

# Information on Using the TU Cluster

## Contents

### Contents

1. Information on Using the TU Cluster
  1. The New Cluster
  2. SSH-Key Authentication
  3. Memory
  4. Multi CPU Processing
  5. The Basics
  6. Interactive Sessions with qlogin
  7. Submitting jobs with qsub
    1. Submitting simultaneous jobs
  8. Getting Statistics with `qstat`
  9. Shared Folders
  10. Getting Mails form the Cluster
  11. X Session
  12. screen
  13. virtualenvwrapper
  14. numpy and scipy
  15. tensorflow
  16. jupyter notebook
  17. Finding files faster

If you have any questions, write an email to ✉ [dominik.kuehne@tu-berlin.de](mailto:dominik.kuehne@tu-berlin.de)

## The New Cluster

- Documentation: <http://gridengine.eu/mangridengine/manuals.html>
- New scheduler: Univa Grid Engine
- New master node: [cluster.ml.tu-berlin.de](http://cluster.ml.tu-berlin.de)
- Issue tracker: <https://git.tu-berlin.de/dkuehne/cluster/-/issues>
- UGE is compatible to GridEngine
- GPUs: To get one or more GPUs you have to type: `qlogin/qsub -l cuda=X`
  - Check free GPU slots: `qstat -F cuda`
  - Accessing a specific GPU: `qlogin/qsub -l cuda=X -l gputype=P100G12.`
  - GPU Types:

P100G12	Tesla P100 with 12GB
P100G16	Tesla P100 with 16GB
GTX	GTX 2080
V100	V100 with 32GB
TITAN	RTX TITAN
RTX6000	Quadro RTX 6000

## SSH-Key Authentication

Please use ssh key authentication. Password Authentication is not available on the cluster.

How to create a key:

- run `ssh-keygen ($ ssh-keygen)`
- copy the public key to `authorized_keys ($ cp ~/.ssh/id_rsa.pub .ssh/authorized_keys)`

## Memory

- By default only 4 GB per Job is allowed

- to increase use `h_vmem` and `m_mem_free`
- example to get a `qlogin` session with 16GB RAM: `qlogin -l h_vmem=16G -l mem_free=16G`

## Multi CPU Processing

- per default your jobs get one physical CPU (2 with Hyperthreading)
- you can request more with: `qlogin -binding linear:3` (for three physical CPUs)


## The Basics

- To get an account, write an email to ✉ dominik.kuehne@tu-berlin.de
- To access the cluster: `ssh cluster.ml.tu-berlin.de`
- You will end up on a machine called `vnc` which is *only a gateway*. You can start your screen or vnc session there, but you shouldn't do any computations on that machine. For that reason, development tools or matlab are not installed on `vnc`
- To get an interactive login on the cluster: `qlogin`. If you get a time out see below.
- To submit jobs: `qsub job.sh` The job has to be a shell script.
- The jobs are prioritized to achieve a fair use of all resources. Your past usage is taken into account, however with a "half-time" of currently one day. It is possible to define projects and allocate a larger share to those projects. If you have an urgent need for computing resources, contact ✉ dominik.kuehne@tu-berlin.de

Here is a list of the basic commands:

- `qlogin`: interactive session on the cluster
  - `-l RESOURCENAME=Property` : request a node with this property. For example `-l cuda=1` for a node with a GPU or `-l h=node17` for a node with name node17
- `qsub`: submit a batch job
- `qdel`: delete a job, i.e. a `qlogin` session
- `qmod`: modify a job (for example, to suspend it)
- `qstat`: see your current logins
  - `-f` : show list of all nodes
  - `-F` : list all available resources for each node
  - `-F vl -r -s r` : show which resources you currently request
- `qacct`: see how much resources have been used
  - `-o USERNAME -d 2` : for the last two days
- `qconf`: List configurations.
  - `-sql` : list all queues
  - `-sq queue-name` : show details for this queue
- `qhost`: List node properties
  - `-j` : detailed list of currently logged in User
- `qmon`: an interactive configuration tool. As a normal user you cannot change anything, but it might be interesting to look around.

Online documentation can be found here:

-  Man pages for the commands (link broken)

## Interactive Sessions with qlogin

`qlogin` gives you an interactive session on the cluster. Since each `qlogin` session uses up a slot on the cluster even if you don't perform any computations, you should try to have only one `qlogin` session active at any moment. If you need more shells, start them from the command line (`xterm`, `gnome-terminal`, or `kterminal`) within your `qlogin` session.

