LME - WIKI

LEHRSTUHL FÜR MUSTERERKENNUNG
PATTERN RECOGNITION LAB

GPU-Cluster

*This cluster is **not intended for CPU-processes but for heavy GPU processes.***
The cluster software we use is called Slurm [http://slurm.schedmd.com/].

All specs for all Nvidia Cards [https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units]

If you want to give a student access to the cluster, please consider first to grant him access to the RRZE HPC Cluster, which also works with slurm and would free LME's resources. The process for HPC can be found here: https://cloud5.cs.fau.de/dokuwiki/doku.php?id=hpc [https://cloud5.cs.fau.de/dokuwiki/doku.php?id=hpc]. Also there is a new fancy cluster coming soon: https://hpc.fau.de/systems-services/systems-documentation-instructions/clusters/alex-cluster/ [https://hpc.fau.de/systems-services/systems-documentation-instructions/clusters/alex-cluster/]

Mailing List

*Before you start using the cluster, please **subscribe to the mailing list** cs5-cluster [https://lists.fau.de/cgi-bin/listinfo/cs5-cluster/].*
(If you run into problems, please let us know at `cs5-admin-cluster@lists.fau.de`).

Help

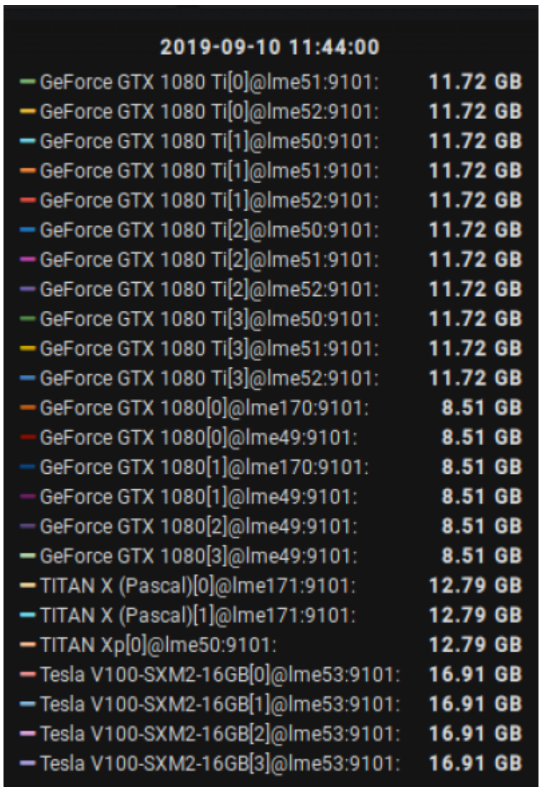
If you have problems using the cluster, we will help you, of course. :)
But before contacting the cluster admins, please check again if the answer to your question is not somewhere in this document.
Or can your advisor help you maybe?
If you're still stuck, please contact the cluster admins on `cs5-admin-cluster@lists.fau.de`.

Basic Concepts

On a cluster you don't normally work directly with the computers performing your computations (the compute nodes). Instead, you connect to a special node (the submit node), submit your job there, and the cluster software will schedule it for execution on one of the compute nodes. As soon as the scheduler has found a node with the resources required for your job (and you haven't exceeded the maximum number of active jobs allowed for your account), the job is executed there.

