a pdf-reader, ...

This is still an alpha version of the manual, most likely it will contain a lot of grammatical and spelling errors. There might also be errors in the code fragments which could cause serious damage.

!!! Do not run anything that you do not fully understand !!!

2 Some details about Seastar

It's probably best to skip this section if you've never used Seastar before, this is more of a technical reference...

Seastar is a Beowulf cluster[1] with the following specifications:

- At the moment it has **25 nodes** that can be used for calculating.
- Nodes are named **beo-*XX*** where ***XX*** is an integer from 16 to 82.
- **beo-00**→**beo-16**: old nodes, not sufficiently powerful anymore to be of use (or defective and not sufficiently cost-effective to be repaired).
- **beo-17**→**beo-28**: older nodes but still powerful enough to use
- **beo-29**→**beo-38**: new nodes
- **beo-39**→**beo-79**: reserved names

[1] See http://en.wikipedia.org/wiki/Beowulf_cluster for more information

- **beo-80→beo-82**: GPU[2]-nodes. Their CPU's[3] are not particularly powerful but their GPU's easily compensate for this
- There are also a couple of "special nodes":
  - **seastar-64**:
    - Login node, the only node in the cluster that you can reach directly from your PC/laptop
    - Its FQDN[4] is **seastar-64.cmi.uantwerpen.be** and its IPv4-address is **143.129.131.159**.
    - also has the name **opt-dev**, but this name can only be used from within the cluster.
    - Use this node to transfer files and submit jobs but do not run code on this node.
  - **seastar**: Master-node, of no concern to you. I'm only mentioning it because you could accidently notice its existence and run into confusing problems because of the name.
  - **sn-1**: Helps the masternode to handle the filesystem, this node is also of no concern to you.
  - **seamouse**: This is actually not a node but a completely separate system that handles all servertasks for CMT. It's listed because for a couple of its tasks it has a pretty tight connection with Seastar.
- Every node is connected with 2 networks:
  - The Ethernet[5] network
    - Speed: Fast (Gigabit cards and switches with CAT-6 cabling)
    - Network interface on the nodes: **eth0**
    - IPv4-address **seastar-64**: **192.168.52.250**
    - IPv4-address **beo-*X***: **192.168.52.*X***
    - Netmask: **/24**
  - The InfiniBand[6] network
    - Speed: Super-fast (IB DDR as "weakest" link)
    - Network interface on the nodes: **ib0**
    - IPv4-address **seastar-64**: N/A[7]
    - IPv4-address **beo-*X***: **192.168.51.*X***
    - Netmask: **/24**
- The operating system is a highly modified version of Ubuntu[8] but still has all properties of a regular GNU/Linux distribution.

[2] See http://en.wikipedia.org/wiki/Graphics_processing_unit for more information
[3] See http://en.wikipedia.org/wiki/Central_processing_unit for more information
[4] See http://en.wikipedia.org/wiki/Fully_qualified_domain_name for more information
[5] See http://en.wikipedia.org/wiki/Ethernet for more information
[6] See http://en.wikipedia.org/wiki/InfiniBand for more information
[7] Seastar-64 is not connected to the InfiniBand-network and never will be, only nodes that actually calculate should have InfiniBand.
[8] See http://en.wikipedia.org/wiki/Ubuntu_(operating_system) for more information

- To manage all available resources the cluster uses TORQUE[9] . It combines this with Maui[10] which makes sure the scheduling of jobs works fine.
- The filesystem of the whole cluster is Lustre[11] with a ZFS[12]-backend. It's mounted on **/l** and there is also a symlink **/data** that points to **/l** . Use this filesystem as permanent storage.
- **/l/home** contains all homedirectories of regular users and also a directory **/l/home/cdsrv1** filled with software you can use on the cluster or your desktop system.
- Although every node also has a lot of other dirs that you shouldn't worry about, there is one dir that I do want you to know about: **/scratch** is a filesystem that's very useful to store temporary results of calculations. It differs from **/l** in the following things:

  - Sharing: **/l** is shared by all nodes on the cluster, which causes it to look the same on the whole cluster. Although you can also see **/scratch** on the whole cluster, every node has a different version of this filesystem, so files saved in **/scratch** on node A are not visible on node B.
  - Speed: **/l** is a filesystem on harddisks shared on a network which makes it relatively slow. **/scratch** is a filesystem in memory not shared with anything so it's extremely fast.
  - Size: This depends from node to node but it's safe to say that **/l** is approximately 1000x the size of **/scratch** on a random node. But this certainly does not mean that **/scratch** is too small to be useful. It just means **/l** is humongous.

