Big Data Analytics 732A54 and TDDE31

Technical Introduction

Maximilian Pfundstein

LINKÖPING UNIVERSITY

Disclaimer

The aim of the labs is not only to learn PySpark, but also to learn how to connect to a cluster, use it and broaden your technical knowledge. Seize that opportunity!

This presentation should give you some hints how to use the NSC Sigma cluster and provide you some theoretical and practical information. This introduction does not cover the programming part of PySpark.

Table of Contents

Table of Contents

Apache Spark and PySpark

Theoretical Introduction

Apache Spark and PySpark

- Apache Spark is written in Java and thus needs the Java JVM to run
- APIs are avilable for Scala, Java, SQL, Python, R
- This course uses Python and thus the PySpark API
- Stand-alone and cluster mode

Linux Systems

Theoretical Introduction

Linux Systems

- Prefer using the CLI rather than GUIs, simplifies the "how-to" long-term
- ThinLinc and a GUI are available for the most parts of your labs
- All relevant information can also be found here:
  - https://www.nsc.liu.se/systems/sigma

Shells

Theoretical Introduction

Shells

- The Terminal is the application, the shell the actual interactor
- Shells:
  - sh
  - bash (default on most Linux systems)
  - fish
  - zsh (new default on macOS since Catalina)

Shells

- Configuration always for each user in `~/.${shell}rc`, e.g. `.bashrc`
- `~` always points to your home directory
- Can be used to setup environment variables and more
- The contents of the file is executed every time you start the shell

Virtual Environments and Modules

Theoretical Introduction

Virtual Environments and Modules

- If you, for example, launch a python script, your OS needs to know where python exetutable (the interpreter) is
- The command `which python` shows the path to the python executable
- If you could change the mapping `python` → `/Users/user/anaconda3/bin/python` to another python installation, you can use multiple versions of python

Virtual Environments and Modules

- There exist programms, that set up environments (venv) or modules for you, that handle this automatically, but it's useful to know what is going on
  - `module`: http://modules.sourceforge.net/
  - `conda`: https://www.anaconda.com/
- Modules are actually doing a bit more, but this will not be part of this introduction
  - `module list`
  - `module avail`

Virtual Environment and Modules

- Exporting environment variables:
  - `export PATH=/Users/${user}/anaconda3/bin:$PATH`
  - `export SBATCH_RESERVATION=devel`

git

Theoretical Introduction

git

- git is a distributed source version-control system
- We cannot cover everything, just some basics
- git is *distributed* and *decentral*, thus GitHub, GitLab, Gitea, bitbucket are "always running" clients
- And maybe do a bit more like Pull Requests, User Management, Wikis, Ticket Tracking, etc.
- The lab is hosted on a self-hosted GitLab instance
  - https://gitlab.liu.se/olaha93/bigdata

git

- git is already installed on unix systems
- Windows: Must install it manually or use one of the CLI recommendations (comes later)
- Good: `git clone git@gitlab.liu.se:olaha93/bigdata.git`
- Bad: `git clone https://gitlab.liu.se/olaha93/bigdata`
- Ugly: Download the zip file. That is like buying a Porsche for grocery shopping

git

- ssh, https, download (https)
- "Forking" is copying a repository on a hosted git-instance from one user to another
- Mainly used for Pull-Requests
- You need to grant access rights to
  - Lab Partners
  - Lab Assistants
  - Teachers
- Simply: Read the readme :)

git

- `git add -A (stages all files)`
- `git commit -m "Commits stages files"`
- `git pull origin master`
- `git push origin master`
- `git remote (add ${name} ${url})`

git

- Merge conflicts happen and are normal!
  - You can prevent them by not working on the same file
- If it happens: Open the conflicted files, search for the conflict, solve it
- Then stage, commit and eventually push the file