Our hardware

Nodes: lme49 lme50 lme51 lme52 lme53 lme170 lme171



```
              2019-09-10 11:44:00
GeForce GTX 1080 Ti[0]@lme51:9101:       11.72 GB
GeForce GTX 1080 Ti[0]@lme52:9101:       11.72 GB
GeForce GTX 1080 Ti[1]@lme50:9101:       11.72 GB
GeForce GTX 1080 Ti[1]@lme51:9101:       11.72 GB
GeForce GTX 1080 Ti[1]@lme52:9101:       11.72 GB
GeForce GTX 1080 Ti[2]@lme50:9101:       11.72 GB
GeForce GTX 1080 Ti[2]@lme51:9101:       11.72 GB
GeForce GTX 1080 Ti[2]@lme52:9101:       11.72 GB
GeForce GTX 1080 Ti[3]@lme50:9101:       11.72 GB
GeForce GTX 1080 Ti[3]@lme51:9101:       11.72 GB
GeForce GTX 1080 Ti[3]@lme52:9101:       11.72 GB
GeForce GTX 1080[0]@lme170:9101:          8.51 GB
GeForce GTX 1080[0]@lme49:9101:           8.51 GB
GeForce GTX 1080[1]@lme170:9101:          8.51 GB
GeForce GTX 1080[1]@lme49:9101:           8.51 GB
GeForce GTX 1080[2]@lme49:9101:           8.51 GB
GeForce GTX 1080[3]@lme49:9101:           8.51 GB
TITAN X (Pascal)[0]@lme171:9101:         12.79 GB
TITAN X (Pascal)[1]@lme171:9101:         12.79 GB
TITAN Xp[0]@lme50:9101:                  12.79 GB
Tesla V100-SXM2-16GB[0]@lme53:9101:      16.91 GB
Tesla V100-SXM2-16GB[1]@lme53:9101:      16.91 GB
Tesla V100-SXM2-16GB[2]@lme53:9101:      16.91 GB
Tesla V100-SXM2-16GB[3]@lme53:9101:      16.91 GB
```

- *lme221 has 4x Quadro RTX 6000 (24.2 GiB)*
- *lme222/lme223 have 4x Quadro RTX 5000 (16.1 GiB)*
- *lme170/lme171 have 2x Quadro RTX 8000 (48.6 GiB)*

Which node has which GPU?

```
 sinfo -h -o "%n %G"
```

```
lme221 gpu:q6000:4
```

```
lme51 gpu:gtx1080ti:4
lme52 gpu:gtx1080ti:4
lme170 gpu:q8000:2
lme171 gpu:q8000:2
lme222 gpu:q5000:4
lme223 gpu:q5000:4
lme49 gpu:gtx1080:4
lme50 gpu:titanxp:1,gpu:gtx1080ti:3
lme53 gpu:teslav100sxm216gb:4
```

You can use those identifiers to request a specific GPU. Like –gres=gpu:1 means I want one GPU. –gres=gpu:q5000:1 means that you want one from that type.

Data Sets

Please move large data sets to `/cluster/shared_data` so that everybody can use them.

Job Submission

You can submit jobs to the cluster from host `cluster.i5.informatik.uni-erlangen.de` *(*`lme242`*). Use SSH to connect to this machine.*

The most common way to run a job on the cluster is to submit a small shell script containing information about the job. Here is one example:

example.sh

```
#!/bin/bash
#SBATCH --job-name=MY_EXAMPLE_JOB
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --mem=12000
#SBATCH --gres=gpu:1
#SBATCH -o /home/%u/%x-%j-on-%N.out
#SBATCH -e /home/%u/%x-%j-on-%N.err
#SBATCH --mail-type=ALL
#Timelimit format: "hours:minutes:seconds" -- max is 24h
#SBATCH --time=24:00:00
#SBATCH --exclude=lme53
### Choose a specific GPU: #SBATCH --gres=gpu:q5000:1
### Run `sinfo -h -o "%n %G"` for GPU types

# Tell's pipenv to install the virtualenvs in the cluster folder
export WORKON_HOME==/cluster/`whoami`/.python_cache

echo "Your job is running on" $(hostname)

# Small Python packages can be installed in own home directory. Not recommended for big packages like tensorflow -> Follow instructions for pipenv below
# cluster_requirements.txt is a text file listing the required pip packages (one package per line)
pip3 install --user -r cluster_requirements.txt
python3 train.py
```

Make sure to put all #SBATCH commands at the top of the file. This would say that the job consists of only 1 parallel task which needs 2 CPUs and 12000 MiB RAM and 1 GPU, the job itself would be `th train.lua` *in this example. There is a number of other interesting options, please refer to the manpage of sbatch [https://slurm.schedmd.com/sbatch.html].*

To submit this you would ssh to `cluster.i5.informatik.uni-erlangen.de` *and run* `sbatch example.sh` *(if that's your file name).*

(You can also get an interactive shell with `srun --pty --nodelist=lme49 bash -i` *— but please avoid using this command.)*

But I need to run Tensorflow 1.XX or an old verion of Tensorflow

You should be able to run older versions of Tensorflow with CUDA 10 dependency when installing tensorflow via `conda`. You can install your own conda installation like so

```
# Use the miniconda installer for faster download / install of conda
# itself
wget http://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh \
    -O miniconda.sh
chmod +x miniconda.sh && ./miniconda.sh -b -p /cluster/$(whoami)/miniconda
```

