



# Uni.lu HPC School 2020

## PS4: Big Data Analytics Batch, stream and hybrid processing Engines



**Uni.lu High Performance Computing (HPC) Team**  
**Dr. S. Varrette**

University of Luxembourg ([UL](#)), Luxembourg

<http://hpc.uni.lu>



## Latest versions available on Github:



UL HPC tutorials:

<https://github.com/ULHPC/tutorials>

UL HPC School:

<http://hpc.uni.lu/hpc-school/>

PS4 tutorial sources:

<ulhpc-tutorials.rtfd.io/en/latest/bigdata/>



# Summary

- 1 Practical Session Objectives
- 2 Interlude: Installing [missing] software with Easybuild
- 3 Python and Data Science
- 4 Big Data Analytics with Hadoop & Spark
  - Apache Hadoop
  - Apache Spark

## Main Objectives of this Session

- **Interlude:** Installing missing software with **Easybuild**
- Review Popular Python-based Data Analytics Frameworks
  - ↪ Sandboxed Environment (**virtualenv**) for reproducible developments
  - ↪ Introduction to **Jupyter Notebooks**
- Introduction to **Big Data analytics**
  - ↪ Distributed File Systems (DFS)
  - ↪ **Mapreduce**
  - ↪ **Hadoop** and **HDFS**
- In-memory [streaming] analysis with **Spark**
  - ↪ Interactive runs
    - ✓ **PySpark**, the Spark Python API; **Scala** and **R** Spark Shell
  - ↪ Standalone cluster runs
  - ↪ Transparent access to master dashboard with SOCK 5 Proxy

## Disclaimer: Acknowledgements

- Part of these slides were **courtesy** borrowed w. permission from:
  - ↪ Prof. Martin Theobald (*Big Data and Data Science Research Group*), UL
- Part of the slides material adapted from:
  - ↪ Advanced Analytics with Spark, O Reilly
  - ↪ Data Analytics with HPC courses
    - ✓ © CC AttributionNonCommercial-ShareAlike 4.0
- similar hands-on material on Github for instance:
  - ↪ Jonathan Dursi: [hadoop-for-hpcers-tutorial](#)

## Before we start...

### Hands-on Pre-requisites

▶ url ◀ | [github](#) | [src](#)

- Access to ULHPC facility ssh  
    ↳ (recommended) Configure/create **GNU Screen** session   `~/.screenrc, screen -S <name>`
- Clone/Pull **ULHPC/tutorials** repository   `~/git/github.com/ULHPC/tutorials`
- Prepare dedicated directory `~/tutorials/bigdata` for this session  
    ↳ set convenient symbolic links to reference material

```
(access)$> mkdir -p ~/tutorials/bigdata
(access)$> cd ~/tutorials/bigdata
# create a symbolic link to the reference material
(access)$> ln -s ~/git/github.com/ULHPC/tutorials/bigdata ref.d
(access)$> ln -s ref.d/scripts .
(access)$> ln -s ref.d/settings .
```

## Before we start...

### Hands-on Pre-requisites

▶ url ◀ | [github](#) | [src](#)

- Access to ULHPC facility ssh  
    ↳ (recommended) Configure/create [GNU Screen](#) session `~/.screenrc, screen -S <name>`
- Clone/Pull [ULHPC/tutorials](#) repository `~/git/github.com/ULHPC/tutorials`
- Prepare dedicated directory `~/tutorials/bigdata` for this session  
    ↳ set convenient symbolic links to reference material

### Hands-on Firefox Extension

▶ url ◀ | [github](#) | [src](#)

- Install SOCKS 5 Proxy plugin

## Summary

- 1 Practical Session Objectives
- 2 Interlude: Installing [missing] software with Easybuild
- 3 Python and Data Science
- 4 Big Data Analytics with Hadoop & Spark
  - Apache Hadoop
  - Apache Spark

## Software/Modules Management

- Easybuild: open-source framework to (automatically) build scientific SW
- Why?: "*Could you please install this software on the cluster?*"
  - ↪ Scientific software is often **difficult** to build
    - ✓ non-standard build tools / incomplete build procedures
    - ✓ hardcoded parameters and/or poor/outdated documentation
  - ↪ EasyBuild helps to facilitate this task
    - ✓ **consistent** software **build and installation** framework,
    - ✓ **automatically generates LMod modulefiles**

```
(node)$> module use $LOCAL_MODULES
(node)$> module load tools/EasyBuild
# Search for recipes for a given software
(node)$> eb -S Spark
(node)$> eb Spark-2.4.5-intel-2019b-Python-3.7.4-Java-1.8.eb -Dr    # Dry-run
(node)$> eb Spark-2.4.5-intel-2019b-Python-3.7.4-Java-1.8.eb -r
```

## ULHPC Toolchains and Software Set Versioning

- **Yearly** release based on Easybuid release of toolchains
  - see Component versions (**fixed per release**) in the **foss** and **intel** toolchains
    - ✓ count 6 months of validation/import after EB release before ULHPC release

Name	Type	2019[a] (prod/old)	2019b (devel)	2020a (next)
GCCCore	compiler	8.2.0	8.3.0	9.3.0
foss	toolchain	2019a	2019b	2020a
intel	toolchain	2019a	2019b	2020a
Python		3.7.2 (and 2.7.15)	3.7.4 (and 2.7.16)	3.8.2

```
# (new) 2019b software set - iris cluster
unset MODULEPATH
module use /opt/apps/resif/iris/2019b/broadwell/modules/all
# OR (when appropriate) skylake/GPU-specialized builds **ONLY** on skylake/GPU nodes
module use /opt/apps/resif/iris/2019b/skylake/modules/all
module use /opt/apps/resif/iris/2019b/gpu/modules/all
```

## ULHPC Toolchains and Software Set Versioning

- Yearly release based on Easybuid release of toolchains
  - see Component versions (**fixed per release**) in the **foss** and **intel** toolchains
    - ✓ count 6 months of validation/import after EB release before ULHPC release

Name	Type	2019[a] (prod/old)	2019b (devel)	2020a (next)
GCCCore	compiler	8.2.0	8.3.0	9.3.0
foss	toolchain	2019a	2019b	2020a
intel	toolchain	2019a	2019b	2020a
Python		3.7.2 (and 2.7.15)	3.7.4 (and 2.7.16)	3.8.2

```
# INCOMING in 2021: (new) 2020a software set - aion cluster
module use /opt/apps/resif/aion/2020a/epyc/modules/all
# iris cluster
module use /opt/apps/resif/iris/2020a/{broadwell,skylake,gpu}/modules/all
```

# Getting Started with Easybuild

## Hands-on Easybuild

▶ url ◀ | [github](#) | [src](#)

- Configure EASYBUILD\_\* variable
- Install/upgrade (local) Easybuild

~/.bashrc, ~/.profile  
./scripts/setup.sh [-n]

```
export EASYBUILD_PREFIX=$HOME/.local/easybuild
export EASYBUILD_MODULES_TOOL=Lmod
export EASYBUILD_MODULE_NAMING_SCHEME=CategorizedModuleNamingScheme
# Use the below variable to run:
#   module use $LOCAL_MODULES
#   module load tools/EasyBuild
export LOCAL_MODULES=${EASYBUILD_PREFIX}/modules/all
function mu(){
    module use $LOCAL_MODULES
    module load tools/EasyBuild
}
```

## Easybuild Installation

```
### You **MUST** be on a compute node to install Easybuild (no 'module' command otherwise)
# (eventually) Source your new config
(node)$> source ~/.bashrc
# Check prefix value
(node)$> echo $EASYBUILD_PREFIX
(node)$> Use setup script to install/update your local Easybuild
# Run the setup script
(node)$> ./scripts/setup.sh -h # help
(node)$> ./scripts/setup.sh -n # Dry-run
(node)$> ./scripts/setup.sh
# Check it
(node)$> eb --version
This is EasyBuild 4.3.2 (framework: 4.3.2, easyblocks: 4.3.2) on host iris-080.
```

# Summary

- 1 Practical Session Objectives
- 2 Interlude: Installing [missing] software with Easybuild
- 3 Python and Data Science

- 4 Big Data Analytics with Hadoop & Spark
  - Apache Hadoop
  - Apache Spark

# Python

- Effective for fast prototype coding
  - Simple (now used a reference language in schools)
  - Easy creation of reproducible and isolated environment
    - ✓ pip: Python package manager
    - ✓ virtualenv: Create virtual environment

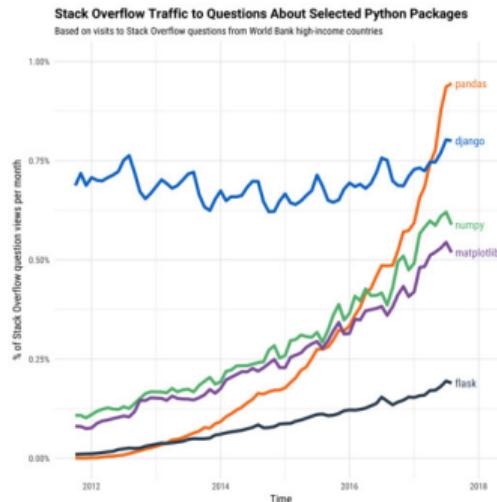
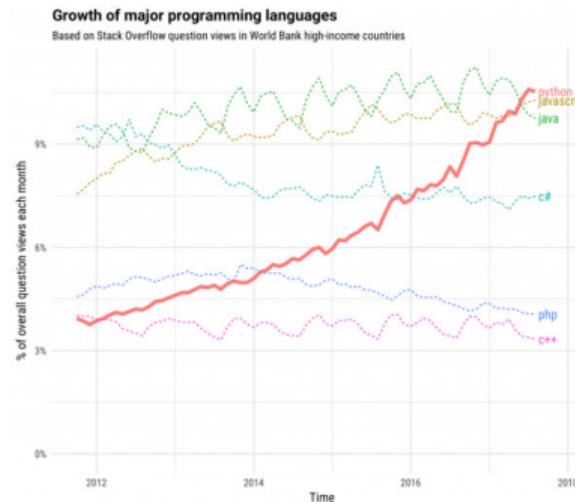
```
$> pip install --user <package>
```

```
$> pip freeze -l > requirements.txt # Dump python environment
```

```
$> pip install -r requirements.txt # Restore saved environment
```

# Python's Role in Data Science

- Dominant language both in **data analytics** and **general programming**, fueled both by:
  - ↪ computational libraries like Numpy, Pandas, and Scikit-Learn
  - ↪ wealth of libraries for visualization, interactive notebooks, collaboration etc.



