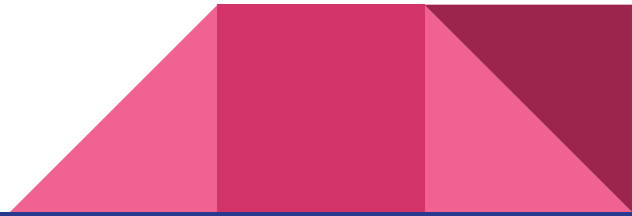
Using Jupyter with HPC backend

Michał Cyrkowski, Sławomir Gicala

Presentation plan

1. Introduction and main goal of the project
2. Creation HPC backend and connection
3. Using jupyter notebooks in SLURM environment
    a. Benefits
        i. Better scaling
        ii. Managing cluster with slurm magics
    b. Drawbacks
        i. Lower single core performance
4. Conclusions
5. References

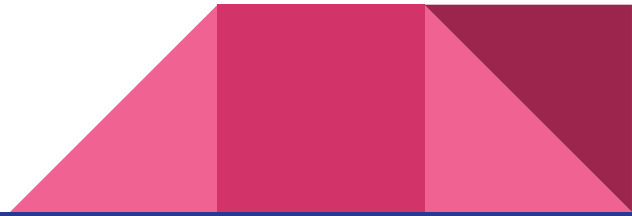


Introduction

Creation HPC backend and connection

Preconditions

Before running scripts it is necessary to create authorised_keys file in ~/.ssh and give it permission rights of 600. This enables automatic tunnel creation and as a result enables our notebook_job.sh to run properly.



start_notebook.sh

```
#!/bin/bash

# This script submits a Slurm job to get resources and
# start a Jupyter notebook on those resources.

module purge
#module load plgrid/tools/python/3.4.2
module load plgrid/tools/gcc/4.8.0
module load plgrid/tools/openmpi/1.6.5-gnu-4.8.0-ib
pip3 install --user --upgrade pip
pip3 install --user virtualenv
python3.4 -m venv env
source env/bin/activate
pip3 install --user jupyter

uid=$(id -u $(whoami))
jobid=$(sbatch --parsable notebook_job.sh --uid=$uid)
workdir=$(squeue --format=%Z --noheader -j $jobid)
logfile=$workdir/jupyter-$jobid.log
errfile=$workdir/jupyter-$jobid.err

echo -n "Waiting for notebook to start  "

# wait for the logfile to appear and be not-empty
sp="/-\|"
while [ ! -s $logfile ]
do
    printf "\b${sp:i++%${#sp}:1}"
    sleep 1;
done

. $logfile

token=""
sleep 5
token=$(tail -1 $errfile | awk 'BEGIN { FS="="} ; { print $2 }')

echo ""
echo "1) Tunnel to the cluster from your laptop: run this command on your laptop:"
echo "    $FWDSSH"

if [ -z $token ]
then
  echo "2) If you've set a notebook password, point your browser to:"
  echo "    http://localhost:8888"
  echo "(waited 5 seconds for token to appear. If you expected a token,"
  echo " you can find it in $errfile)"
else
  echo "2) and point your browser to:"
  echo "    http://localhost:$NOTEBOOK_PORT?token=$token"
fi
```

- Script disables loaded modules
- Loads modules needed for virtualenv and jupyter installations
- creates new virtualenv and activates it
- Runs notebook_job.sh as current user
- Awaits jupyter startup
- Displays informations needed for connection and shutting down jupyter core
- It should be run as ./start_notebook.sh