  The properties above make **/scratch** very useful to save intermediate results of jobs before saving the final results on **/l**.
- Buildenvironment: Everything you need to write and build software should be available, we have interpreters for fortran (`ifort` and `gfort`), `python`, `php`, `perl`, ... and compilers for C and languages similar to C (`gcc`, `icc`, `mpicc`, `nvcc`). If you don't know what compiler/interpreter to use for the languages that have multiple options, I would recommend using those from gnu (`gcc` and `gfort`).

3 Getting started

3.1 Account-creation

In order to get started you'll need an account, so contact me and I'll create it for you. Most of the account names on seastar start with the first character of

[9] See http://en.wikipedia.org/wiki/TORQUE for more information
[10] See http://en.wikipedia.org/wiki/Maui_Cluster_Scheduler for more information
[11] See http://en.wikipedia.org/wiki/Lustre_(file_system) for more information
[12] See http://en.wikipedia.org/wiki/ZFS for more information

the first name of the user followed by the last name (e.g. someone named Kurt Cobain would receive the username "kcobain").

I'll also ask you to choose a password. I might forget to tell you but make sure that this is a strong password. You can do this by following the usual rules that most sites demand (9 chars or longer, mix in special chars, don't base it on recognizable words, ...) but if you don't mind typing in long passwords then I would recommend XKCD-style[13] passwords. Whatever password you choose, make sure you don't use it anywhere else.

3.2 Logging in

In this document I'm assuming that you are using a POSIX-compliant[14] operating system. To be honest I have to admit that it's also possible to connect to the cluster from other operating systems, so why am I doing this ?

- Seastar itself is running a POSIX-compliant OS (GNU/Linux). It's easier to use if you already have some experience from using similar operating systems on your PC(s)
- Other operating systems, especially Microsoft Windows, are known to cause strange and hard to diagnose bugs...

I also have to assume that you already have some basic experience with GNU/Linux[15]. Otherwise I would have to explain every tiniest step and this document would become unreadably long.

So now your account is created. You should now be able to login if you start an SSH[16]-client and connect to server **seastar-64.cmi.uantwerpen.be**[17] with your username and password and use the TCP-port[18] that's registered for ssh. For our example user "Kurt Cobain" this would work like this:

1. Start a shell: You can do this by looking in the menu for a program named "terminal", "console" or something very similar. Starting this should open a (probably black) window with a prompt and the possibility to enter commands behind this prompt. The prompt differs a bit from system to system but it will probably look similar to this:
   ```
   kurt@nirvana ~ $
   ```
   `kurt` is here your username on your pc and `nirvana` the hostname of your pc.

2. Login on Seastar: This is really easy, use the command below, enter a password when requested[19] and you will be connected. (The password

[13] See http://imgs.xkcd.com/comics/password_strength.png for more info

[14] See http://en.wikipedia.org/wiki/POSIX for more information. The best known examples of POSIX-compliant operating systems are Mac OS X and the GNU/Linux distributions

[15] If not, every year the admins of CALCUA are holding a "Introduction to Linux" course that I can strongly recommend, see here for more info: https://www.uantwerpen.be/en/research-and-innovation/research-at-uantwerp/core-facilities/core-facilities/calcua/training/intro-linux/

[16] See http://en.wikipedia.org/wiki/Secure_Shell for more information

[17] Depending on how you are connecting you can skip the **.cmi.uantwerpen.be**-part. For the sake of brievity I'm skipping it in the rest of the document. If you are not sure if you can skip it: keep it.

[18] You will not need to enter this in most ssh-clients, but in case you do: it's **22**

[19] You can also use ssh-keys to login. I would recommend using keys but I'm not obligating you. More on ssh-keys will follow in section 7

won't be visible on the screen, this is normal behaviour)

```
kurt@nirvana ~ $ ssh kcobain@seastar-64
```

The screen will now show some uninteresting information about the cluster and on the last line you will have a prompt that looks like this:

```
0 kcobain@seastar-64 .../ $
```

In case you are wondering what the number at the front is, it's the return-code[20] of the last command you executed. When you want to logout, just type the command **exit** and you'll be disconnected.