# Useful Python Libraries / Data Sciences

Library	Description
numpy	Fundamental package for scientific computing
pandas	High-performance, easy-to-use data structures/data analysis tools
iPython	Interactive computing
jupyter (notebook)	Create and share documents that contain live code
Tensorflow, Pytorch	Machine learning frameworks
Scoop	Scalable COncurrent Operations in Python
RAPIDS	GPU Accelerated libraries for data science
Scikit-Learn	Machine Learning in Python
Scikit-image	Image processing in Python
pythran / Cithon	Python->C++ compilation, C bindings
Dask	Flexible library for parallel computing in Python.
...	

## SciPy bundle

```
$> module show lang/SciPy-bundle
-----
/opt/apps/resif/iris/2019b/broadwell/modules/all/lang/SciPy-bundle/[...]
-----
help([[Description
=====
Bundle of Python packages for scientific software
[...]
Included extensions
=====
mpi4py-3.0.2, mpmath-1.1.0, numpy-1.17.3, pandas-0.25.3, scipy-1.3.1
]])
```

## Python Virtualenv / venv

- Virtualenv allows you to create several environments
  - ↪ each will contain their own list of Python packages
  - ↪ Built-in support in **Python 3** for virtual environments with `venv`
- **Best-practice:** **create one virtual environment per project**

```
$> si
# (eventually) load 2019b software set
$> source settings/2019b
# load your preferred **3.x** version of Python - 2.7 DEPRECATED
$> module load lang/Python/3.7.4-GCCcore-8.3.0
$> pip install --user virtualenv
# Create virtualenv (preferred): virtualenv <name>
$> virtualenv venv
# Alternative: python -m venv <name>
# Note: You may want to centralize your virtualenvs under '~/venv/<name>'
# instead of relative to your project directory
```

## Virtualenv activation and population

```
# source <name>/bin/activate
$> source venv/bin/activate
## Populate with your wished data science packages
# jupyter
pip install jupyter
# matplotlib to plot the graph inside your notebook
pip install matplotlib
# To use our virtualenv in the notebook, we need to install this module
pip install ipykernel
```

- Save / Retore your packages

```
# Save versionned configuration - git add it!
pip freeze -l > requirements.txt
```

```
# Restore full install
pip install -r requirements.txt
```

## Application to Jupyter Notebook

```
# Create your own kernel matching <name> virtualenv and launch your Notebook
python -m ipykernel install --user --name=venv
# launch notebook
jupyter notebook --generate-config
jupyter notebook password
jupyter notebook --ip $(ip addr | grep '172.17' | grep 'inet ' | awk '{print $2}' | cut -d/ -f1)
# [I 02:53:08.682 NotebookApp] Serving notebooks from local directory: /mmt/irisgpfs/users/svarret
# [I 02:53:08.683 NotebookApp] Jupyter Notebook 6.1.5 is running at:
# [I 02:53:08.683 NotebookApp] http://172.17.6.118:8888/
# [I 02:53:08.683 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice)
```

- Connect **transparently** to the master Web UI via the SOCK5 proxys  
→ thus **using your browser on your laptop at home**

# Summary

- 1 Practical Session Objectives
- 2 Interlude: Installing [missing] software with Easybuild
- 3 Python and Data Science
- 4 Big Data Analytics with Hadoop & Spark

Apache Hadoop  
Apache Spark

## What is a Distributed File System?

- Straightforward idea: **separate logical from physical storage.**
  - ↪ Not all files reside on a single physical disk,
  - ↪ or the same physical server,
  - ↪ or the same physical rack,
  - ↪ or the same geographical location,...
- **Distributed file system (DFS):**
  - ↪ virtual file system that enables clients to access files
    - ✓ ... as if they were stored locally.

# What is a Distributed File System?

- Straightforward idea: **separate logical from physical storage.**
  - ↪ Not all files reside on a single physical disk,
  - ↪ or the same physical server,
  - ↪ or the same physical rack,
  - ↪ or the same geographical location,...
- **Distributed file system (DFS):**
  - ↪ virtual file system that enables clients to access files
    - ✓ ... as if they were stored locally.
- **Major DFS distributions:**
  - ↪ **NFS**: originally developed by Sun Microsystems, started in 1984
  - ↪ **AFS/CODA**: originally prototypes at Carnegie Mellon University
  - ↪ **GFS**: Google paper published in 2003, not available outside Google
  - ↪ **HDFS**: designed after GFS, part of Apache Hadoop since 2006

# Distributed File System Architecture?

## Master-Slave Pattern

- Single (or few) **master** nodes maintain state info. about clients
- All clients R&W requests go through the global master node.
- **Ex:** GFS, HDFS

# Distributed File System Architecture?

## Master-Slave Pattern

- Single (or few) **master** nodes maintain state info. about clients
- All clients R&W requests go through the global master node.
- **Ex:** GFS, HDFS

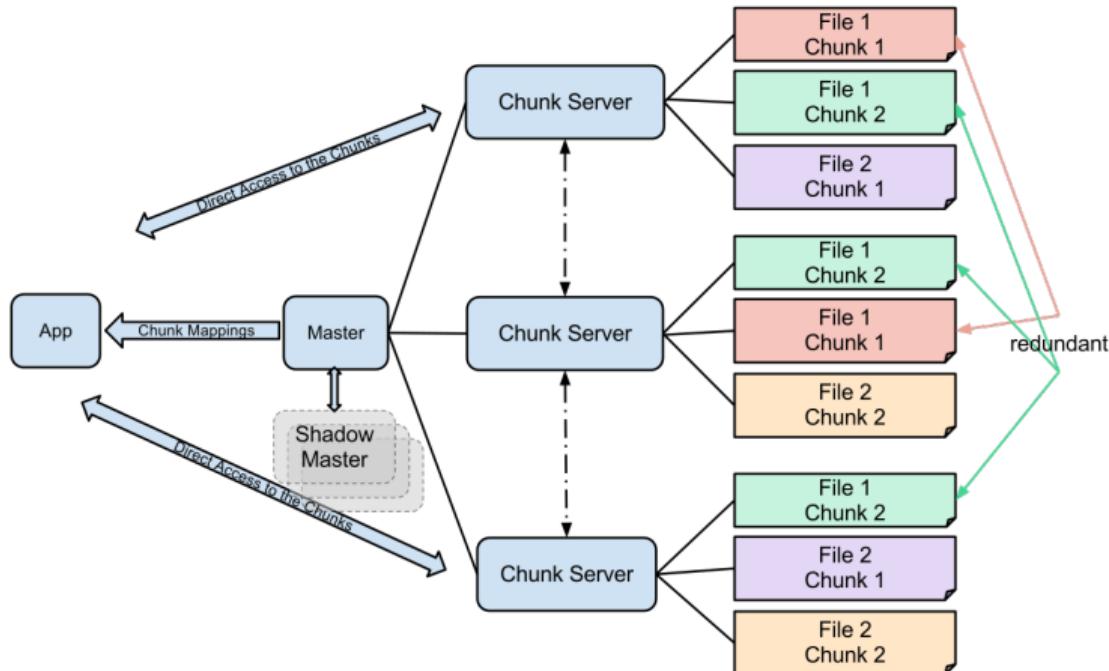
## Peer-to-Peer Pattern

- No global state information.
- Each node may both serve and process data.

## Google File System (GFS) (2003)

- Radically different architecture compared to NFS, AFS and CODA.
  - specifically tailored towards **large-scale** and **long-running analytical processing tasks**
  - over thousands of storage nodes.
- **Basic assumption:**
  - client nodes (aka. *chunk servers*) may fail any time!
  - Bugs or hardware failures.
  - Special tools for monitoring, periodic checks.
  - Large files (multiple GBs or even TBs) are split into 64 MB *chunks*.
  - Data modifications are mostly append operations to files.
  - Even the master node may fail any time!
    - ✓ Additional *shadow master* fallback with read-only data access.
- Two types of reads: Large sequential reads & small random reads

# Google File System (GFS) (2003)



## GFS Consistency Model

- **Atomic File Namespace Mutations**

- File creations/deletions centrally controlled by the master node.
- Clients typically create and write entire file,
  - ✓ then add the file name to the file namespace stored at the master.

- **Atomic Data Mutations**

- only 1 atomic modification of 1 replica (!) at a time is guaranteed.

- **Stateful Master**

- Master sends regular **heartbeat** messages to the chunk servers
- Master keeps chunk locations of all files (+ replicas) in memory.
- locations not stored persistently...
  - ✓ but polled from the clients at startup.

- **Session Semantics**

- Weak consistency model for file replicas and client caches only.
- Multiple clients may read and/or write the same file concurrently.
- The client that last writes to a file **wins**.

# Fault Tolerance & Fault Detection

- **Fast Recovery**

- master & chunk servers can restore their states and (re-)start in s.
    - ✓ regardless of previous termination conditions.

- **Master Replication**

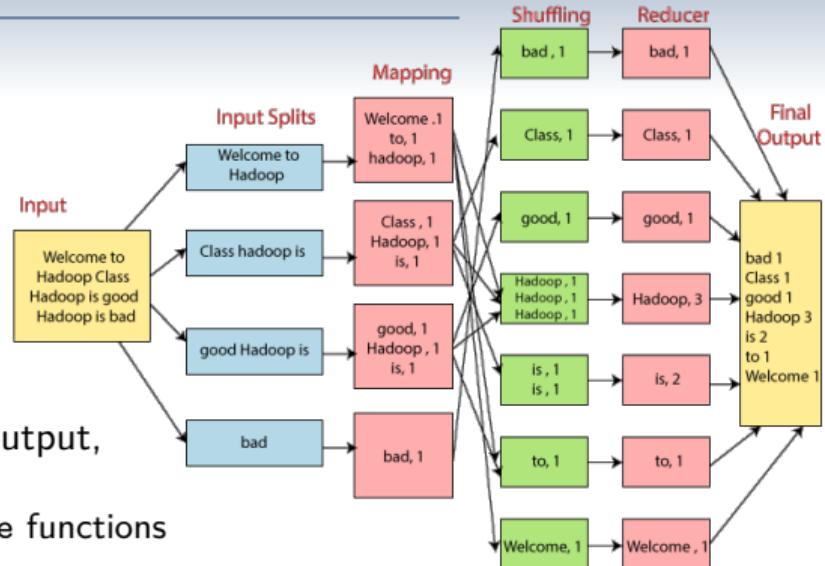
- *shadow master* provides RO access when primary master is down.
    - ✓ Switches back to read/write mode when primary master is back.
  - Master node does not keep a persistent state info. of its clients,
    - ✓ rather polls clients for their states when started.

- **Chunk Replication & Integrity Checks**

- chunk divided into 64 KB blocks, each with its own 32-bit checksum
    - ✓ verified at read and write times.
  - Higher replication factors for more intensively requested chunks (**hotspots**) can be configured.