Originally, we had a provision that you could put interactive sessions on a "spare queue". This is now the default, so the old option `qlogin -l spare` does **not** work anymore.

One very **important thing** with using `qlogin` is that the number of available `qlogin` sessions is quite restricted. Therefore:

- Do not use multiple `qlogin` session just to have several shells. Instead, you should

- use `screen` to get several virtual text terminals.
- use a terminal like `gnome-terminal` or `kterminal` which provides different "tabs."
- Make sure that your `qlogin` sessions are properly terminated. In particular, **you should not simply close a window with a `qlogin` session in it**. This leaves the `qlogin` session in an unstable state where it is effectively ended, but still occupies a slot which is not freed anymore. Closing a `qlogin` session can be done by typing `exit`, or hitting `Ctrl-D` on an empty commandline.

## Submitting jobs with `qsub`

`qsub` submits a batch job to the cluster it is queued and depending on its priority scheduled on some node.

`qsub` has many many switches. Here are some of the most important ones, most of them also apply to `qlogin`

- `qsub` only accepts shell-scripts. If you have a binary, you have to wrap it in a script. All is not lost, though, because you can put standard options inside the script.
- It has to be a valid script: the first line needs to be `#!/bin/bash`
- By default, the job is started in your home directory. Use the `-cwd` switch to start it in the current directory
- The outputs are piped into files `<jobname>.o<jobid>` and `<jobname>.e<jobid>`. With the `-o` and `-e` switches, you can change these, or you can pass a directory where the outputs are then stored.
- You can specify the shell under which your script is run with `-S <shell>`. Usually you want to have `bash`
- You can specify resources with the `-l` switch. The example to log into a specific node use: `-l h=node02` where `h` is the `host` resource. For a list of all resources, `qconf -sc`
- IMPORTANT: The flags you specify have to be before the script. Example: `qsub -N "my funny script for the tesla gpu" -l -pgpu=1 funny.sh`

You can also specify such flags in your script by putting them on a line starting with `#$`, for example

```
#$ -l mem_free=500M
echo "Only runs on nodes which have more than 500 megabytes of free memory"
```

## Submitting simultaneous jobs

It is recommended *not* to queue thousands of jobs, e.g. in a loop from a script. It's better to use **job arrays**. Here is a small example, say `script.sh`:

Toggle line numbers

```
1 #!/usr/bin/env bash
2 #$ -binding linear:4 # request 4 cpus (8 with Hyperthreading) (some recommend 4 per GPU)
3 #$ -N ex           # set consistent base name for output and error file (allows for easy deletion
alias)
4 #$ -q all.q        # don't fill the qlogin queue (can some add why and when to use?)
5 #$ -cwd            # change working directory (to current)
6 #$ -V              # provide environment variables
7 #$ -t 1-100        # start 100 instances: from 1 to 100
8
9
10 # if you also want to request a GPU, add the following line to the above block:
11 #$ -l cuda=1       # request one GPU
12
13 echo "I am a job task with ID $SGE_TASK_ID."
14 python main.py     # here you perform any commands to start your program
15
```

Start the script with `qsub script.sh` and find the outputs in `script.sh.o<job id>.<task id>`. You can e.g. start Python scripts from `script.sh` and pass the task ID as an argument, or directly start Python from `qsub`.

## Getting Statistics with `qstat``

Besides submitting, one of the most important things is to get some statistics to see what is happening to your jobs, if they are stuck, or how much room is left on the cluster. `qstat` is the right tool for that. By default, it only lists your own jobs. It has many different options, here are a few:

- `-f` full format, shows information grouped by nodes

- `-u userid` lists specified users, use "\*" for all users
- `-q queueid` lists jobs only for some queue. We have the normal queue `all.q` and `spare.q` for interactive low-priority jobs
- `-g c` show summary for clusters
- `-ext` shows additional prioritization information. Have a look at the `tckts` column which shows you your relative priority with respect to the others.

