INTRODUCTION TO HADOOP AND MAPREDUCE

Giovanna Roda & Liana Akobian

Big Data analysis with Hadoop and RHadoop, October 19th, 2022

Outline

- Introduction
- What is Big Data?
- The Hadoop distributed computing architecture
- HDFS hands-on exercises
- The YARN resource manager
- MapReduce
- MapReduce hands-on
- Concluding remarks

Timetable

October 19th	
13:00–13:15	**Introduction to the course**
13:15–14:00	**Hadoop for distributed computing** Big Data and the Hadoop architecture
14:00–14:15	*break*
14:15–15:00	**Hadoop Distributed File System (HDFS)** Basic concepts and hands-on: manage data on HDFS
15:00–15:15	*break*
15:15–16:00	**MapReduce (MR)** MR computing model: split/map/sort/shuffle/reduce
16:00–16:15	*break*
16:15–17:00	**Hands-on exercises with MapReduce**

EuroCC

This course arises from a cooperation (*twinning*) between the Austrian and Slovenian EuroCC national competence centers.

National Competence Centres are the central points of contact for accessing computing resources as well as training and consultancy in the areas of:

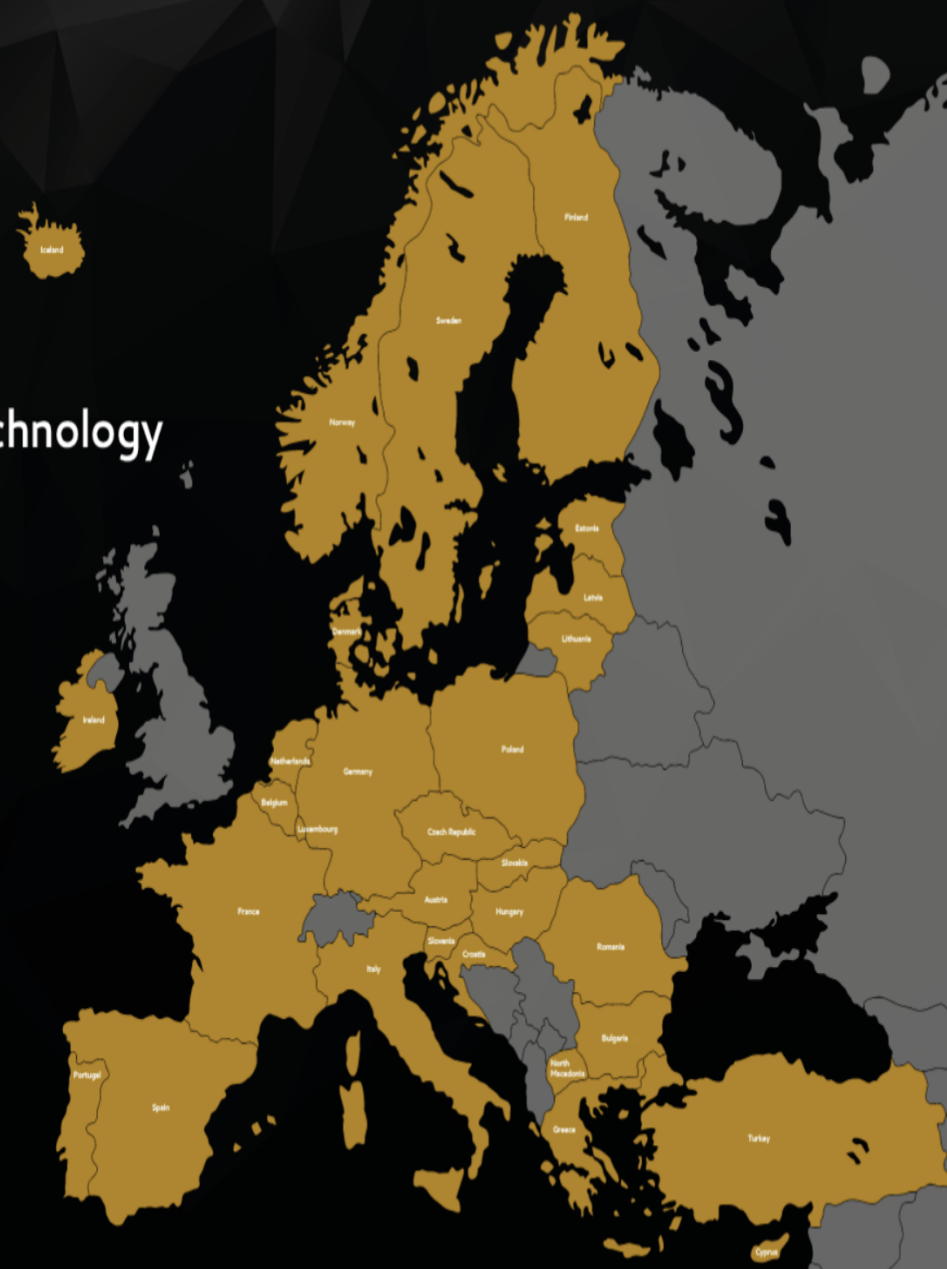
- High-Performance Computing (HPC)
- High-Performance Data Analytics (HPDA)
- Artificial Intelligence (AI)