# Map-Reduce

- Breaks the processing into two main phases:
  - 1 the **map** phase
  - 2 the **reduce** phase.
- Each phase has **key-value pairs** as input and output,
  - types may be chosen by the programmer.
  - the programmer also specifies map and reduce functions



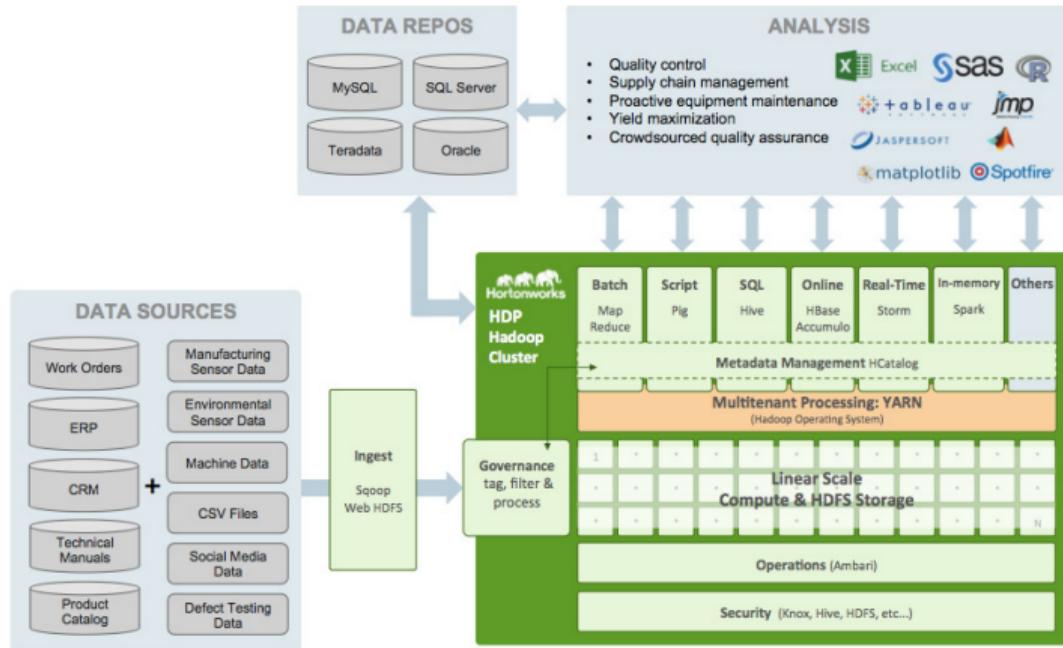
## Hadoop

- Initially started as a student project at Yahoo! labs in 2006
  - ↪ Open-source Java implem. of GFS and MapReduce frameworks
- Switched to Apache in 2009. Now consists of three main modules:
  - ① **HDFS**: Hadoop distributed file system
  - ② **YARN**: Hadoop job scheduling and resource allocation
  - ③ **MapReduce**: Hadoop adaptation of the MapReduce principle

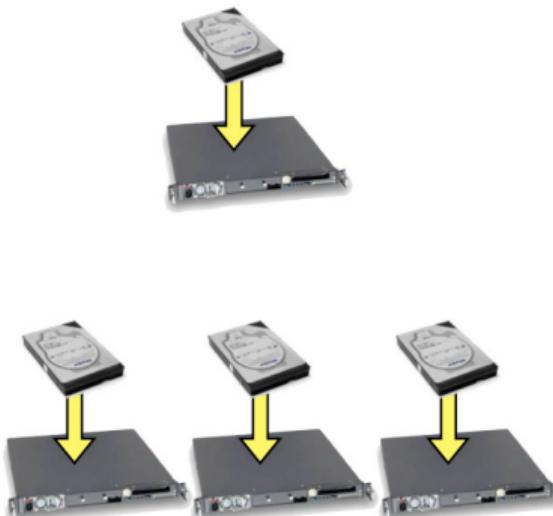
## Hadoop

- Initially started as a student project at Yahoo! labs in 2006
  - ↪ Open-source Java implem. of GFS and MapReduce frameworks
- Switched to Apache in 2009. Now consists of three main modules:
  - ① **HDFS**: Hadoop distributed file system
  - ② **YARN**: Hadoop job scheduling and resource allocation
  - ③ **MapReduce**: Hadoop adaptation of the MapReduce principle
- Basis for many other open-source Apache toolkits:
  - ↪ **PIG/PigLatin**: file-oriented data storage & script-based query language
  - ↪ **HIVE**: distributed SQL-style data warehouse
  - ↪ **HBase**: distributed key-value store
  - ↪ **Cassandra**: fault-tolerant distributed database, etc.
- HDFS still mostly follows the original GFS architecture.

## Hadoop Ecosystem

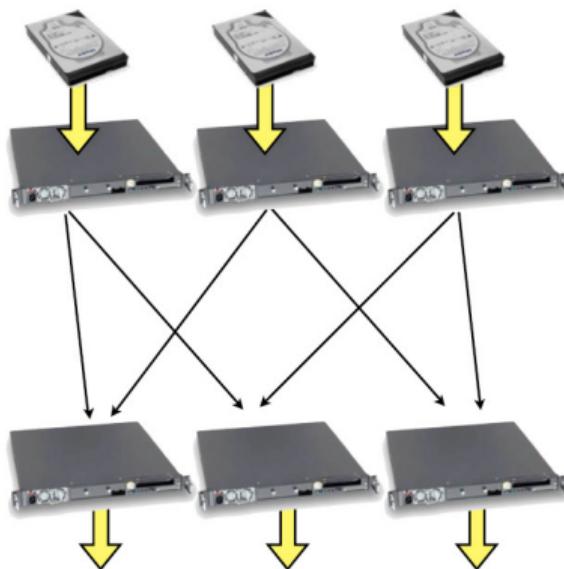


## Scale-Out Design



- HDD streaming speed  $\sim 50\text{MB/s}$ 
  - ↪ 3TB = 17.5 hrs
  - ↪ 1PB = 8 months
- Scale-out (weak scaling)
  - ↪ **FS distributes data on ingest**
- Seeking too slow
  - ↪  $\sim 10\text{ms}$  for a seek
  - ↪ Enough time to read half a megabyte
- **Batch processing**
- Go through entire data set in one (or small number) of passes

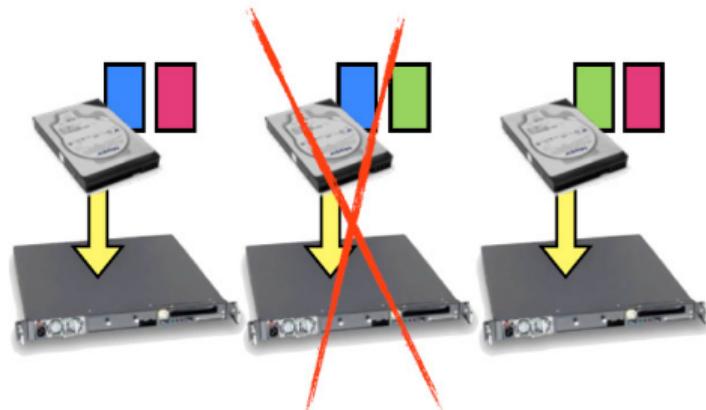
## Combining Results



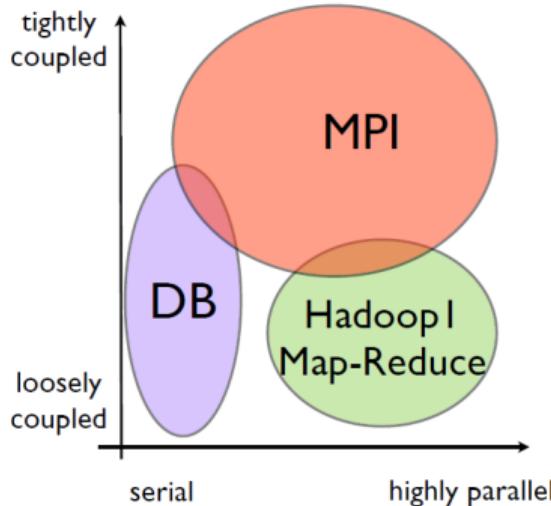
- Each node preprocesses its local data
  - ↪ Shuffles its data to a small number of other nodes
- Final processing, output is done there

# Fault Tolerance

- Data also replicated upon ingest
- Runtime watches for dead tasks, restarts them on live nodes
- Re-replicates



# Hadoop: What is it Good At?



- **Compare with databases**

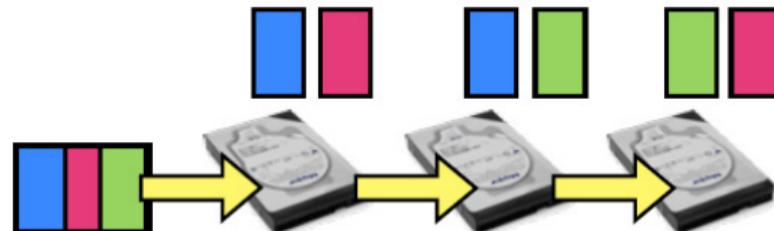
- **Compare with databases**
  - ↪ very good at working on small subsets of large databases
    - ✓ DBs: very interactive for many tasks
    - ✓ ... yet have been difficult to scale

- **Compare with HPC (MPI)**

- **Compare with HPC (MPI)**
  - ↪ Also typically batch
  - ↪ Can (and does) go up to enormous scales
- **Works extremely well for very tightly coupled problems:**
  - ↪ zillions of iterations/timesteps/ exchanges.

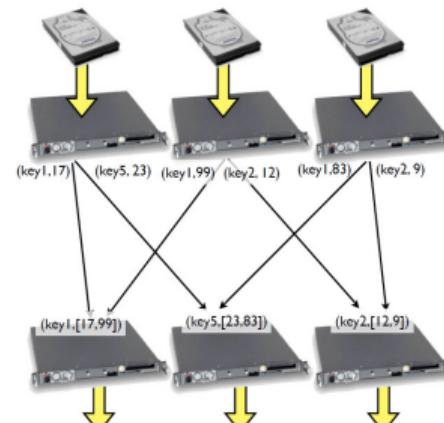
## Data Distribution: Disk

- Hadoop & al. arch. handle the hardest part of parallelism for you
  - ↪ aka **data distribution**.
- **On disk:**
  - ↪ HDFS distributes, replicates data as it comes in
  - ↪ Keeps track of computations local to data



## Data Distribution: Network

- **On network:** Map Reduce (eg) works in terms of key-value pairs.
  - ↪ Preprocessing (map) phase ingests data, emits  $(k, v)$  pairs
  - ↪ Shuffle phase assigns reducers,
    - ✓ gets all pairs with same key onto that reducer.
  - ↪ Programmer does not have to design communication patterns



## Makes the problem easier

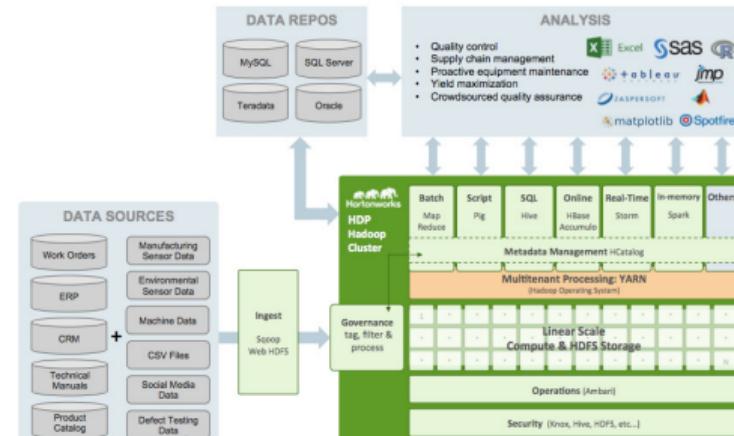
- Hardest parts of parallel programming with HPC tools
  - ↪ Decomposing the problem, and,
  - ↪ Getting the intermediate data where it needs to go,

- Hadoop does that for you

- ↪ automatically
  - ↪ for a wide range of problems.