3. Submit a script: This step will be handled in detail later, but for now I'll tell you the basic steps:

   - Create a shellscript that runs the command(s) you want to run: We'll let Kurt create `nevermind.sh`
   - Submit the script and you'll see it'll get a Job ID:

     ```
     0 kcobain@seastar-64 .../ $ qsub nevermind.sh
     314159.beosrv-c
     ```

   - When the job is finished, you'll probably want to read files containing the standard output (**STDOUT**) and the standard error output (**STDERR**)[21] from the job:

     ```
     0 kcobain@seastar-64 .../ $ cat nevermind.sh.o314159
     Here we are now, entertain us...
     0 kcobain@seastar-64 .../ $ cat nevermind.sh.e314159
     Oops, there was a suicide
     ```

4. Transferring files: There are multiple ways to transfer files between your PC and the cluster that use a GUI[22] but the 2 most popular methods use the shell:

   - `kurt@nirvana ~ $ scp seastar-64:/foo/somefile /bar`
     This will copy the file `somefile` in the directory `/foo` on seastar-64 to the directory `/bar` on your system. Scp has a lot of options to copy files, run **man scp** to see all the options.
   - If you want to copy a lot of files/directories at the same time, it's probably better to use **rsync**. A lot of files means that you will probably also want the options **-Havz**, these options make sure that all properties of the files remains the same in the destination, compression is used to speed it up and that everything is a bit more verbose so that you'll be able to follow the process. For example: the command below makes a copy of the whole `/foo` directory on seastar-64 to your system and names it `bar`

     ```
     kurt@nirvana ~ $ rsync -Havz seastar-64:/foo/ /bar/
     ```

     [23]

[20] See http://en.wikipedia.org/wiki/Return_code for more information
[21] See http://en.wikipedia.org/wiki/Standard_streams for more information
[22] See http://en.wikipedia.org/wiki/GUI for more information
[23] The `/` after `foo` and `bar` is NO mistake

4 Writing and submitting scripts

4.1 What qsub does

In section 3 I glanced over how you should submit scripts. I'll now look at how you should write scripts and give some more info about submitting them. The most important thing that you should know is: **qsub** submits the scripts and transforms them into jobs. All the rest are just details.

I even have some good news: If you don't want to do anything "fancy" and just want to run a simple program you don't even need a script. You can just use pipe the commands to **qsub**:

```
0 kcobain@seastar-64 .../ $ echo "sing 'Smells Like Teen Spirit'" | qsub
314160.beosrv-c
```

What's now happening is the following:

- **echo** sends the text **sing 'Smells Like Teen Spirit'** to **qsub**, a command part of the cluster's job scheduler.
- The job scheduler transforms this text into a command, the command **sing** with argument **'Smells Like Teen Spirit'**
- The job scheduler creates a new job that will execute this command later. The user didn't pass any options so the job has all the default options. As job id the next free number is given and this number (314160.BEOSRV-C) is shown by **qsub** to the user.
- The job scheduler places the job in a queue (this will be the default one because the user didn't specify anything). Most likely the cluster is running a lot of jobs at the moment so our new job has to wait for a while in the queue.
- Somewhere in the near future the job scheduler gets word from the resource manager that there are enough free resources available for our job and the job scheduler might than decide that the job can launch. Notice that jobs are not necessarily launched in the order that they are submitted ! The scheduler takes a lot of information into consideration to decide when to launch a job. In short it boils down to a combination of the following 2 rules:
  - The less resources a job needs, the quicker it starts.
  - Jobs that are already waiting for a long time will start quicker.
- So now that the scheduler decides that the job can launch, it asks the resource manager to reserve the necessary resources and launch the job.
- The resource manager lets the command run on a node but it redirects the STDOUT and STDERR streams to 2 files.
- When the job is finished those files are visible on your login node as **STDIN.o314160** (STDOUT) and **STDIN.e314160** (STDERR). In case you are wondering why the name is STDIN, it's because qsub received the command on it's standard input and because no name was given to the job. The number 31460 is the job id of the job.