EuroCC national competence centers were created in 2020 in the framework of EuroHPC (https://www.eurocc-access.eu/about-us/the-projects/)

EuroHPC Joint Undertaking – Support for HPC users based in the EU

→ AIM: Establishment of a European HPC ecosystem

- Financial support
  - Provision of research and innovation grants
  - European "low-energy processor" initiative
  - Exascale supercomputers featuring European technology
  - Developing a world-class supercomputing infrastructure in Europe (procurement and deployment of three pre-exascale and five petascale-systems)
- Establishment of national Competence Centres
  → EuroCC
- Establishment of an academic HPC Master programme



VSC-4

The Vienna Scientific Cluster (VSC) is a consortium of Austrian Universities providing high-performance computing resources.

VSC-4 is VSC's current flagship supercomputer. Technical specifications:

- 37920 *cores*[1]
- theoretical peak performance: 3.76 PFlop/s (*petaflops*[2] per second)

For more information see: https://vsc.ac.at/

[1] a core is a processing unit that can perform computational tasks independently

[2] a petaflop is a unit of computing speed equal to one thousand million million (10^{15}) floating-point operations per second

Introduction speakers

The trainers for today's session are:

- Liana Akobian
- Giovanna Roda

Our experience with Hadoop arises from configuring and mantaining Hadoop clusters at TU Wien's (Technische Universität Wien) dataLAB.

Thanks

Thanks to:

Dieter Kvasnicka (dataLAB and VSC), Claudia Blaas-Schenner (head of training and education at VSC and EuroCC Austria), Katrin Muck for support with the VSC Jupyterhub infrastructure, Simeon Harrison and the whole EuroCC Austria team, and last but not least Jan Zabloudil (technical support VSC).

Prerequisites for Hands-on

Prerequisites for Hands-on

Course Resources

All resources (slides, code files, ..) are available on the BitBucket repository that can be downloaded via:

```
git clone https://git@bitbucket.org/bdtrain/resources.git
```

Folder `day_1` contains all resources for today's training and folder `day_2` for tomorrow's.

Additionally, all code is on your trainee account directory.

Prerequisites for Hands-on

VSC - Cluster Connection

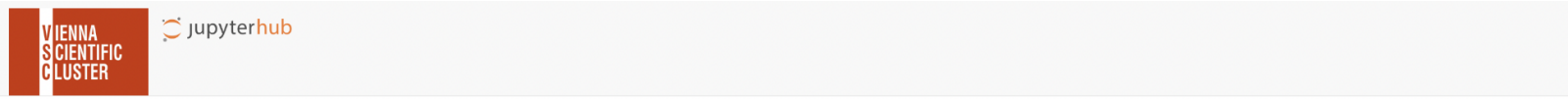
You should have received your VSC credentials via email.

You can now access the VSC Jupyterhub on

`https://jupyterhub.vsc.ac.at`

Please allow for some time for your Jupyterhub server to start.