## Built a Reusable Substrate

- HDFS and the MapReduce layer were nicely architected.
  - ↪ Enables many higher-level tools
  - ↪ Data analysis, machine learning, NoSQL DBs,...
- Very productive environment
  - ↪ ... and Hadoop 2.x (YARN) is now much much more than just MapReduce



# The Hadoop Filesystem

- **HDFS is a distributed parallel filesystem**
  - ↪ Not a general purpose file system
    - ✓ does not implement posix
    - ✓ cannot just mount it and view files
- Access via `hdfs fs` commands or programmatic APIs
  - ↪ relatively complex to interact with

```
$> hdfs fs -[cmd]
```

cat	chgrp
chmod	chown
copyFromLocal	copyToLocal
cp	du
dus	expunge
get	getmerge
ls	lsr
mkdir	movefromLocal
mv	put
rm	rmr
setrep	stat
tail	test
text	touchz

## The Hadoop Filesystem

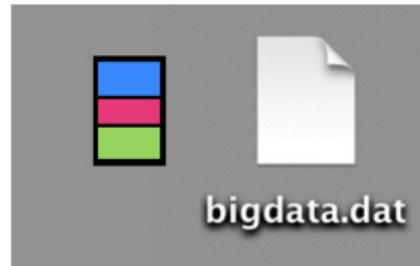
- **Required** to be:

- ↪ able to deal with large files, large amounts of data
- ↪ scalable & reliable in the presence of failures
- ↪ fast at reading contiguous streams of data
- ↪ only need to write to new files or append to files
- ↪ require only commodity hardware

- **As a result:**

- ↪ Replication
- ↪ Supports mainly high bandwidth, **not** especially low latency
- ↪ No caching
  - ✓ what is the point if primarily for streaming reads?
  - ✓ Poor support for seeking around files
  - ✓ Poor support for millions of files
- ↪ Have to use separate API to see filesystem
- ↪ Modelled after Google File System (2004 Map Reduce paper)

## Hadoop vs [HPC] File Systems

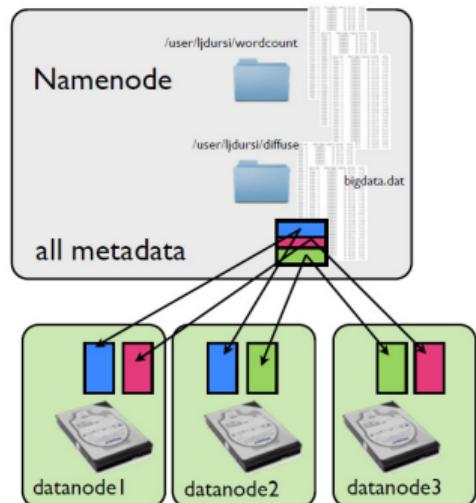


- HDFS is a **block-based FS**
  - ↪ A file is broken into blocks,
  - ↪ these blocks are distributed across nodes
- **Blocks are large;**
  - ↪ 64MB is default,
  - ↪ many installations use 128MB or larger
- Large block size
  - ↪ time to stream a block much larger than time disk time to access the block.



```
# Lists all blocks in all files:  
$> hdfs fsck / -files -blocks
```

# Datanodes and Namenode



Two types of nodes in the filesystem:

## ① Namenode

- ↪ stores all metadata / block locations in memory
- ↪ Metadata updates stored to persistent journal

## ② Datanodes

- ↪ store/retrieve blocks for client/namenode

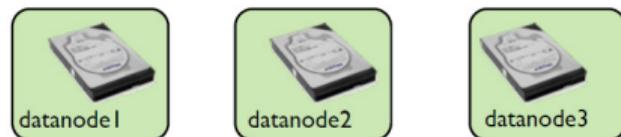
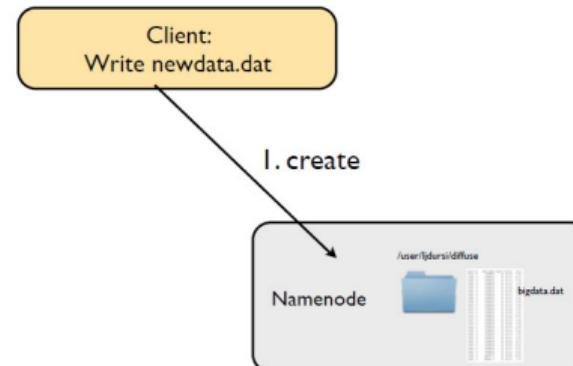
- Newer versions of Hadoop: federation

- ↪ ≠ namenodes for /user, /data...
- ↪ High Availability namenode pairs

## Writing a file

- **Writing a file** multiple stage process:

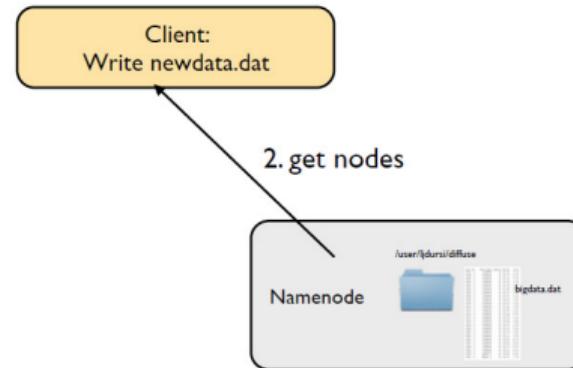
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



## Writing a file

- **Writing a file** multiple stage process:

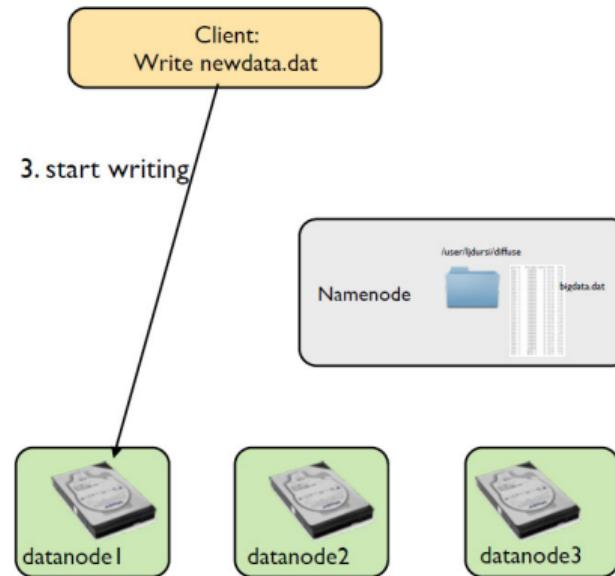
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



## Writing a file

- **Writing a file** multiple stage process:

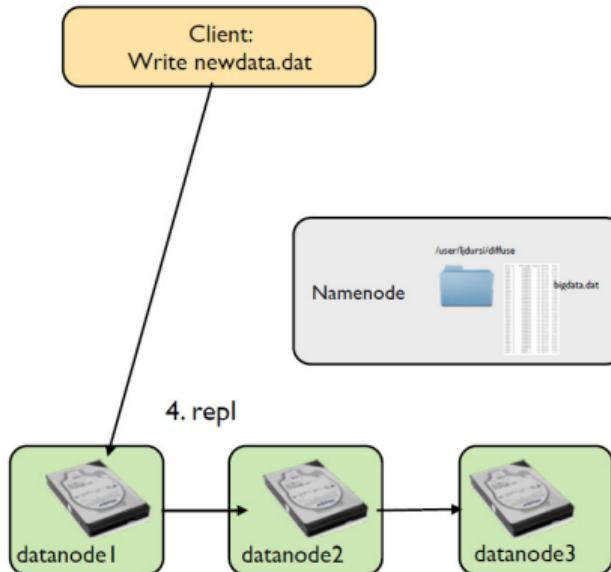
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



## Writing a file

- **Writing a file** multiple stage process:

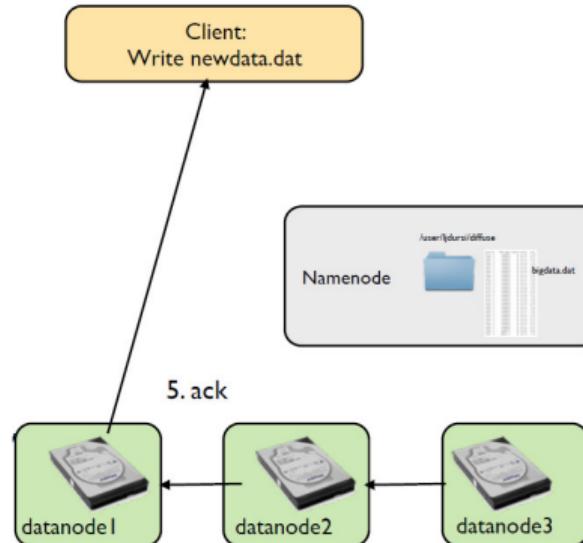
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



# Writing a file

- **Writing a file** multiple stage process:

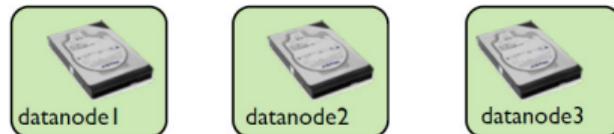
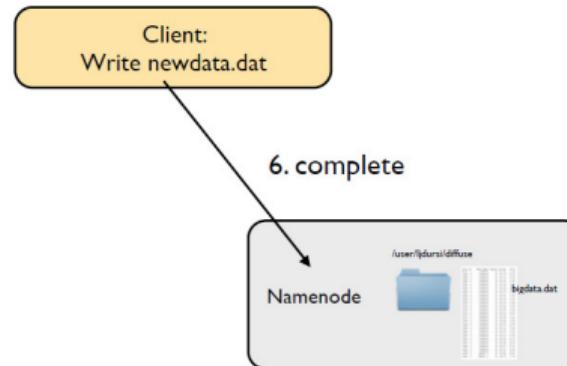
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back (**while writing**)
- Complete