git

- GUI clients:
  - GitKraken
  - SourceTree
  - Sublime Merge
  - and many more…

Secure Shell & Keys

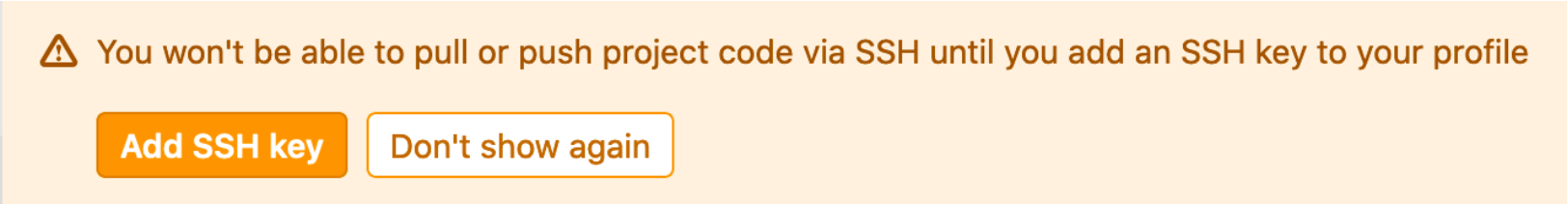
Practical Introduction

Secure Shell & Keys

- Enables to create a remote secure shell, a tunnel
- Can do forward and backwards forwarding
- As well as x-forwarding
- Uses a keypair of a public and a private key, default location is `.ssh`. Unix systems have a default key pair which you can use or use your exiting keys
- If not: `ssh-keygen`
- On Windows (e.g. PuTTY) you must create them on your own or use WSL

Secure Shell & Keys

- git can use https or ssh as the underlying protocol
- If using ssh and key paris, the authentication is automised!
- If you log into any git system (GitHub, GitLab) the first time, they usually want you to upload your **public** key for authentication



```
(base) ➜  .ssh cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC5o2fgA3WMD0IsadxA07xcm/PyCdfqddRm8xC/D
E6jWZjYdRcf2UbrckBx78VJcSpxf8PiWxQBwOrsXgZa6Qp7z6GOYja03EOHux8m2ERX0D+0+UVnKR
LepiHtwlMjbWCShck1hrrzRA5BQ/MSYW41hTZ78+IPO8aeYogkH97RAscD2HiX/oMPlkRxJAl7taj
tGEApKQeAGJUggRFs3D8K20RFLOng1iwlMz7Or3uz10pTK+ABAQAuRoEXorRrbQFWNZ1wMb9cRELY
BQZhpG7EEEiXDCNgcxUnB2c8ns9DDyl0OcfsmgVyHe3SegQkmxYMy07fp88pQbupGECCtcJZ flen
nic@Tridir
```

Add an SSH key

To add an SSH key you need to generate one or use an existing key.

Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id_ed25519.pub' or '~/.ssh/id_rsa.pub' and begins with 'ssh-ed25519' or 'ssh-rsa'. Don't use your private SSH key.

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC5o2fgA3WMD0IsadxA07xcm/PyCdfqddRm8xC
/DE6jWZjYdRcf2UbrckBx78VJcSpxf8PiWxQBwOrsXgZa6Qp7z6GOYja03EOHux8m2ERX0D+0
+UVnKRLepiHtwlMjbWCShck1hrrzRA5BQ/MSYW41hTZ78+IPO8aeYogkH97RAscD2HiX
/oMPlkRxJAl7tajtGEApKQeAGJUggRFs3D8K20RFLOng1iwlMz7Or3uz10pTK+ABAQAuRoEXorR
rbQFWNZ1wMb9cRELYBQZhpG7EEEiXDCNgcxUnB2c8ns9DDyl0OcfsmgVyHe3SegQkmxYMy
07fp88pQbupGECCtcJZ flennic@Tridir

Title

flennic@Tridir

Expires at

dd / mm / 2021

Give your individual key a title

Add key

Demo 1

git setup

Connecting

Practical Introduction

Connecting

- Request Project Membership at SNIC/NSC
  - Project is "LiU-compute-2020-3”
  - https://supr.snic.se/project/request/?search=LiU-compute-2020-3
- Request a login account for Sigma
  - https://supr.snic.se/, login with SWAMID
  - Sigma: https://www.nsc.liu.se/systems/sigma