Prerequisites for Hands-on



Prerequisites for Hands-on

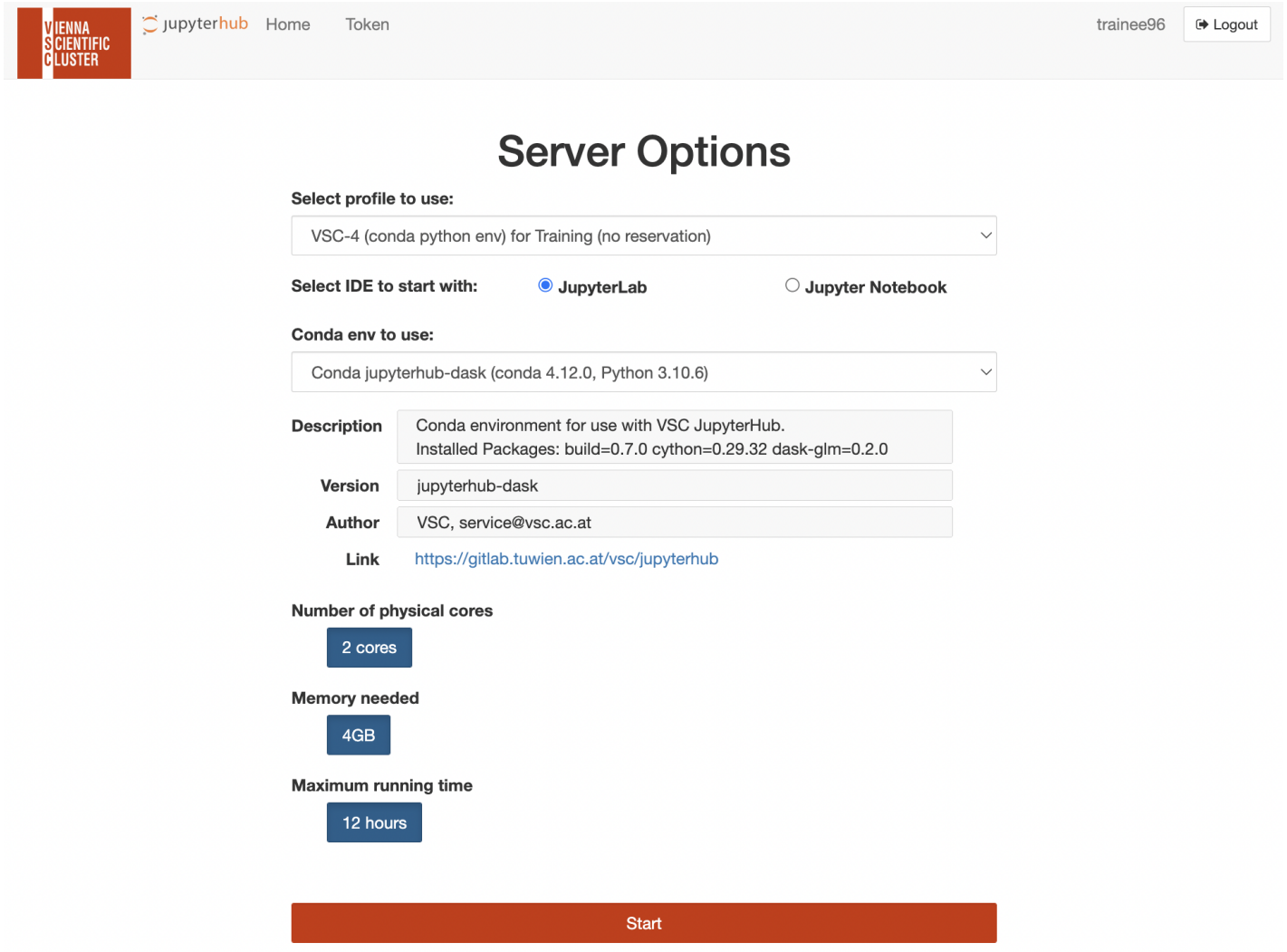
Prerequisites for Hands-on

EURO

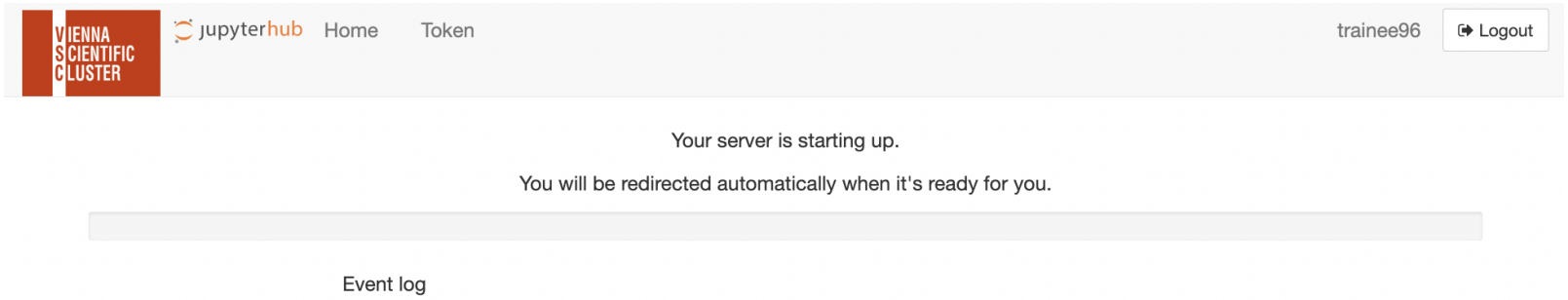
You should have received a *one-time password* (OTP) on your phone.