## Writing a file

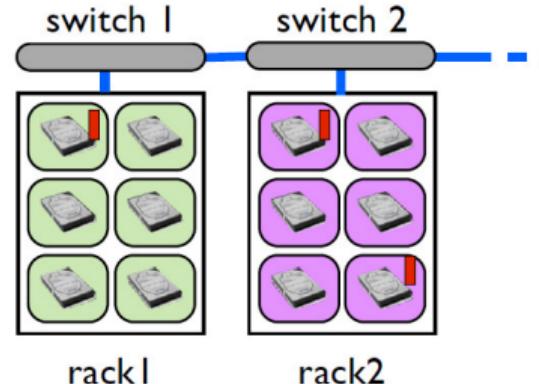
- **Writing a file** multiple stage process:

- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back (**while writing**)
- Complete



## Where to Replicate?

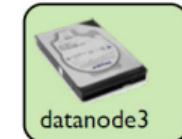
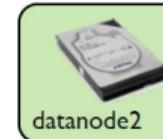
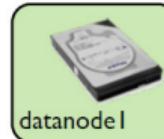
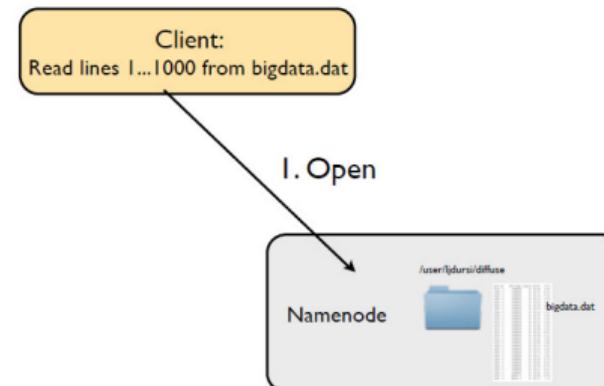
- **Tradeoff** to choosing replication locations
  - **Close**: faster updates, less network bandwidth
  - **Further**: better failure tolerance
- **Default strategy:**
  - ① copy on different location on same node
  - ② second on different *rack*(switch),
  - ③ third on same rack location, different node.
- Strategy configurable.
  - Need to configure Hadoop file system to know location of nodes



## Reading a file

- **Reading a file**

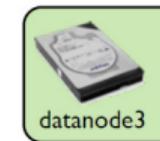
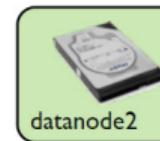
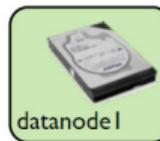
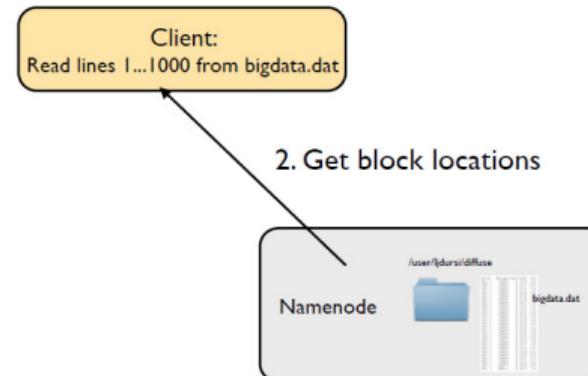
- Open call
- Get block locations
- Read from a replica



## Reading a file

- **Reading a file**

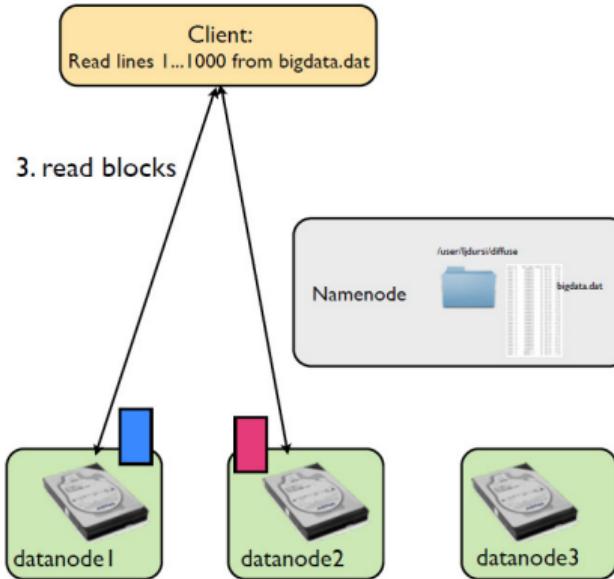
- Open call
- Get block locations
- Read from a replica



## Reading a file

- **Reading a file**

- Open call
- Get block locations
- Read from a replica



## Configuring HDFS

- Need to tell HDFS how to set up filesystem
  - ↪ `data.dir, name.dir`
    - ✓ where on local system (eg, local disk) to write data
  - ↪ parameters like replication
    - ✓ how many copies to make
  - ↪ default name - default file system to use
  - ↪ Can specify multiple FSs

## Configuring HDFS

```
<configuration>          <!-- $HADOOP_PREFIX/etc/hadoop/core-site.xml -->
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://<server>:9000</value>
  </property>
  <property>
    <name>dfs.data.dir</name>
    <value>/home/username/hdfs/data</value>
  </property>
  <property>
    <name>dfs.name.dir</name>
    <value>/home/username/hdfs/name</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
</configuration>
```

## Configuring HDFS

- In Practice, in single mode
  - ↪ Only one node to be used, the VM
  - ↪ **default server:** localhost
  - ↪ Since only one node:
    - ✓ need to specify replication factor of 1, or will always fail

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
[...]
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
```

## Configuring HDFS - General Scenario

- You will need to make sure that environment variables (path to Java/Hadoop etc.) are set
  - ↪ Easybuild does **most** of the job for you
  - ↪ Still a lot to define in {hadoop, mapred, yarn}-env.sh
- You will need passwordless SSH access across all nodes
- You can then start processes on various FS nodes
- Other configuration files (outside core-site.xml) to check:
  - ↪ \$HADOOP\_PREFIX/etc/hadoop/{hdfs-site, mapred-site, yarn-site}.xml
  - ↪ slaves list

## Configuring HDFS - General Scenario

- You will need to make sure that environment variables (path to Java/Hadoop etc.) are set
  - ↪ Easybuild does **most** of the job for you
  - ↪ Still a lot to define in {hadoop, mapred, yarn}-env.sh
- You will need passwordless SSH access across all nodes
- You can then start processes on various FS nodes
- Other configuration files (outside core-site.xml) to check:
  - ↪ \$HADOOP\_PREFIX/etc/hadoop/{hdfs-site, mapred-site, yarn-site}.xml
  - ↪ slaves list
- Once configuration files are set up,
  - ↪ you can format the namenode like so you can start up just the file systems

```
$> hdfs namenode -format  
$> start-dfs.sh
```

## Using HDFS

- Once the file system is up and running,  
→ ... you can copy files back and forth

```
$> hadoop fs -{get|put|copyFromLocal|copyToLocal} [...]
```

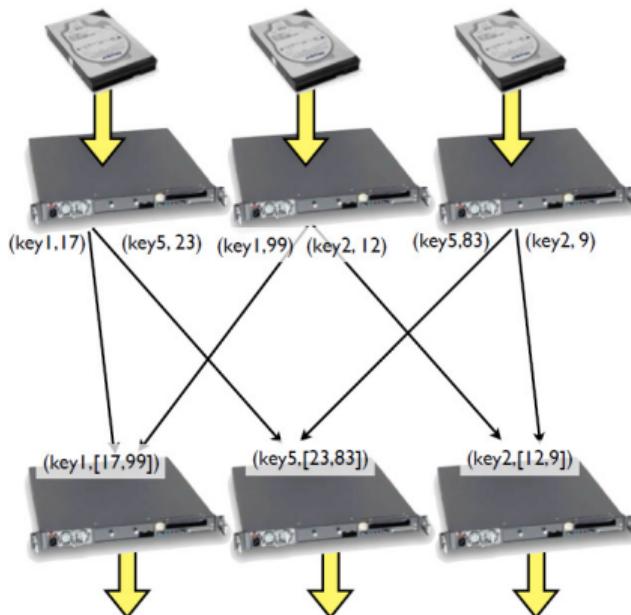
- Default directory is /user/\${username}  
→ Nothing like a cd