To make miniconda's `conda` and `python` binary available you need to add `/cluster/$(whoami)/miniconda/bin`. E.g. `export PATH=/cluster/$(whoami)/miniconda/bin/:$PATH`

Chain Multiple Jobs

```
#!/bin/bash
TASKS="pre-processing.sl mpi.sl post-processing.sl"
DEPENDENCY=""
for TASK in $TASKS ; do
    JOB_CMD="sbatch"
    if [ -n "$DEPENDENCY" ] ; then
        JOB_CMD="$JOB_CMD --dependency afterok:$DEPENDENCY"
    fi
    JOB_CMD="$JOB_CMD $TASK"
    echo -n "Running command: $JOB_CMD  "
    OUT=`$JOB_CMD`
    echo "Result: $OUT"
    DEPENDENCY=`echo $OUT | awk '{print $4}'`
done
```

More info: https://github.com/HPCNow/hpcnow-labs/blob/master/user-training/05-setting-up-complex-workflows.md [https://github.com/HPCNow/hpcnow-labs/blob/master/user-training/05-setting-up-complex-workflows.md]

Miniconda

Miniconda is installed on cluster nodes at `/opt/miniconda`*(not lme242) on if you prefer installations using conda or you need Python 3.7.*

Just add `export PATH=/opt/miniconda/bin:$PATH` *to your job script. This will enable* `python`, `pip`, `conda`*. Be aware that you still have to use the* `–user` *option of pip to be able to install packages.*

Or install your own version of Miniconda:

```
wget http://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh \
    -O miniconda.sh
```

```
chmod +x miniconda.sh && ./miniconda.sh -b -p /cluster/$(whoami)/miniconda
```

Create your own Pipenv (to use your own package version)

Works like this (TODO: fix link) ~~https://asciinema.org/a/ME2qoAatFqIRph0bl3D8kBTHb [https://asciinema.org/a/ME2qoAatFqIRph0bl3D8kBTHb]~~

And then execute on cluster (TODO: fix link) ~~https://asciinema.org/a/IwL7TBoR1iTBhfwUdJMbcTpD2 [https://asciinema.org/a/IwL7TBoR1iTBhfwUdJMbcTpD2]~~ *(*`cd` *to the correct folder in your sbatch script to be on the safe side)*

You can also copy the created folder to your PC. Be careful, Tensorflow must be compatible with installed CUDA and cudnn version. E.g.

```
  pipenv install tensorflow-gpu==1.12
```

If your own project needs to be installed:

```
pipenv install --dev -e .
```

Or you have a requirements.txt:

```
pipenv install -r requirements.txt
```

But be aware that Python will never allow fully reproducible runs. It's here to surprise you everyday!

This will tell `pipenv` *to install the dependencies on our cluster drive*

```
export WORKON_HOME=/cluster/`whoami`/.python_cache
```

Debug your script

If something fails and you need to debug your script it's usually good not to request a GPU in your sbatch file (remove `#SBATCH –gres=gpu:1`*). Then you don't need to wait until a GPU is available. So you can immediately run your script. Add a time a time limit via* `#SBATCH –time=5` *for 5min maximum runtime.*

You can also launch an interactive session to see your script fail in real-time (with or without GPU). Please do not spend too much time in this mode:

```
    srun --pty --gres=gpu:1 bash -i
```