Most likely you won't like the default options for the jobs so let's see how we can change them. I'll show you how to change one option now and I'll give you a list with the other possible options later:

```
0 kcobain@seastar-64 .../ $ echo "sing 'Come as you are'" | qsub -N singasong
314161.beosrv-c
```

This is pretty much the same job as the previous with one, but by using the **-N** option we gave it the name **singasong** and the 2 files that will be created after the job is finished will be **singasong.o314161** and **singasong.e314161**

4.2 Writing scripts

The previous way of creating jobs works, but usually it's better to write a script. When you write a script you don't have to pipe anything to **qsub**, instead you just run **qsub** (with the arguments you want) and behind the last argument you place the path to the script:

```
0 kcobain@seastar-64 .../ $ qsub -N singasong teenspirit.sh
314162.beosrv-c
```

As you can see, the scriptname ends with **.sh**, the reason for this is that the script is a shell script. You don't have to end your scripts with **.sh** but it's a convention that most people follow so I would recommend doing the same thing if you don't want to complicate things. Although the submit scripts don't have to be shell scripts (you can also write submit scripts in perl, python, ...) I would recommend using regular shell scripts written in (ba)sh[24]. Other scripts can cause strange problems[25].

Covering shell scripts completely will lead us to far. For now I'll have to be very brief about it:

- Make sure the first line in your script is always a shebang[26] (i.d. the 2 characters `#` and `!`) followed by the path of the shell-interpreter (e.g. `/bin/bash`, `/bin/sh`, ...). So you first line should look like this: `#!/bin/bash`
- All the following lines will be interpreted as if you typed them in on that specific shell

For example, this could be the submitscript of the last job (`teenspirit.sh`):

```
#!/bin/bash
echo Here we are now, entertain us...
```

The following is a example of a submitscript that contains some interesting features of bash:

[24] See http://en.wikipedia.org/wiki/Bash_%28Unix_shell%29 for more information

[25] Note that it's perfectly OK to start a perl, python, whatever... script from inside a shell-submitscript

[26] See http://en.wikipedia.org/wiki/Shebang_(Unix) for more information

```
#!/bin/bash
#Lines starting with # are comments and are ignored by bash
#Notice that ‘ instead of ’ is used on the lines with seq and mktemp
#The difference between these chars is important !
#The following command is the first that runs,
#bash looks for it in the current directory
./runsfirst
#The following command is searched for in the directory
#that is the PATH-variable of bash
runsnext
#The following command starts running in parallel with the
#rest of the script right after ’runsnext’ is finished
#Although the node sees it as a separate process, the cluster
#still sees it as 1 job together with the rest of the script
./runparallel &
#the following runs ’runalot’ 10 times
for i in ‘seq 1 10‘ ; do ./runalot ; done
#the following creates a temporary file and saves its
#path in the variable MYTEMPFILE
MYTEMPFILE=‘mktemp‘
#the next command’s output is sent to the temporary file
./sendtofile > $MYTEMPFILE
#the next command receives its input from the file
./receivefromfile < $MYTEMPFILE
#the following removes the tempfile
rm $MYTEMPFILE
#the following line checks if the tempfile is really deleted
if [ ! -e $MYTEMPFILE ] ; then echo temporary file is removed ; done
#the following command is the last that runs.
#it’s not located in the current directory
/l/home/kcobain/somewhere/runslast
```

Arguments that you pass to qsub can also be placed in the submitscript by adding lines in this format right behind `#!/bin/bash`:

```
#PBS -argument1 optionsforargument1
#PBS -argument2 optionsforargument2
```

So if we change `teenspirit.sh` like this:

```
#!/bin/bash
#PBS -N singasong
echo Here we are now, entertain us...
```

then we can submit it like this and it would still have the name “singasong”:

```
0 kcobain@seastar-64 .../ $ qsub teenspirit.sh
314163.beosrv-c
```

4.3 PBS-options

I'm only mentioning the options here that I think are actually useful for you. For a whole list, see `man qsub` .