Connecting

- CLI (SSH)
- GUI (SSH, X-Forwarding)
- ThinLinc

- More Information for GUI: https://www.nsc.liu.se/support/graphics/

Connecting

- `ssh ${account}@sigma.nsc.liu.se`
  - where `${account}` = NSC account name, e.g. x_user
  - The password you chose when requesting an account for Sigma

Connecting

- Want to be lazy?
- `vim ~/.ssh/config`

```
Host sigma
Hostname sigma.nsc.liu.se
Port 22
User ${account}
```

- Then `ssh sigma`

Connecting

- Want to be super lazy? Upload your public key!
  - `ssh-copy-id ${account}@sigma.nsc.liu.se`
  - Issue that command in your **local** terminal!

Connecting

- If you're stuck on Windows, you have three options
- Try to get a well-functioning terminal working. Recommendations:
  1. If up-to-date machine: WSL https://docs.microsoft.com/en-us/windows/wsl/install-win10
  2. Else: https://cmder.net/
  3. Except: Virtual Machine

Connecting

- Some useful Linux commands
  - `ls, mkdir, cd, ssh`
  - `vi, vim, emacs`
  - `scp`

Connecting

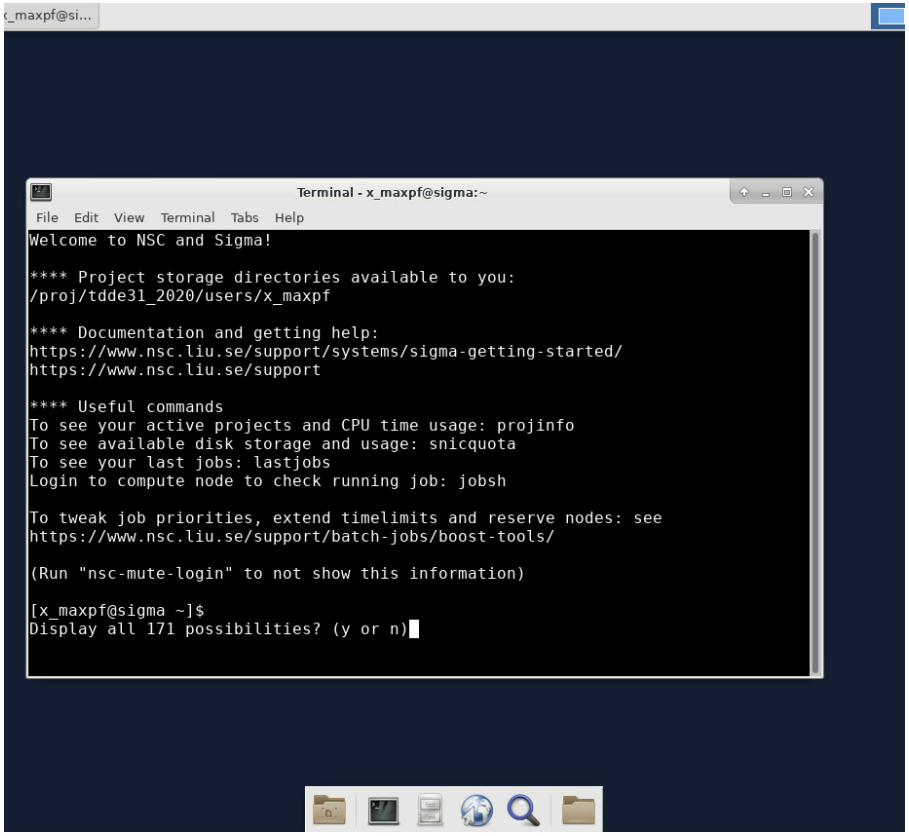
- SSH can do X-forwarding, meaning that you can display a remote GUI applications locally
- Therefore you need a X Window system…
  - macOS: https://www.xquartz.org/
  - Windows: PuTTY
- When you ssh into a machine, add the option **`-X`**
- Don't use `.ssh/config`, it won't set your `$DISPLAY` variable
- Then simply enter "firefox" to start Firefox remotely

Connecting

- `export SBATCH_RESERVATION=devel`
- `spark_browse_historyserver -A liu-compute-2020-3 --reservation devel`
- There you will see an overview of all running jobs on the cluster