```
$> hdfs fs -mkdir /home/vagrant/hdfs-test
$> hdfs fs -ls    /home/vagrant
$> hdfs fs -ls    /home/vagrant/hdfs-test
$> hdfs fs -put data.dat /home/vagrant/hdfs-test
$> hdfs fs -ls    /home/vagrant/hdfs-test
```

## Using HDFS

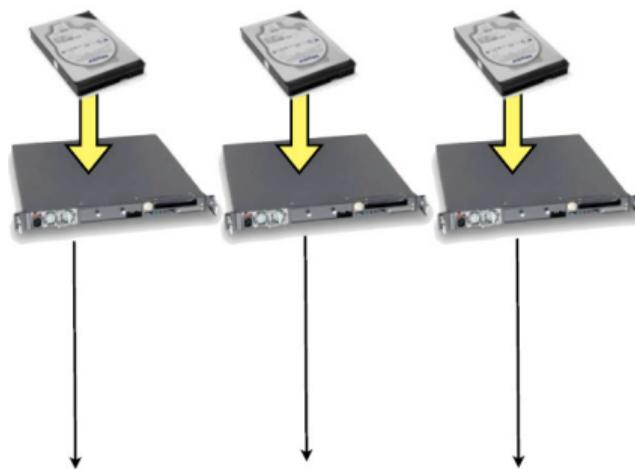
- In general, the data files you send to HDFS will be **large**  
    ↳ or else why bother with Hadoop.
- Do not want to be constantly copying back and forth  
    ↳ **view, append in place**
- Several APIs to accessing the HDFS  
    ↳ Java, C++, Python
- Here, we use one to get a file status, and read some data from it at some given offset

## Back to Map-Reduce



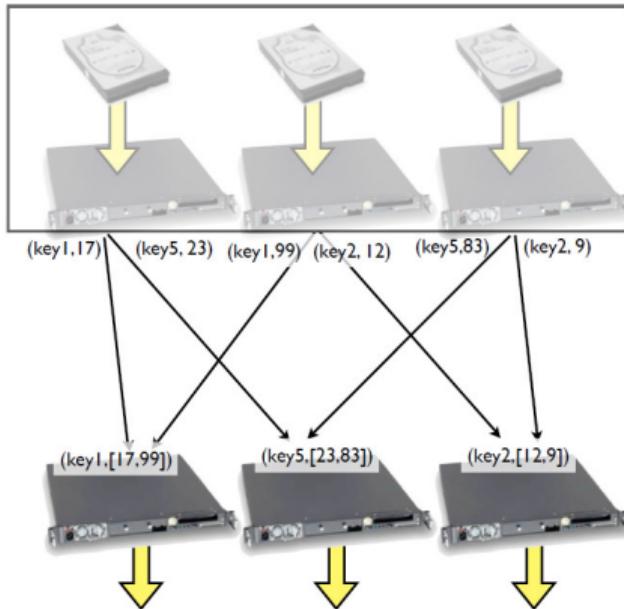
- Map processes **one element at a time**
  - ↪ emits results as (key, value) pairs.
- All results with **same key are gathered to the same reducers**
  - ↪ Reducers process list of values
  - ↪ emit results as (key, value) pairs

## Map



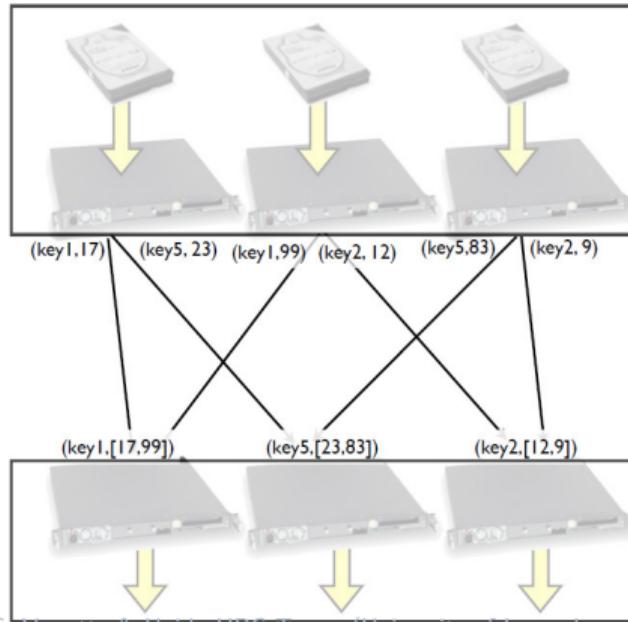
- All coupling done during **shuffle** phase
  - ↪ Embarrassingly parallel task
  - ↪ all map
- Take input, map it to output, done.
- **Famous case**
  - ↪ NYT using Hadoop to convert 11 million image files to PDFs
    - ✓ almost pure serial farm job

## Reduce



- Reducing gives the coupling
- In the case of the NYT task:
  - not quite embarrassingly parallel:
    - ✓ images from multi-page articles
    - ✓ Convert a page at a time,
    - ✓ gather images with same article id onto node for conversion

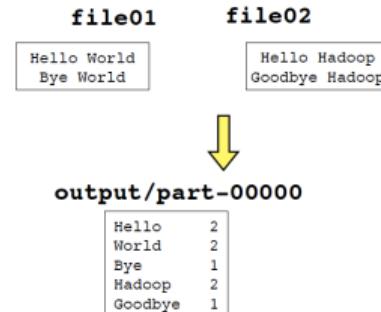
# Shuffle



- **Shuffle is part of the Hadoop magic**
  - ↪ By default, keys are hashed
  - ↪ hash space is partitioned between reducers
- **On reducer:**
  - ↪ gathered (k,v) pairs from mappers are sorted by key,
  - ↪ then merged together by key
  - ↪ Reducer then runs on one (k,[v]) tuple at a time
- **you can supply your own partitioner**
  - ↪ Assign **similar** keys to same node
  - ↪ Reducer still only sees one (k, [v]) tuple at a time.

## Example: Wordcount

- Was used as an example in the original MapReduce paper
  - Now basically the **hello world** of map reduce
- Problem description:** Given a **set** of documents:
  - count occurrences of words within these documents



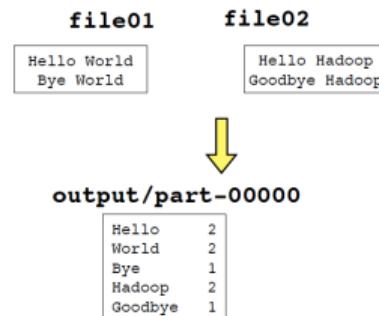
## Example: Wordcount

- How would you do this with a huge document?

- ↪ Each time you see a word:
  - ✓ if it is a new word, add a tick mark beside it,
  - ✓ otherwise add a new word with a tick

- ... But hard to parallelize

- ↪ pb when updating the list



## Example: Wordcount

**file01**

Hello World  
Bye World

**file02**

Hello Hadoop  
Goodbye Hadoop



**output/part-00000**

Hello	2
World	2
Bye	1
Hadoop	2
Goodbye	1

- **MapReduce way**

- ↪ all hard work done automatically by shuffle

- **Map:**

- ↪ just emit a 1 for each word you see

- **Shuffle:**

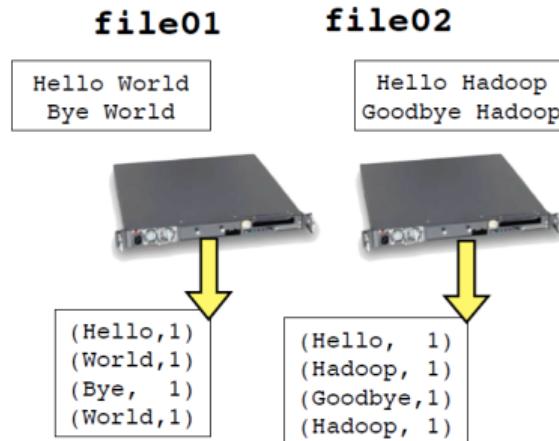
- ↪ assigns keys (words) to each reducer,

- ↪ sends (k,v) pairs to appropriate reducer

- **Reducer**

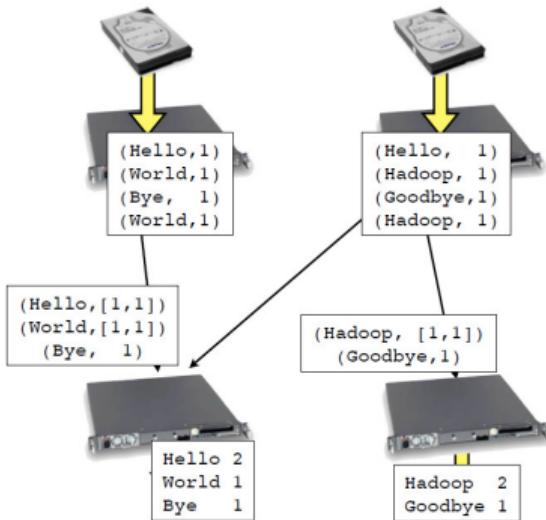
- ↪ just has to sum up the ones

## Example: Wordcount



- **MapReduce way**
  - ↪ all hard work done automatically by shuffle
- **Map:**
  - ↪ just emit a 1 for each word you see
- **Shuffle:**
  - ↪ assigns keys (words) to each reducer,
  - ↪ sends (k,v) pairs to appropriate reducer
- **Reducer**
  - ↪ just has to sum up the ones

## Example: Wordcount



- **MapReduce way**

→ all hard work done automatically by shuffle

- **Map:**

→ just emit a 1 for each word you see

- **Shuffle:**

→ assigns keys (words) to each reducer,

→ sends (k,v) pairs to appropriate reducer

- **Reducer**

→ just has to sum up the ones

# Getting Started with Hadoop

## Hands-on / Hadoop

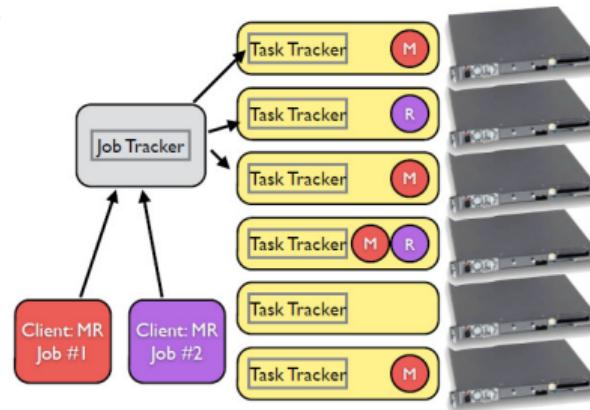
▶ url ◀ | [github](#) | [src](#)

- Install latest Hadoop from the provided easyconfig  
    ↳ interactive node --ntasks-per-node 1 -c 14 to accelerate the build
- Test the tools/Hadoop modules in Single mode  
    ↳ test on a Map-reduce grep application
- (eventually) [Pseudo-Distributed Operation](#) and [Full Cluster Setup](#)