Option	Arguments	Explanation
`-a`	`[[[[CC]YY]MM]DD]hhmm[.SS]`	Do not launch the job before this time. (e.g. `-a 2150` will make sure the job is not launched before 9:50pm today (or tomorrow if it's already after 21:50) and with `-a 11250345.07` it won't start before 03:45:07 on the 25th of november)
`-c`	`enabled`	Enables checkpointing of the job, the job will run a tiny bit slower (nanoseconds, not noticable) but it has the advantage that if for some reason the cluster needs to be rebooted the job can be saved first. It's always a good idea to use this option
`-I`		Makes job "interactive": when it starts running, the 3 standard streams will be connected to the shell where you ran qsub, this makes it look as if the job is running in that shell
`-j`	`oe` or `eo`	STDOUT and STDERR of the job will be joined together, with `oe` they both end up in STDOUT, with `eo` they land in STDERR
`-l`	resourcelist	A list of resources and their values separated by comma's that the job will need that are different from the default values. (e.g. `-l walltime=50,nodes=3` to request that the job can run on 3 nodes for 50 seconds[27]). See also the resources-table in section 4.3.1.
`-m`	`abe`	You will receive a mail when the job is aborted, when it begins or when it ends. (You're not obliged to use all 3 of these chars)
`-M`	`kurt.cobain@nirvana.com`	Mail-address used by `-m`. If you want multiple people to receive the mail, seperate the addresses with commas
`-N`	`name`	Gives the job a name (max. 15 chars long and the first char should be alphabetic).
`-q`	`node`, `queue` or `node@queue`	Tells the cluster where the job should be executed. The `node@queue` option is handy because a node can be part of multiple queues and in that case it has multiple versions of default resources
`-t`	`n`	The job will be submitted `n` times.[28]
`-V`		The job will receive all environment variables available in the shell where you ran qsub (seastar-64's bash). If you want to know what these vars are, run `set`
`-v`	`var1=value1[,var2=value2...]`	A list of environment variables and their values that the job will need
`-w`	`/some/where`	Set the default working directory to `/some/where`

`-W`	attributelist	A list of attributes and their values separated by commas that the job will need. (e.g. `-W depend=afterok:314164.beosrv-c` to request that this job gets scheduled when `314164.beosrv-c` finishes without any errors) Be careful if you start combining multiple attributes, you can get some strange effects! See also the attributes-table in section 4.3.2
`-X`		Enables X-forwarding[29]
`-z`		The job id will not be sent to STDOUT when you submit the job.

4.3.1 Resources

There are a lot of different resources that you can request, for a full list see http://docs.adaptivecomputing.com/torque/4-1-3/help.htm#topics/2-jobs/requestingRes.htm. If you are overwhelmed by the options, then just using the default instead of choosing options yourself is mostly a good choice. This is a list with the most popular options:

Option	Arguments	Explanation
`cput`	`[[HH:]MM;]SS`	Maximum amount of CPU-time used by all processes in the job
`walltime`	`[[HH:]MM;]SS`	Maximum amount of real time during which the job can be in the running state
`nodes`	*options*	Reserve a list of nodes, see the examples on the website mentioned above
`mem`	`size`[30]	Maximum amount of memory the job can use
`pmem`	`size`	Maximum amount of memory any single process can use
`pvmem`	`size`	Maximum amount of virtual memory any single process can use
`vmem`	`size`	Maximum amount of virtual memory

4.3.2 Attributes

Attributes are useful if you want to arrange the order of your own jobs, to see a full list run `man qsub` and scroll to the `-W additional_attributes`

[27] This does not mean that job will only stop after 50 seconds or that it will certainly run on 3 nodes. If could very well be that it finishes after 2 seconds and only needs 1 node if it's a tiny job. You should always make sure that your job requests the resources it will need. It's always safer to ask a bit more than you need.