## Shared Folders

The easiest way to share a folder `<directory>` with a user `<username>` are *access control lists*: `setfacl -R -m u:<username>:rwx <directory>`

You can also check the permissions: `getfacl <directory>`

## Getting Mails form the Cluster

- `-M` eMail Address
- `-m` what information do you want
  - 'b' Mail is sent at the beginning of the job.
  - 'e' Mail is sent at the end of the job.
  - 'a' Mail is sent when the job is aborted or rescheduled.
  - 's' Mail is sent when the job is suspended.
  - 'n' No mail is sent.
- Example: `qsub -m era -M foo@bar.org fnord.sh`

## X Session

Use X2Go instead of VNC:

- Client: <http://wiki.x2go.org/doku.php>
- Config:
  - Host: `cluster.ml.tu-berlin.de`
  - Login: `$User`
  - Session-Type (Sitzungsart): Custom -> `startxfce4`
  - Turn off Printer, File and Audio sharing.

## screen

You have to use screen from the vnc server and create a new tab for each node login you would like to use. Most important commands:

- `screen -S name` : creates a new screen session with name
- `screen -r name` : recovers session with name
- `screen -x name` : attach to session with name even if its attached elsewhere

During a session:

- `ctrl-a, c` : new tab
- `ctrl-a, n` : next tab
- `ctrl-a, k` : kill tab
- `ctrl-a, A` : rename tab
- `ctrl-a, "` : get the named list to select and enter directly
- `ctrl-a, ctrl-d` : dettach session (programm execution continues)

For more details see [screen reference](#)

To allow scrolling in a screen session add `termcapinfo xterm* ti@:te@`

to you `~/.screenrc` [link](#)

if you resize your terminal, screen may fuck up some view (`htop`, `man` etc). To fix this in the current window, execute `resize` or `dettach` and attach again.

## virtualenvwrapper

add `source /etc/bash_completion.d/virtualenvwrapper` to your `.bashrc` script.

`pip / virtualenv / virtualenvwrapper` are NOT installed on `vnc`, you have to `qlogin` to test if its working

`virtualenvwrapper` is not available on `node25`

## numpy and scipy

Numpy and scipy default to 1 thread. Don't increase the number of threads.

```
export MKL_NUM_THREADS=1
export OMP_NUM_THREADS=1
```

to your `.bashrc` file.

`OMP_NUM_THREADS` sets the threads for the default installation (e.g `pip install numpy scipy`) which uses `openblas`.

**MKL is usually faster** - to install it you have to compile numpy or scipy by yourself (TODO: failed on the cluster for me) or use `conda install numpy scipy` [Download Miniconda](#)

MKL will set the number of threads to the number of physical cores (thats normally half of the visible cores, because of hyperthreading). This is normally a very good setting, but if you have one node for yourself a slightly higher number may achieve a higher performance. To overwrite the default behavior use:

`MKL_DYNAMIC=FALSE MKL_NUM_THREADS=14 python myProgram.py` (will set to 14 threads) [Link](#)

## tensorflow

Every user has to install it's own version of tensorflow.

Using `pip`:

- login to one of the nodes: `qlogin`
- create an environment:
  - `virtualenv: virtualenv --system-site-packages -p python3 TARGETNAME`
  - or `mkvirtualenv TARGETNAME` if you use `virtualenvwrapper`
  - or `conda create --name TARGETNAME python=3.5` if you use `anaconda` (**recommended**)
- switch to the environment:
  - `source ~/TARGETNAME/bin/activate`
  - `workon TARGETNAME`
  - `source activate TARGETNAME`
- update pip: `easy_install -U pip`
- install *local* tensorflow with *cuda10* support:
 

```
pip install /home/local/tensorflow_pkg/tensorflow-1.12.0-cp36-cp36m-linux_x86_64.whl
```

 or
  - Note that this might be working only on certain nodes e.g. `node36`
- install latest tensorflow (once officially supported): `pip install --upgrade tensorflow` or `tensorflow-gpu`

`conda install tensorflow` does not work, but you can always use `pip` inside a `conda` environment.

Install from sources

- install bazel local: <https://docs.bazel.build/versions/master/install-ubuntu.html#3-run-the-installer>
- git clone <https://github.com/tensorflow/tensorflow> buildtf
- switch to branch you want to build: `git checkout r1.4`
- `./configure` inside buildtf
  - TODO: optimal flags here?
  - build with `cuda` if you want to use the GPU
- `bazel build --config=cuda --config=opt --config=mkl //tensorflow/tools/pip_package:build_pip_package`

- `bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/tensorflow_pkg`
- create/switch to your environment (like above)
  - `pip install /tmp/tensorflow_pkg/tensorflow-1.4.0-cp35-cp35m-linux_x86_64.whl`

To use tensorflow with your scripts you have to switch to your environment before using them (remember for qsub, but currently tensorflow does not work with qsub )

Building allows you to use MKL and extended CPU operations (AVX SSE etc) which may result in a speedup for some CPU operations. If you only use the GPU it is probably not necessary.