notebook_job.sh

```
#!/bin/bash

#SBATCH --job-name=jupyter_notebook_launch
#SBATCH --output=jupyter-%j.log
#SBATCH --error=jupyter-%j.err
#SBATCH --nodes 1
#SBATCH --ntasks-per-node 12
#SBATCH --time 0:50:00

# Load module with python and jupyter notebook

# Get available port for notebook and tunneling to your local machine
NOTEBOOK_PORT=$(python -c 'import socket; s=socket.socket(); s.bind(("", 0)); print(s.getsockname()[1]); s.close()');
TUNNEL_PORT=$(python -c 'import socket; s=socket.socket(); s.bind(("", 0)); print(s.getsockname()[1]); s.close()');
echo "NOTEBOOK_PORT='$NOTEBOOK_PORT'"

# Tunneling
ssh -R$TUNNEL_PORT:localhost:$NOTEBOOK_PORT ui.cyfronet.pl -N -f

echo "FWDSSH='ssh -L$NOTEBOOK_PORT:localhost:$TUNNEL_PORT $(whoami)@ui.cyfronet.pl -N'"

# Start the notebook
unset XDG_RUNTIME_DIR
jupyter-notebook --no-browser --no-mathjax --port=$NOTEBOOK_PORT
```

- It is a batch job of jupyter core
- Firstly it configures jupyter environment such as:
  - log files
  - Number of nodes
  - Number of workers
  - timeout
- Then it starts new tunnel
- Finally it starts jupyter core

Using jupyter notebooks in SLURM environment

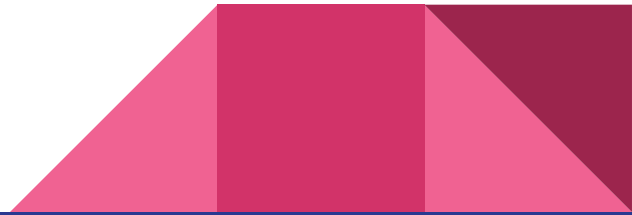
Benefits

1. Better scaling
2. Managing cluster with slurm magics

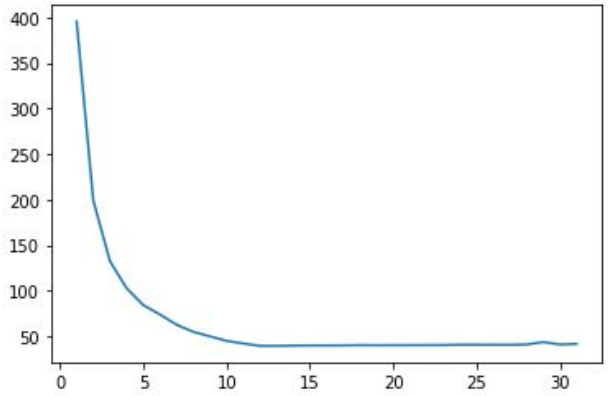
Better scaling

To test scaling we have decided to use model fitting in RandomForestClassifier that is a part of sklearn library. Then we run script on three machines:

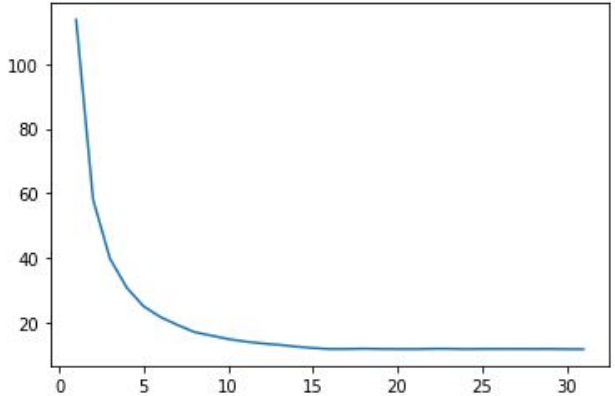
- Modern workstation: i7-10700K (8C/16T, overclocked to 5GHz)
- Modern office laptop: i5-113G7(4C/8T, 2.4GHz, boost up to 4.2GHz)
- PLGRID Zeus: 2x Intel Xeon E5645(6C/12T, 2.40 GHz, boost up to 2.67 GHz), but due to queuing we only requested 12 workers



