CL Wiki

Institute of Computational Linguistics – University of Zurich

rattle

Rattle is our server for model calculations on graphics processing units (GPUs). The machine is available for all members of the institue (CL and IFI) after registration (see below).

Registration / Access

Whoever wants to work on rattle has to send an email containing with her/his ifi account ID (normally `oscar` for oscar@cl.uzh.ch) to techno@ifi.uzh.ch [mailto:techno@ifi.uzh.ch] and a carbon copy (CC:) to laeubli@cl.uzh.ch [mailto:laeubli@cl.uzh.ch].

After activation, the server is accessible as usual by SSH with ifi username and passwort, for example:

```
ssh oscar@rattle.ifi.uzh.ch
```

Discussion forum

Whoever runs calculations on rattle must consult the respective Slack Channel regurarly. Announcements regarding performance bottlenecks, agreements on long time usage and other helpful information is published there. For registration and more visit https://rattle-uzh.slack.com/signup [https://rattle-uzh.slack.com/signup].

Home directory

Each user has a local home directory (e.g. `~/oscar`) at her/his disposal. Moreover, the machine has access to software on clfiles on `/mnt/storage/clfiles/resources`. For reproducible data, especially frequently changing data that uses much space, that needs to be exchanged between rattle and the rest of the CL infrastructure, you use one of the NFS partitions `/mnt/storage/scratch[012]` and remember that network partitions have slightly worse read and write performance than local partitions.

Local data in the home directories is saved on hourly basis by snapshots. The snapshots are available in `~/.snapshots`.

Disk consumption of the respective home directories can be looked up with the command `duc /home`. This data is refreshed once a week.

Quick guide for pew and working keras/tensorflow

Does not yet work properly with jupyter-notebooks, because of an inconsistency of sqlite3.

```
$ pew install 3.4.5
$ pew new dl34 --python=$(pythonz locate 3.4.5)
dl34$ pip install --ignore-installed  keras matplotlib
dl34$ pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.11.0rc0-cp34-cp34m-linux_x86_64.whl
dl34$ pip install --ignore-installed  ipython jupyter sqlite
```

Quick guide for anaconda3

Please adapt to the current versions…

download anaconda3:

```
$ wget https://repo.continuum.io/archive/Anaconda3-4.2.0-Linux-x86_64.sh
$ bash Anaconda3-4.2.0-Linux-x86_64.sh
# include in your bash path
export PATH="/home/user/oscar/anaconda3/bin:$PATH"
# install cudnn6.0
conda install -c anaconda cudnn=6.0
$ pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-1.4.1-cp35-cp35m-linux_x86_64.whl
$ pip install keras
```

Monitor GPU activities

```
$ watch nvidia-smi
```

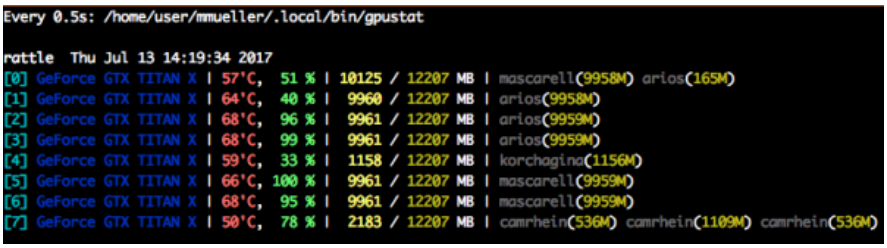
For monitoring in a more `htop` like manner use this:

```
alias gtop="watch --color -n 2 /mnt/storage/clfiles/projects/clresources/scripts/gpustat"
```

You can put the alias into your `.bashrc` or `.zshrc`. Now, run

```
$ gtop
```

and leave with Ctrl+C. Sample output:



CUDA and cuDNN versions

Currently (07.03.2018) only CUDA 8 is installed on rattle. You might also need a specific version of cuDNN (`libcudnn.so.6`) which is not currently installed. You can download this from NVIDIA (`cudnn-8.0-linux-x64-v6.0.tgz`, **not** a .deb) and put the `.so` with links into a local folder, e.g. `~/lib`. Then, make sure the environment variable `LD_LIBRARY_PATH=<your home>/lib/:$LD_LIBRARY_PATH` is exported.

Being nice on rattle

GPUs 0 to 7 are available for us. This means that resources are scarce. Before occupying GPUs for longer, please verify on slack if resources are available and if so, leave a message on which cores you are using and for how long you intend to use them.

GPUs 0, 1, and 7 are shared units, which means that several users can run processes on them, while 2 to 5 are exclusive and can only be used by one user at a time. So, if you're process is most likely not to use up all 12G of one GPU, consider sharing one GPU with others and thus use 0, 1 or 7.

In advance of an experiment, also think about how much memory you are going to need. It might well be that your process won't need a whole GPU, but that some space available on a shared GPU might suffice.

So, after having checked which GPUs are free or have unused resources, you will want to define which GPU you want to used in your script. Per default, libraries such as TensorFlow (or if used as a backend with Keras), are greedy and want to use all GPUs available, which will not work. By setting the right environment variable, you can choose the GPU you want to use. In Python, it would look like this:

```
import os

os.environ['CUDA_VISIBLE_DEVICES'] = '1'  # for one GPU
os.environ['CUDA_VISIBLE_DEVICES'] = '2,4'  # for several GPUs
```

Also, if you won't need all the processing power a GPU can offer you, you might want to use only a fraction of it. With TensorFlow and Keras, memory allocation could look like so:

```
import tensorflow as tf
from keras.backend.tensorflow_backend import set_session
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.visible_device_list = "0"
set_session(tf.Session(config=config))
```

This would mean you will use half a GPU for training your model.