Building with MKL will download the newest version into `~/buildtf/third_party/mkl/mklml_lnx_2018.0.20170425/` but not probably add it to your lib path. If you start tensorflow and it is complaining about a missing `libmklml_intel.so` copy the library and add `export LD_LIBRARY_PATH=/home/YOURNAME/mkl/mklml_lnx_2018.0.20170425/lib/:$LD_LIBRARY_PATH` to your `.bashrc`.

The nodes have different compute capabilities and different cudnn versions installed. Therefore you may need to create different environments for some node:

- node02 & node03 :
  - reach via `qlogin -l pgpu=1`
  - GeForce GTX 1080 => compute capability 6.1
  - Cuda 8: `/usr/local/cuda-8.0`
  - Cudnn 5: `/usr/lib/x86_64-linux-gnu/libcudnn.so.5`
- node09 & node10 & node11 & node12:
  - reach via `qlogin -l gpu=1 (09 & 10)`
  - reach via `qlogin -l p100gpu=1 (11 & 12)`
  - Tesla P100-PCIE => compute capability 6.0
  - Cuda 8 and 9: `/usr/local/cuda-8.0` & `/usr/local/cuda-9.0`
  - Cudnn 6 and 7:
    - `/usr/lib/x86_64-linux-gnu/libcudnn.so.7`
    - `/usr/lib/x86_64-linux-gnu/libcudnn.so.6`
- node28:
  - reach via `qlogin -l gpu=1`
  - Tesla K40c => compute capability 3.5
  - Cuda 7.5 and 8: `/usr/local/cuda-7.5` & `/usr/local/cuda-8.0`
  - Cudnn 4,5 and 6:
    - `/usr/local/cuda-7.5/cuDNN/cuda/lib64/libcudnn.so.4`
    - `/usr/lib/x86_64-linux-gnu/libcudnn.so.5`
    - `/usr/local/cuda-8.0/targets/x86_64-linux/lib/libcudnn.so.6`

tensorflow 1.4 uses cudnn 6 as a default. Because node 2 and 3 doesn't provide this, you have to download it by yourself: <https://developer.nvidia.com/rdp/cudnn-download> Extracting and adding `/home/YOURNAME/cuda/lib64` to `LD_LIBRARY_PATH` should work with the pip install.

You can check the current GPU workload via `nvidia-smi`

If you do not need all GPUs available on one node, you should give tensorflow only the number of GPUs you need: `CUDA_VISIBLE_DEVICES=0,1 python3 myprog.py` [more details](#)

Tensorflow only allows to use 64GB of Host memory in combination with the GPU [link to github issue](#). To increase the value set `TF_CUDA_HOST_MEM_LIMIT_IN_MB=256000`

## jupyter notebook

How to access the notebook from your local computer:

- login into a node
- start Jupyter notebook: `/home/local/intelpython3/bin/jupyter notebook --ip 0.0.0.0 --port 8888` -> you have to choose a free Port
- Create an ssh tunnel from your local machine: `ssh -L 8888:$NODE:8888 $EMAIL@cluster.ml.tu-berlin.de`

- > Change \$NODE to the node-hostname egg: node03 or node20
- Launch the notebook on your local browser: <http://localhost:8888>

Setting the virtual environment of your notebook:

- use virtualenv: `virtualenv -p python3 TARGETNAME`
- activate by: `source TARGETNAME/bin/activate`
- setup your environment: `pip install jupyter numpy scipy matplotlib`
- get the kernel creation tool: `pip install ipykernel`
- create the kernel `python -m ipykernel install --user --name other-env --display-name "Python (other-env)"`
- select the corresponding kernel in the redirected notebook. (for redirection see previous section)
- get rid of unnecessary kernels by deleting respective folder of `jupyter kernelspec list`

## Finding files faster

On every weekend [robinhood](#) scan the filesystem, you can use it to find files:

- Disk Usage: `/opt/robinhood/bin/rbh-du -H $PATH`
- Find: `/opt/robinhood/bin/rbh-find`

IDA Wiki: IDA/TheTuCluster (last edited 2021-03-19 16:10:42 by DominikKuehne)