Photo by Diogo Brandao on Unsplash

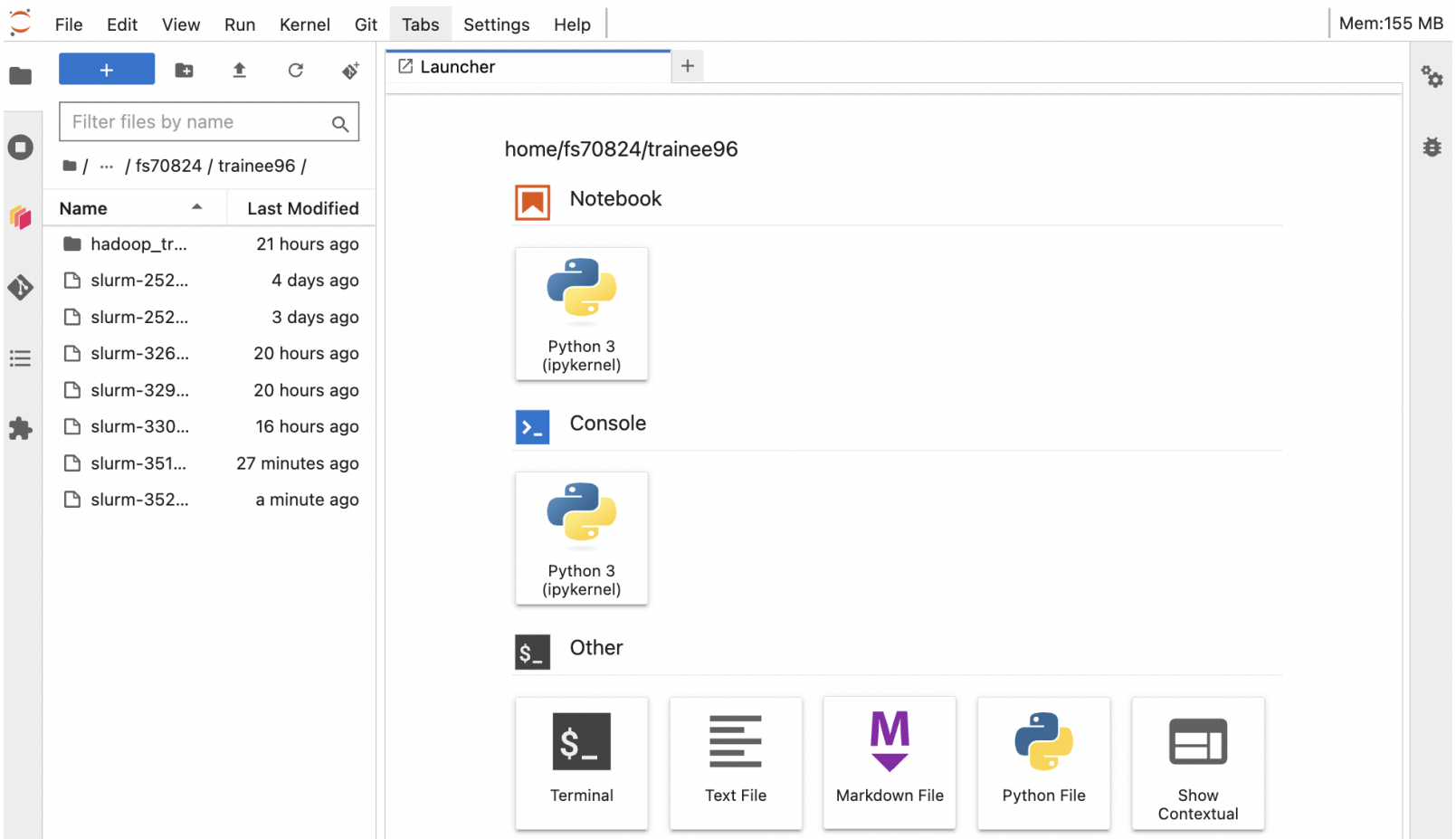
Prerequisites for Hands-on



Prerequisites for Hands-on



Prerequisites for Hands-on



Outline/next

- Introduction
- What is Big Data?
- The Hadoop distributed computing architecture
- HDFS hands-on exercises
- The YARN resource manager
- MapReduce
- MapReduce hands-on
- Concluding remarks