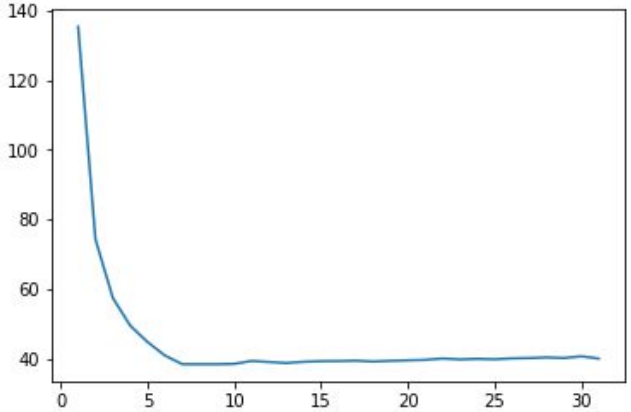
Execution time plots



Zeus

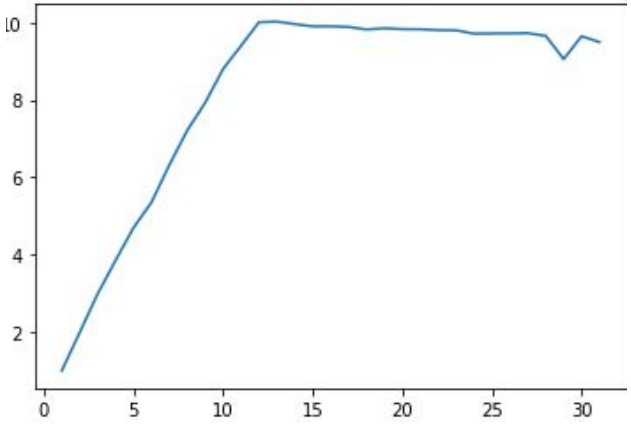


Workstation

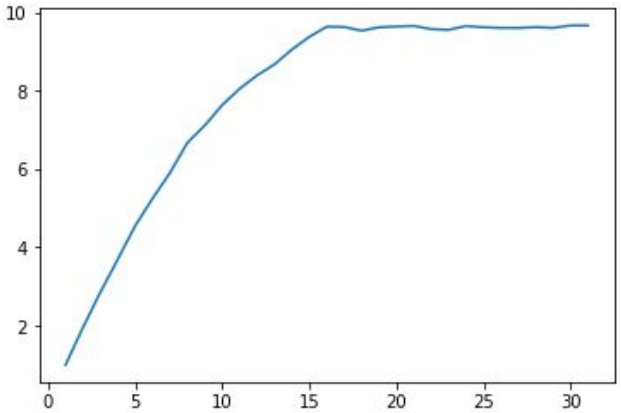


Laptop

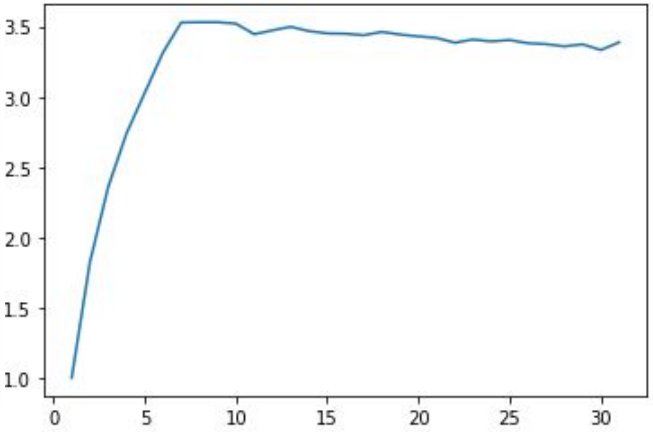
Acceleration plots



Zeus

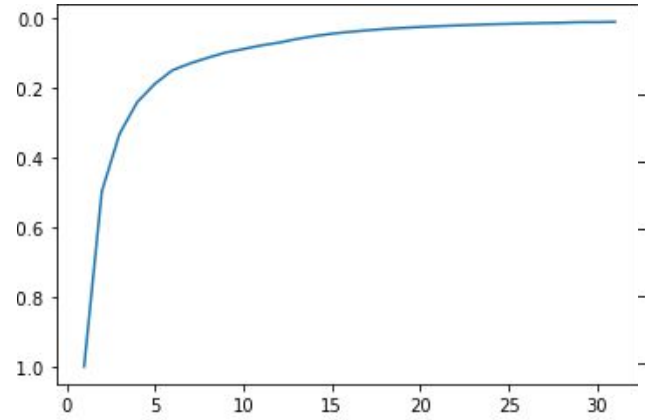


Workstation

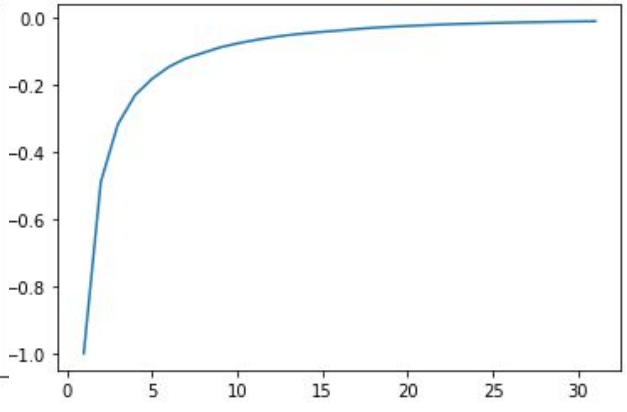


Laptop

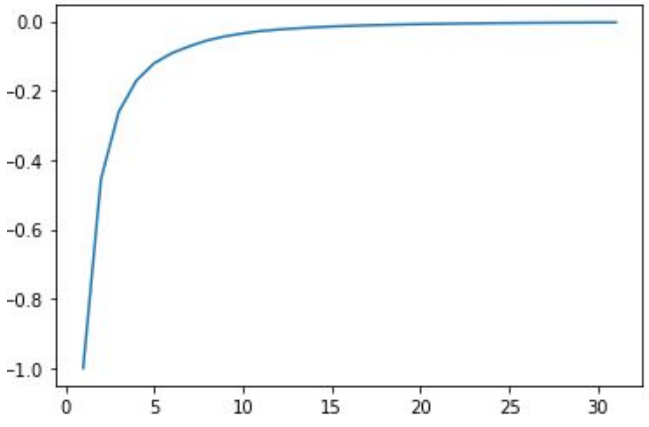
-Efficiency plots



Zeus



Workstation

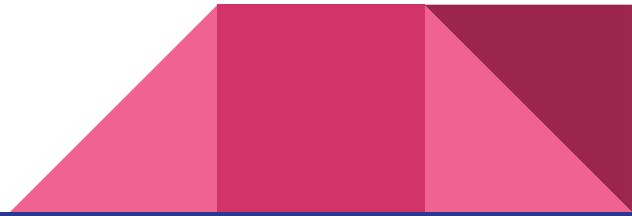


Laptop

SLURM management in jupyter notebook

There are some packedes that enables slurm environment management by adding additional jupyter magics. This not only enables remote method invocation but also in some cases improves visualisation and lets us create new batch jobs easier. During our work we used NERSC surm-magic.

https://github.com/NERSC/slurm-magic



Drawbacks

1. Lower single core performance

Lower single core performance

While running jupyter with SLURM core we should remember that all our actions will be hosted on machine that was designed to run HPC jobs that are dependent on high concurrency and multithreading not on single core performance. As a result single jubs can last longer on HPC. As a reason programs not modeled for concurrent execution can run much longer than on a modern consumer devices.

As an example we run on 3 previously mentioned machines single core implementation of Monte Carlo problem. The results are as follows:

- Zeus: 115.40958571434021 seconds
- Workstation: 12.941054105758667 seconds
- Laptop: 15.544895887374878 seconds

Conclusions

References

https://github.com/NERSC/slurm-magic

https://www.pik-potsdam.de/en/institute/about/it-services/overview/news/jupyter-notebooks-on-the-cluster

http://www.apohllo.pl/blog/skrypt-dla-jupytera-na-prometeuszu