```
(access)$> si -c 14      # In normal times: target all cores i.e. 28
(node)$> source settings/2019b
(node)$> eb --version
# Install Hadoop
(node)$> eb src/Hadoop-2.10.1-GCCcore-8.3.0-native.eb -Dr    # Dry-run, check dependencies
(node)$> eb src/Hadoop-2.10.1-GCCcore-8.3.0-native.eb -r
```

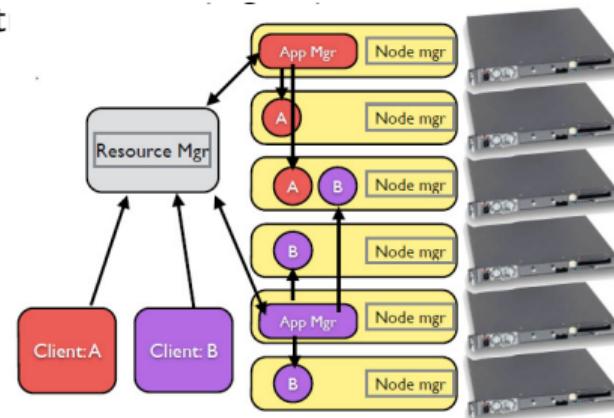
# Hadoop 1x

- Original Hadoop was basically HDFS + infrastructure for MapReduce
  - ↪ Very faithful implementation of Google MapReduce paper.
  - ↪ Job tracking, orchestration all very tied to M/R model
- Made it difficult to run other sorts of jobs



## YARN and Hadoop 2

- **YARN:** Yet Another Resource Negotiator
  - ↪ Looks a lot more like a cluster scheduler/resource manager
  - ↪ Allows arbitrary jobs.
- Allow for new compute/data tools. Ex: st



## Apache Spark



- Spark is (yet) a(-nother) distributed, **Big Data** processing platform.
  - ↪ Everything you can do in Hadoop, you can also do in Spark.

### In contrast to Hadoop

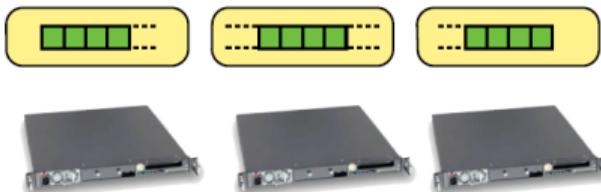
- Spark computation paradigm is not **just** MapReduce job
- Key feature - **in-memory analyses**.
  - ↪ **multi-stage, in-memory dataflow graph based on Resilient Distributed Datasets (RDDs)**.

## Apache Spark



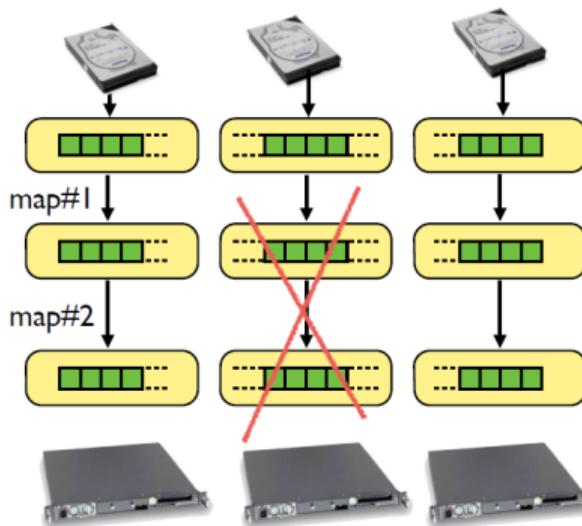
- Spark is implemented in Scala, running in a Java Virtual Machine.
  - ↪ Spark supports different languages for application development:
    - ✓ Java, Scala, Python, R, and SQL.
- Originally developed in AMPLab (UC Berkeley) from 2009,
  - ↪ donated to the Apache Software Foundation in 2013,
  - ↪ top-level project as of 2014.
- **Latest release:** 2.2.1 (Dec. 2017)

## RDD



- Resilient Distributed Dataset (RDD)
  - ↪ Partitioned collections (lists, maps..) across nodes
  - ↪ Set of well-defined operations (incl map, reduce) defined on these RDDs.

# RDD



- Fault tolerance works three ways:
  - ↪ Storing, reconstructing lineage
  - ↪ Replication (optional)
  - ↪ Persistence to disk (optional)

## RDD Lineage

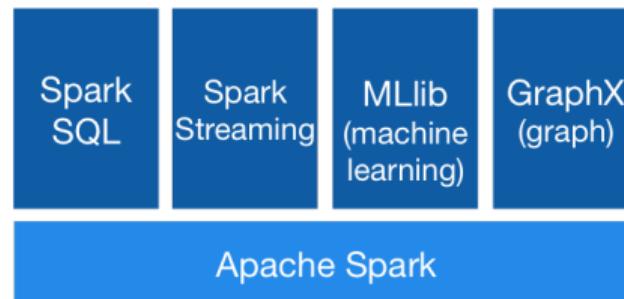
- Map Reduce implemented FT by outputting everything to disk always.
  - ↪ Effective but extremely costly.
  - ↪ **How to maintain fault tolerance without sacrificing in-memory performance?**
    - ✓ for truly large-scale analyses

## RDD Lineage

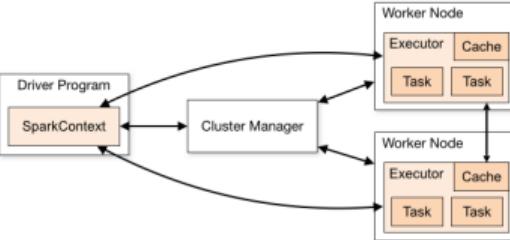
- Map Reduce implemented FT by outputting everything to disk always.
  - ↪ Effective but extremely costly.
  - ↪ **How to maintain fault tolerance without sacrificing in-memory performance?**
    - ✓ for truly large-scale analyses
- **Solution:**
  - ↪ Record lineage of an RDD (think version control)
  - ↪ If container, node goes down, reconstruct RDD from scratch
    - ✓ Either from beginning,
    - ✓ or from (occasional) checkpoints which user has some control over.
  - ↪ User can suggest caching current state of RDD in memory,
    - ✓ or persisting it to disk, or both.
  - ↪ You can also save RDD to disk, or replicate partitions across nodes for other forms of fault tolerance.

## Main Building Blocks

- The **Spark Core API** provides the general execution layer
  - ↪ on top of it, all other functionality is built upon.
- Four higher-level components (in the \_Spark ecosystem):
  - ① **Spark SQL** (formerly **Shark**),
  - ② **Streaming**, to build scalable fault-tolerant streaming applications.
  - ③ **MLlib** for machine learning
  - ④ **GraphX**, the API for graphs and graph-parallel computation



# Getting Started with Spark



## Hands-on: Multi-node Spark Standalone Cluster

[▶ url](#) [◀](#) | [github](#) | [src](#)

- (eventually) reconnect with SOCK5 proxy enabled
- Load Spark from the 2019b software set
- Check a single **interactive run**
  - ↪ **PySpark**, the Spark Python API; **Scala** Spark Shell
- Deploy **Spark standalone cluster** launcher.Spark.sh -i; spark-submit [...]
  - ↪ use ULHPC launcher to deploy an (interactive) Spark cluster
    - ✓ spawn **master** and **worker** Spark processes interactively (-i)
  - ↪ access the **web UI** of the master
    - ✓ **Ex:** SOCKS 5 proxy approach + FoxyProxy plugin
  - ↪ submit a **sample job** (Pi estimation)
    - ✓ spark-submit [...] \$EBROOTSPARK/examples/src/main/python/pi.py 1000
- Scale across multiple nodes

```
ssh -D 1080 iris-cluster
module load devel/Spark
pyspark,spark-shell
```

```
launcher.Spark.sh -i; spark-submit [...]
```

↪ use ULHPC launcher to deploy an (interactive) Spark cluster

✓ spawn **master** and **worker** Spark processes interactively (-i)

↪ access the **web UI** of the master

✓ **Ex:** SOCKS 5 proxy approach + FoxyProxy plugin

↪ submit a **sample job** (Pi estimation)

✓ spark-submit [...] \$EBROOTSPARK/examples/src/main/python/pi.py 1000

## Multi-node Spark Cluster Deployment

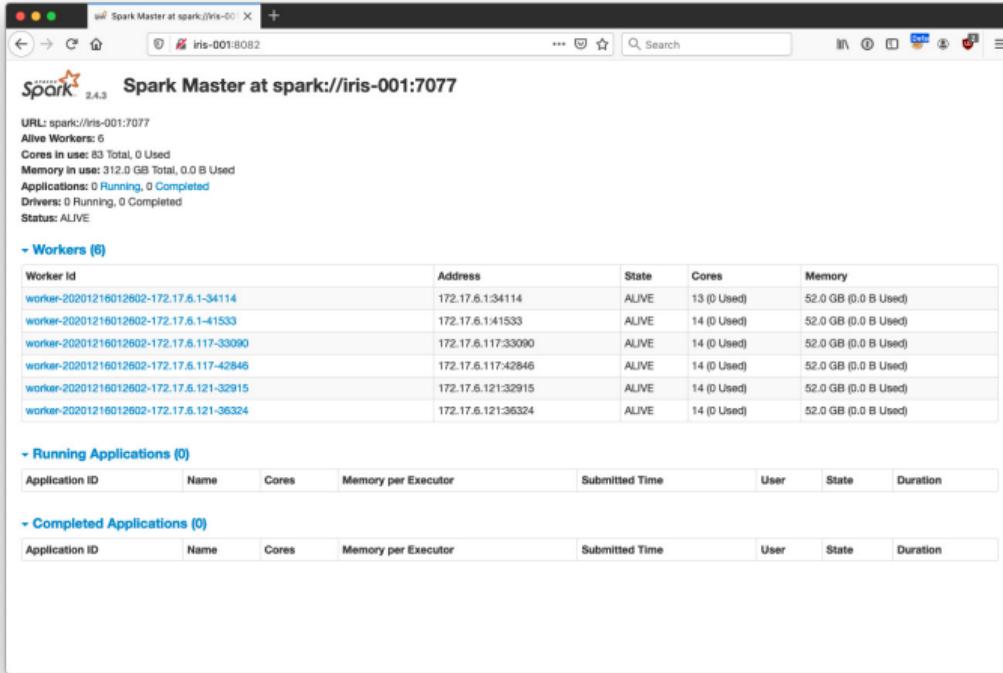
```
(laptop)$> ssh -D 1080 -C iris-cluster
(access)$> cd ~/git/github.com/ULHPC/tutorials/bigdata
# Eventually - adapt the reservation name accordingly
(access)$> srun [--reservation=<name>] -N3 --ntasks-per-node 2 -c 14 --exclusive --pty bash
$> source settings/2019b
$> module load devel/Spark
```

## Multi-node Spark Cluster Deployment