[28] This option also accepts different arguments to tune the jobid (you probably don't need this), see the man page for more info

[29] If your job creates windows, with X-forwarding they will be drawn on your screen instead of that of the cluster

[30] Sizes need a suffix (e.g `123kb`). See also the url mentioned above for more info

option. But the most important attributes are the following

Option	Arguments	Explanation
`depend=afterok:`	`jobid`	This job can only be scheduled after the job `jobid` has terminated without errors
`depend=beforeok:`	`jobid`	If this job has terminated without errors, then job `jobid` can begin
`depend=afterany:`	`jobid`	The same thing as `afterok` but you can ignore errors
`depend=beforeany:`	`jobid`	The same thing as `beforeok` but you can ignore errors
`depend=afternotok:`	`jobid`	The same thing as `afterok` but replace "without" by "with"
`depend=beforenotok:`	`jobid`	The same thing as `beforeok` but replace "without" by "with"

5 Checking stuff

Now that you know everything that you should know to create and submit jobs, you'll probably also want to be able to monitor them. Seastar has multiple ways to follow them of which there are 3 that regular users should know about[31]:

- `qstat` to ask for a list of your jobs:

```
0 kcobain@seastar-64 .../ $ qstat -a -u kcobain
beosrv-c:
                                                                  Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
314165.beosrv-c         kcobain     infiniba ComeAsYouAre      27183     1      1    1gb 300:00:00 R 299:59:24
314166.beosrv-c         kcobain     infiniba Bleach            27184     1      1    2gb 300:00:00 R 299:49:24
314167.beosrv-c         kcobain     infiniba Nevermind         27185     1      1    3gb 300:00:00 R 299:39:24
```

  Most of the info shown is not really important, but what is important is the difference between "`Elap Time`" and "`Req'd Time`". This difference is how long your job can still run. Also important is the character below `S`, this has the following meaning:

  - `C`: Job is completed after having run
  - `E`: Job is exiting after having run
  - `H`: Job is held
  - `Q`: Job is queued, eligible to run
  - `R`: Job is running
  - `T`: Job is being moved to a new location
  - `W`: Job is waiting for its execution time
  - `S`: Job is suspended

- `showstart` to estimate when your job will launch:

[31] There is a lot of other software on seastar that is useful to monitor jobs, most will probably not be of any interest to you but in case you think I'm wrong: Take a look at http://docs.adaptivecomputing.com/maui/a.gcommandoverview.php

```
0 kcobain@seastar-64 .../ $ showstart 314168.beosrv-c

job 314168.beosrv-c requires 2 procs for 0:33:20

Estimated Rsv based start in                1:04:55 on Fri Jul 15 12:53:40
Estimated Rsv based completion in           2:44:55 on Fri Jul 15 14:33:40

Estimated Priority based start in           5:14:55 on Fri Jul 15 17:03:40
Estimated Priority based completion in      6:54:55 on Fri Jul 15 18:43:40

Estimated Historical based start in        00:00:00 on Fri Jul 15 11:48:45
Estimated Historical based completion in    1:40:00 on Fri Jul 15 13:28:45

Best Partition: fast
```

- You'll probably also want to known what queues exist, you can check them like this:

```
0 kcobain@seastar-64 .../ $ qmgr -c 'p s'
#
# Create queues and set their attributes.
#
#
# Create and define queue fast
#
create queue fast
set queue fast queue_type = Execution
set queue fast Priority = 10
set queue fast resources_max.nodect = 22
set queue fast resources_max.pmem = 1gb
set queue fast resources_max.walltime = 48:00:00
set queue fast resources_default.nodect = 1
set queue fast resources_default.nodes = 1
set queue fast resources_default.pmem = 500mb
set queue fast resources_default.walltime = 48:00:00
set queue fast resources_available.nodect = 26
set queue fast enabled = True
set queue fast started = True
#
# Create and define queue stress
...and so on...
```

  - Lines starting with `#` can be ignored
  - Every queue definition starts with `create queue` *somename*
  - The default queue properties are set with
    `set queue fast` *nameofthequeue somesetting* `= somevalue`
  - Queues can only be used if the definition contains the lines:
    `set queue` *nameofthequeue* `enabled = True`
    `set queue` *nameofthequeue* `started = True`