1. *Run your script the first times interactive with `srun –pty –gres=gpu:1 bash -i` then start your script from normal shell*
2. *Learn how to use pdb from command line: ``python3 -i your_script.py`` (will stop on exception)*
3. *Use logging: https://martinheinz.dev/blog/24 [https://martinheinz.dev/blog/24] , also log to stdout: https://stackoverflow.com/questions/13733552/logger-configuration-to-log-to-file-and-print-to-stdout#13733863 [https://stackoverflow.com/questions/13733552/logger-configuration-to-log-to-file-and-print-to-stdout#13733863]*

```
import argparse
from os.path import join, dirname
import logging
import os
import sys
import time



def write_config_file(args, output_path, use_toml=False):
    if use_toml:
        import toml as json # requires 'pip3 install toml'
    else:
        import json
    filename = os.path.join(output_path, 'args.toml' if use_toml else 'args.json')

    with open(filename, 'w') as json_file:

        json.dump({}, json_file)
        try:
            import git # require pygit
            repo = git.Repo(search_parent_directories=True)
            sha = repo.head.object.hexsha
            commit_message = repo.head.object.message
        except Exception:
            logging.warn(
                'Not executing within git repo! No commit message will be included in log')
            sha = '???'
            commit_message = '???'

        json.dump(
            {'args': sys.argv,
             'git revision':
                 {'message': commit_message, 'sha': sha},
             'arg.dict': args.__dict__
             }, json_file)
        logging.info('Writing file %s...' % filename)

def main():

    parser = argparse.ArgumentParser()
    parser.add_argument('--input-folder', default='.')
    parser.add_argument('--output-folder', default='.')
    parser.add_argument('--output-tag', default='unnamed')
    args = parser.parse_args()

    args.output_tag += time.strftime("_%d.%m.%y_%H.%M.%S")
    args.output_folder = os.path.join(os.path.expanduser(args.output_folder), args.output_tag)
    os.makedirs(args.output_folder, exist_ok=True)

    logging.basicConfig(
        filename=join(args.output_folder, os.path.basename(__file__)[:-3] + '.log'),
        level=logging.DEBUG,
        format= '[%(asctime)s] {%(pathname)s:%(lineno)d} %(levelname)s - %(message)s',
        datefmt='%H:%M:%S',
        force=True,
    )
    # Will log to stderr
    logging.getLogger().addHandler(logging.StreamHandler())
     # for information on experiment and reproduce with same parameters
     # just use: args = toml.load(...)
    write_config_file(args, args.output_folder, use_toml=True)

    my_awesome_script(args)
```

Data

The cluster has two main directories for storing intermediate data:

- `/cluster` *is a distributed file system available on all cluster nodes*
- `/scratch` *is a local directory for data used within a single job (faster)*

If you have to work with significant amounts of data, please don't use your home directory, this could impair performance for everybody else!

Important:

- *Put your data in a* ***sub-directory*** *with the same name as your user account (e.g.* `/cluster/gropp` *or* `/scratch/gropp`*).*
- *Don't forget to* ***delete your data*** *from the cluster directories when you're done.*
- *Note that there is* ***no backup*** *of the cluster storage.*
- *Don't use your home directory for processing big amounts of data!*
- *Don't store data you cannot afford to lose on the cluster file systems!*

The `/cluster` *file system is accessible from non-cluster (Linux) computers on* `/net/cluster`*, on Windows computers you can use WinSCP and access* `/cluster` *on* `cluster.i5.informatik.uni-erlangen.de`.

Output

By default both standard output (stdout) and standard error (stderr) are directed to a file of the name "slurm-%j.out", where the "%j" is replaced with the job allocation number.

It is possible to change the file names with the `#SBATCH -o` *and* `#SBATCH -e` *options (you can use the variable* `%j`*).*

⚠ *If you run into problems with you cluster jobs, first check these output files!*

Highscore

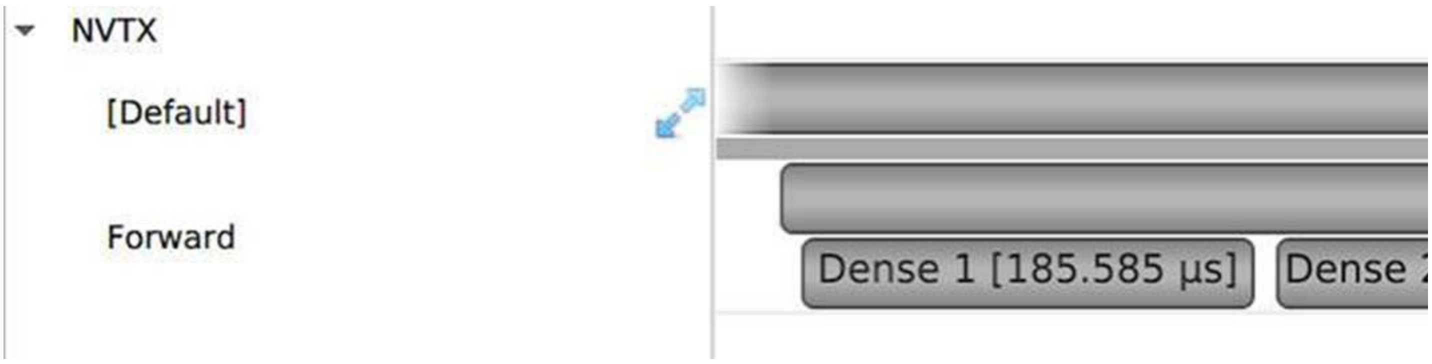
```
sreport user top topcount=10   -t hourper --tres=gres/gpu start=$(date --date="$(date +'%Y-%m-01')" +%D)
```