```
# Deploy an interactive Spark cluster **ACROSS** all reserved nodes
$>./scripts/launcher.Spark.sh -i
===== Spark Master =====
url: spark://iris-001:7077
Web UI: http://iris-001:8082
===== 6 Spark Workers =====
export SPARK_HOME=$EBROOTSPARK
export MASTER_URL=spark://iris-001:7077
export SPARK_DAEMON_MEMORY=4096m
export SPARK_WORKER_CORES=14
export SPARK_WORKER_MEMORY=53248m
export SPARK_EXECUTOR_MEMORY=53248m
# [...] *** Interactive mode ***
```

- Connect **transparently** to the master Web UI via the SOCK5 proxys  
→ thus **using your browser on your laptop at home**

# Multi-node Spark Cluster Deployment



The screenshot shows the Spark Master UI at `spark://iris-001:7077`. The UI displays the following information:

- Spark Version:** 2.4.3
- URL:** `spark://iris-001:7077`
- Alive Workers:** 6
- Cores in use:** 83 Total, 0 Used
- Memory in use:** 312.0 GB Total, 0.0 B Used
- Applications:** 0 Running, 0 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

**Workers (6)**

Worker Id	Address	State	Cores	Memory
worker-20201216012602-172.17.6.1-34114	172.17.6.1:34114	ALIVE	13 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.1-41533	172.17.6.1:41533	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.117-33090	172.17.6.117:33090	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.117-42846	172.17.6.117:42846	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.121-32915	172.17.6.121:32915	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.121-36324	172.17.6.121:36324	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)

**Running Applications (0)**

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

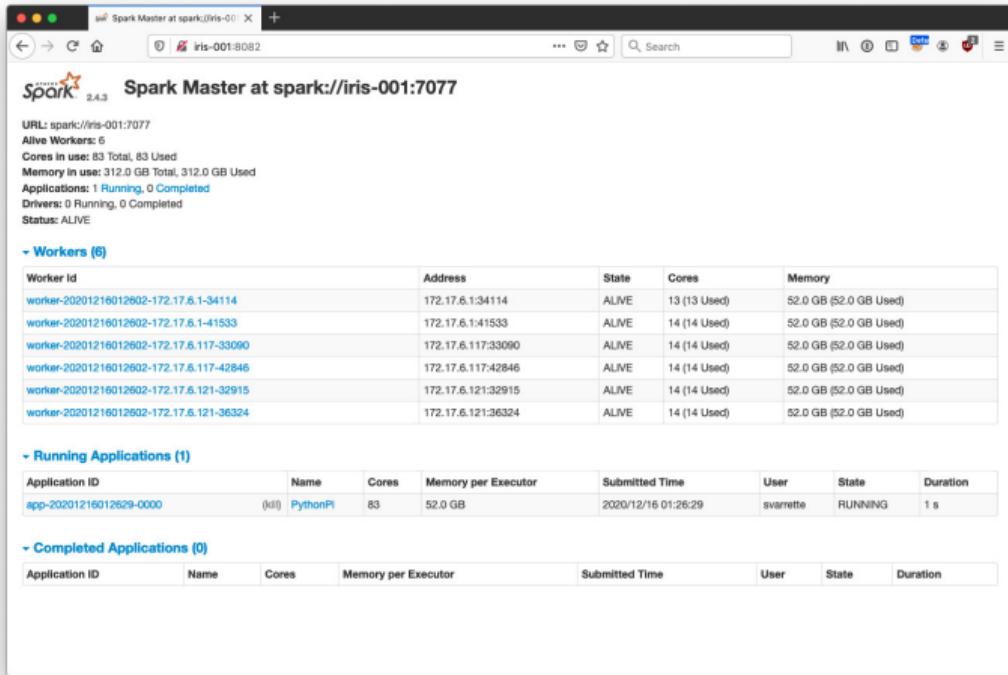
**Completed Applications (0)**

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

## Multi-node Spark Cluster Deployment

```
# Submit your Spark job
spark-submit \
    --master spark://$(scontrol show hostname $SLURM_NODELIST | head -n 1):7077 \
    --conf spark.driver.memory=${SPARK_DAEMON_MEMORY} \
    --conf spark.executor.memory=${SPARK_EXECUTOR_MEMORY} \
    --conf spark.python.worker.memory=${SPARK_WORKER_MEMORY} \
    ${SPARK_HOME}/examples/src/main/python/pi.py 1000
```

# Multi-node Spark Cluster Deployment



The screenshot shows the Spark Master UI at `spark://iris-001:7077`. The UI displays the following information:

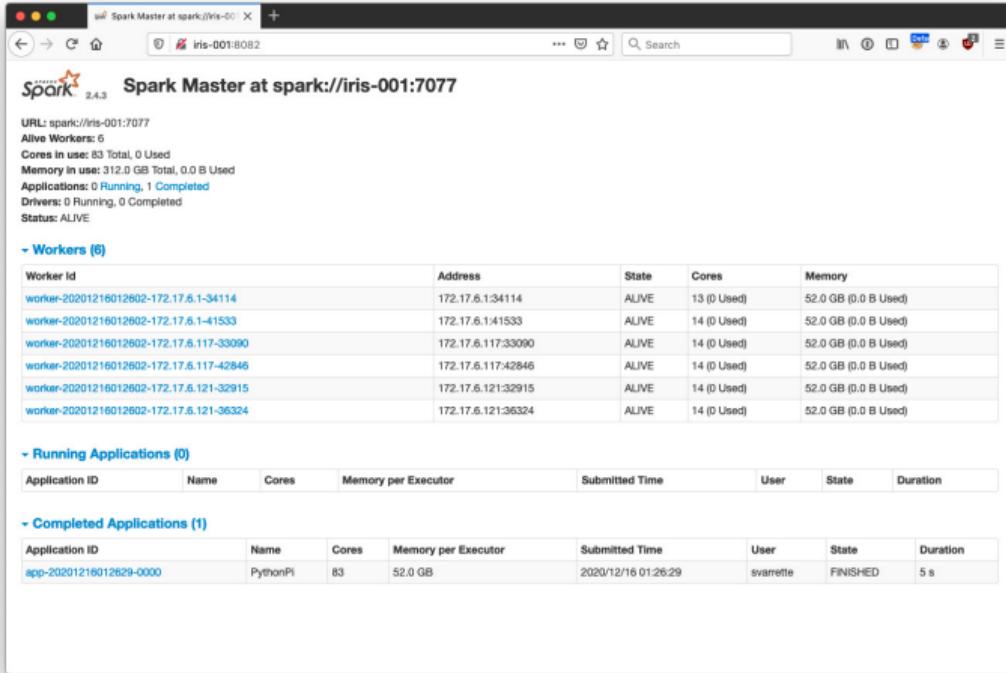
- Cluster Summary:**
  - URL: `spark://iris-001:7077`
  - Alive Workers: 6
  - Cores in use: 83 Total, 83 Used
  - Memory in use: 312.0 GB Total, 312.0 GB Used
  - Applications: 1 Running, 0 Completed
  - Drivers: 0 Running, 0 Completed
  - Status: ALIVE
- Workers (6):**

Worker Id	Address	State	Cores	Memory
worker-20201216012602-172.17.6.1-34114	172.17.6.1:34114	ALIVE	13 (13 Used)	52.0 GB (52.0 GB Used)
worker-20201216012602-172.17.6.1-41533	172.17.6.1:41533	ALIVE	14 (14 Used)	52.0 GB (52.0 GB Used)
worker-20201216012602-172.17.6.117-33090	172.17.6.117:33090	ALIVE	14 (14 Used)	52.0 GB (52.0 GB Used)
worker-20201216012602-172.17.6.117-42846	172.17.6.117:42846	ALIVE	14 (14 Used)	52.0 GB (52.0 GB Used)
worker-20201216012602-172.17.6.121-32915	172.17.6.121:32915	ALIVE	14 (14 Used)	52.0 GB (52.0 GB Used)
worker-20201216012602-172.17.6.121-36324	172.17.6.121:36324	ALIVE	14 (14 Used)	52.0 GB (52.0 GB Used)
- Running Applications (1):**

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20201216012629-0000	(kill) PythonPi	83	52.0 GB	2020/12/16 01:26:29	svarrette	RUNNING	1 s
- Completed Applications (0):**

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration

# Multi-node Spark Cluster Deployment



The screenshot shows the Spark Master UI at `spark://iris-001:7077`. The UI displays cluster statistics and lists workers and completed applications.

**Cluster Statistics:**

- URL: `spark://iris-001:7077`
- Alive Workers: 6
- Cores in use: 83 Total, 0 Used
- Memory in use: 312.0 GB Total, 0.0 B Used
- Applications: 0 Running, 1 Completed
- Drivers: 0 Running, 0 Completed
- Status: ALIVE

**Workers (6)**

Worker Id	Address	State	Cores	Memory
worker-20201216012602-172.17.6.1-34114	172.17.6.1:34114	ALIVE	13 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.1-41533	172.17.6.1:41533	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.117-33090	172.17.6.117:33090	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.117-42846	172.17.6.117:42846	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.121-32915	172.17.6.121:32915	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)
worker-20201216012602-172.17.6.121-36324	172.17.6.121:36324	ALIVE	14 (0 Used)	52.0 GB (0.0 B Used)

**Running Applications (0)**

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20201216012629-0000	PythonPi	83	52.0 GB	2020/12/16 01:26:29	svarrette	FINISHED	5 s

**Completed Applications (1)**

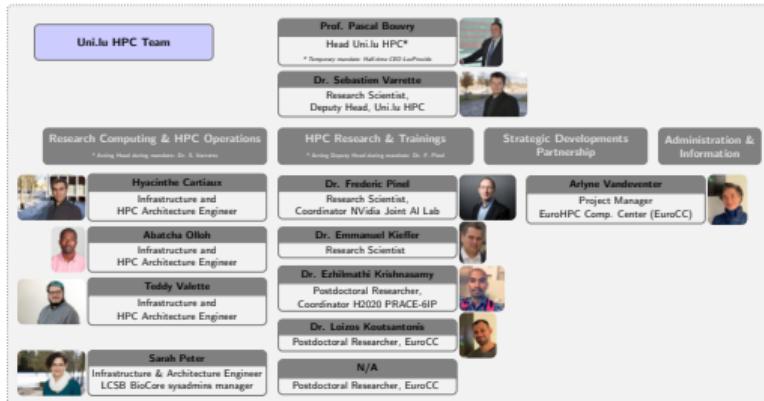
Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20201216012629-0000	PythonPi	83	52.0 GB	2020/12/16 01:26:29	svarrette	FINISHED	5 s

Thank you for your attention...



# Questions?

## High Performance Computing @ Uni.lu



The organizational chart shows the structure of the Uni.lu HPC Team. At the top is Prof. Stephane Pallage, Rector. Below him is Prof. Pascal Boutry, Head Uni.lu HPC\*. Dr. Sébastien Varrette, Research Scientist and Deputy Head, Uni.lu HPC, is also listed. The team is divided into several groups:

- Uni.lu HPC Team**: Contains Prof. Stephane Pallage, Rector.
- Research Computing & HPC Operations**: Contains Dr. Sébastien Varrette, Research Scientist and Deputy Head, Uni.lu HPC; Dr. Hyacinthe Cartiaux, Infrastructure and HPC Architecture Engineer; Dr. Abatcha Ollah, Infrastructure and HPC Architecture Engineer; Dr. Teddy Vallette, Infrastructure and HPC Architecture Engineer; and Sarah Peter, Infrastructure & Architecture Engineer LCSB BioCore sysadmin manager.
- HPC Research & Trainings**: Contains Dr. Frederic Pimed, Research Scientist, Coordinator NVIDIA Joint AI Lab; Dr. Emmanuel Kieller, Research Scientist; Dr. Ezhilmati Krishnamony, Postdoctoral Researcher, Coordinator H2020 PRACE-6IP; and Dr. Loïos Koutsantonis, Postdoctoral Researcher, EuroCC.
- Strategic Developments Partnership**: Contains Arlyne Vandeventer, Project Manager EuroHPC Comp. Center (EuroCC).
- Administration & Information**: Contains N/A, Postdoctoral Researcher, EuroCC.

University of Luxembourg, Belval Campus  
Maison du Nombre, 4th floor  
2, avenue de l'Université  
L-4365 Esch-sur-Alzette  
mail: [hpc@uni.lu](mailto:hpc@uni.lu)

### 1 Practical Session Objectives

### 2 Interlude: Installing [missing] software with Easybuild

### 3 Python and Data Science

### 4 Big Data Analytics with Hadoop & Spark

Apache Hadoop  
Apache Spark

<https://hpc.uni.lu>