  This command does not show which queues contain which nodes, so here is a small list:

	fast	stress	Qgpu	uhimem	himem	dque	infiniband	interactive
beo-01	X	X			X		X	
beo-02	X	X			X		X	
beo-03	X	X			X		X	
beo-04	X	X			X		X	
beo-16	X	X			X			X
beo-17	X	X			X			X
beo-18	X	X			X			X
beo-19	X	X			X			X
beo-20	X	X			X			X
beo-21	X	X			X			X
beo-22	X	X			X			X
beo-23	X	X			X			X
beo-24	X	X			X			X
beo-25	X	X			X			X
beo-26	X	X			X			X
beo-27	X	X			X			X
beo-28	X	X		X				X
beo-29	X	X					X	
beo-30	X	X					X	
beo-31	X	X					X	
beo-32	X	X					X	
beo-33	X	X					X	X
beo-34	X	X					X	
beo-35	X	X					X	
beo-36	X	X			X		X	
beo-37	X	X			X		X	
beo-38	X	X			X		X	
beo-80			X					
beo-81			X					
beo-82			X					

- You can also use programs like `showq`, `showres`, `checkjob`, `mdiag`, `showstate`, `showstats` and so on to view even more info but the most important info is retrievable from the commands shown above

6 Running interactive programs

If you are planning to run interactive programs, make sure you know about X-forwarding and `showstart`. The most interesting script here is `runmatlabXXX`. It works like this: You replace the `XXX` by a version number (most likely this will be 714 for 7.14). As first argument you give the amount of hours (as integer) you want matlab to run, all the other arguments are passed on to matlab itself. So Kurt Cobain would run something like this if he wanted to run `matlab -desktop` for almost[32] 3 hours:

[32] "almost" because you should always reserve more time than you need

```
0 kcobain@seastar-64 .../  $ runmatlab714 3 -desktop
```

7 Differences with the VSC-systems

Many of you will also have an account on the VSC-system's (also known as Calcua or Turing and Hopper), these are the most important differences:

- Seastar is pretty small compared to Turing and Hopper, this might lead you to think that your jobs will run faster on those clusters BUT remember that you have to share that hardware with a LOT of other people. I would recommend to use what's available when it's available.
- Turing and Hopper are not tuned to physics. Not all of the software that you are using on Seastar is already available on the Calcua-systems. If you need anything that is not yet available, contact hpc@uantwerpen.be
- On the calcua system's, you'll have to use the `module` commands to decorate the environment where you are going to calculate.
- To login on the calcua systems you will need ssh-keys. To login on seastar ssh-keys are optional but not obligated (I'm strongly recommending them). As promised a while ago I was going to explain how to generate ssh-keys, it's actually really easy:
  - Run `ssh-keygen` . This will create the following 2 files in your homedir that form your key: `.ssh/id_rsa` (the private part, nobody except you needs this) and `.ssh/id_rsa.pub` (the public part, this should land on every system where you want to use it). During this procedure a passphrase will be asked, this is a password that protects your key so that you are the only one that can use it. I would recommend setting a passphrase but if you don't want to use a passphrase, just press enter when it's requested.
  - To get the public key on seastar run `ssh-copy-id -i kcobain@seastar-64` (obviously replacing `kcobain` with your username)
  - To get the public key on the VSC-systems, contact hpc@uantwerpen.be
  - If it worked you should now be able to `ssh` to the systems that have your public key without having to enter a password. If you created a key with passphrase, the passphrase will be asked.

Obviously there are also a lot of smaller differences hidden away. If you are planning to work with the VSC-system I would recommend that you start here: https://www.uantwerpen.be/en/research-and-innovation/research-at-uantwerp/core-facilities/core-facilities/calcua/support/

8 Seamouse's services

Seamouse does a lot of things of which the following might be interesting for you:

- It runs a printserver for the 4 CMT-printers and the big printer on the 4th floor. You use it like this:
  - Surf to http://seamouse:49631/printers/
  - Copy the link to the printer that you want to add (e.g. http://seamouse:49631/printers/U305BW)
  - Start installing a new printer but make you sure you choose "add by url" and use the link you copied
- It runs a webserver, everything that's placed on `seastar-64:/l/home/public_html/` will be hosted by seamouse (e.g. `/l/home/public_html/kcobain/mymusic` will be visible as `http://cmt.uantwerpen.be/kcobain/mymusic`). Note that the it can take up to 1 hour before the data is available online ! The webserver is updated every hour at ±5 minutes after the hour[33].

9 Questions and "fake" bugs

... I am waiting for input from the reader here ...

You just reached the end of the documentation.
Contact me for even more information,
Also, report what's unclear,
I will fix it, don't fear!
And when bored: Feel free to write a translation !

[33] This might change to a better system in the near future