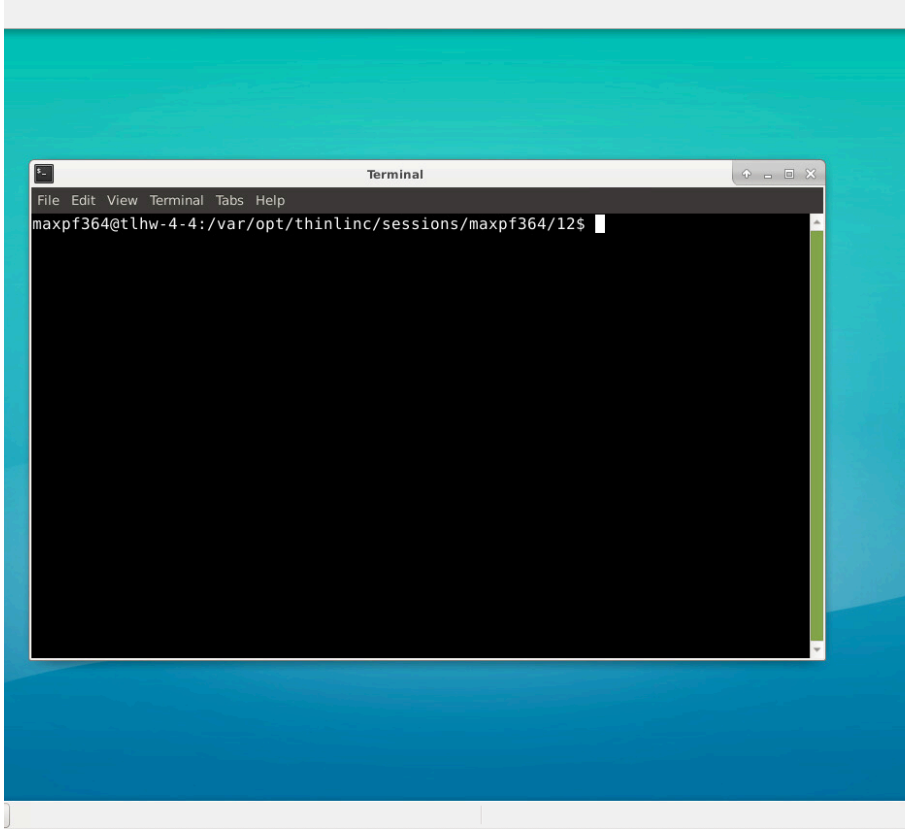
Connecting

- Directly use ThinLinc to connect to the cluster
  - `sigma.nsc.liu.se`
  - `${account}`
  - password



Connecting

- Connect to LiU via ThinLinc (Linux Mint) and do everything from there
  - `thinlinc.edu.liu.se`
  - `{liuid}@student.liu.se`
  - password



Demo 2

connecting

Developing

Practical Introduction

Developing

- Given that you are working on a cluster, writing and executing code are not tightly bound together
- One approach is to simply code local in an editor and then copy the .py file to the cluster and execute it
- Or develop directly on the cluster

Developing

- Using a IDE with a linter will throw a lot of errors as PySpark is most likely not installed
- Recommendations:
  - VS Code with python Plugin
  - PyCharm
  - JupyterLab
  - vim/emacs

Submit a Job

Practical Introduction

1. Copy files
2. Load module
3. Submit Job
4. Monitor Job
5. Retrieve Results

Submit a Job | Copy files

- `scp temperature_readings-small.csv ${account}@sigma.nsc.liu.se:/home/${account}/`
- Adjust username

Submit a Job | Load module

- `cat run.q`
- Inspect contents

Submit a Job | Submit Job

- `sbatch run.q`
  - Manages HDFS
- `squeue -a`
- `squeue -A ${account}`

Submit a Job | Retrieve results

- `tail -f ${file}`
- Look in output

Submit a Job | Copy files

- `scp -r ${account}@sigma.nsc.liu.se:/home/${account}/output*`

Demo 3

submit jobs

Demo 4

local development