Why is my job slow? Profile it!

py-spy (https://github.com/benfred/py-spy [https://github.com/benfred/py-spy]) is installed on the cluster. So you can put this in your train.sh to generate a flamegraph (profile for 100s):

```
  py-spy --flame profile.svg --duration 100 -- python3 matmul.py
```

Even better is to profile your GPU utilization. Watch this video for that https://www.youtube.com/watch?v=SpZ5MYRQc0U [https://www.youtube.com/watch?v=SpZ5MYRQc0U]. ``nsys`` is installed on our nodes.



You can find their slides here: https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9339-profiling-deep-learning-networks.pdf [https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9339-profiling-deep-learning-networks.pdf]

```
nsys profile -t nvtx,cuda,cudnn,cublas --force-overwrite=true --stats=true --output=myapp python3 ./train.py
```

It's good to add some regions to see which part of your application takes most time.

- *Use it with Tensorflow: https://github.com/NVIDIA/nvtx-plugins [https://github.com/NVIDIA/nvtx-plugins]*
- *Use it with Torch:*

```
import torch.cuda.nvtx

with torch.cuda.nvtx.range("forward pass"):
      # do stuff here
      with torch.cuda.nvtx.range("encoder"):
           # do stuff here

      with torch.cuda.nvtx.range("decoder"):
           # do stuff here
```

Job Control

Use `squeue` *to show running jobs, or* `scancel` *to abort a job.*

- `scancel <jobid>`*: cancel one job*
- `scancel -u <user>`*: cancel all jobs from one user*
- `scancel -t PENDING -u <user>`*: cancel all pending jobs from one user*

This shows your priority:

```
squeue -o "%8i %8u %15a %.10r %.10L %.5D %.10Q"
```

Advanced Topics

eMail Notifications

Slurm can notify you when the status of a job changes. See notifications.

Python Virtual Environments

In case you'd like to have complete control over your python packages or need a different version of a library than what is installed, you can use virtualenvs.

Cluster Administration

→ *admin*

Time Limit and Job Dependencies

A time limit of 24h is enforced on the cluster. You can launch follow-up jobs with dependencies (they will only start if your first job successfully finished.

http://www.vrlab.umu.se/documentation/batchsystem/job-dependencies [http://www.vrlab.umu.se/documentation/batchsystem/job-dependencies]

PyTorch

PyTorch is installed for python3. If you need a different version please see the instructions for creating a virtualenv above.

I need software X on the cluster

If its an offical Ubuntu package there's no problem asking the admins to install it.

Otherwise install the software on /cluster/$(whoami)/opt or /cluster/$(whoami)/local. You may need to set PATH, CPATH, LD_LIBRARY_PATH, LD_LOAD_PATH accordingly. E.g.

```
        export LD_LOAD_PATH=/cluster/$(whoami)/local/lib:$LD_LOAD_PATH
        export LD_LIBRARY_PATH=/cluster/$(whoami)/local/lib:$LD_LIBRARY_PATH
        export CPATH=/cluster/$(whoami)/local/include:$CPATH
        export PATH=/cluster/$(whoami)/local/include:$PATH
```

Then you can simply set CMAKE_INSTALL_PREFIX to /cluster/$(whoami)/local and you will use your self compiled libraries.

Analyse GPU usage on a specific node

```
srun --pty --nodelist=lme50 /opt/cluster/bin/nvtop
```

References

- *man sbatch [https://slurm.schedmd.com/sbatch.html]*
- *man squeue [https://slurm.schedmd.com/squeue.html]*
- *man scancel [https://slurm.schedmd.com/scancel.html]*
- *man sinfo [https://slurm.schedmd.com/sinfo.html]*
- *man scontrol [https://slurm.schedmd.com/scontrol.html]*

serverservices/gpu-cluster.txt · Last modified: 2021/11/18 17:23 by hernandez