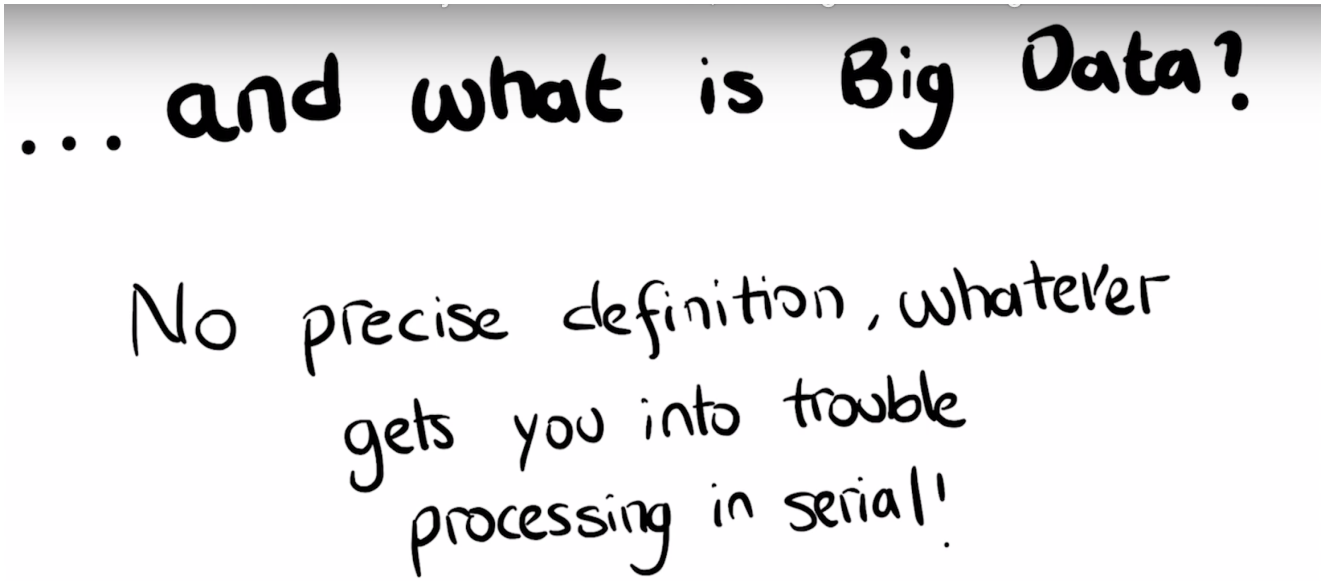
What is Big Data?

What is Big Data?

"**Big Data**" is the catch-all term for massive amounts of data as well as for frameworks and R&D initiatives aimed at working with them efficiently.

Image source: erpinnews.com

A short definition of Big Data



A nice definition from last year's PRACE Summer of HPC presentation "Convergence of HPC and Big Data".

The three V's of Big Data

Big Data is often characterized by three V's:

- **V**olume (the sheer volume of data)
- **V**elocity (rate of flow of the data and processing speed needs)
- **V**ariety (different sources and formats)

The three V's of Big Data

Data arise from disparate sources and come in many sizes and formats. Velocity refers to the speed of data generation as well as to processing speed requirements.

Volume	Velocity	Variety
MB	batch	table
GB	periodic	database
TB	near-real time	multimedia
PB	real time	unstructured
...	...	...

Reference: metric prefixes

1000000000000000000000000	10^{24}	yotta	Y	septillion
1000000000000000000000	10^{21}	zetta	Z	sextillion
1000000000000000000	10^{18}	exa	E	quintillion
1000000000000000	10^{15}	peta	P	quadrillion
1000000000000	10^{12}	tera	T	trillion
1000000000	10^{9}	giga	G	billion
1000000	10^{6}	mega	M	million
1000	10^{3}	kilo	k	thousand

Note: 1 Gigabyte (GB) is 10^9 bytes. Sometimes GB is also used to denote 1024^3 or 2^{30} bytes, which is actually one *gibibyte* (GiB).

Different processing paradigms

- *Batch processing* is when data are collected and submitted to the system in batches without human interaction. Processing is carried out at a later time depending on the availability of resources. Examples of batch processing are: monthly reporting, scientific simulations, model building.
- *Real-time processing* is when a response is guaranteed within a given time frame (seconds, milliseconds, ...). Real-time processing is required by interactive applications such as ATM transactions or computer vision.

Hadoop's MapReduce is a typical batch processing tool.

Structured vs. unstructured data

- by *structured* data one refers to highly organized data that are usually stored in relational databases or data warehouses. Structured data are easy to search but unflexible in terms of the three "V"s.
- *Unstructured* data come in mixed formats, usually require pre-processing, and are difficult to search. Structured data are usually stored in noSQL databases or in *data lakes* (these are scalable storage spaces for raw data of mixed formats).
- With *semi-structured* data one usually refers to structured data containing unstructured elements (such as free text).

Examples of structured/unstructured data

Industry	Structured data	Unstructured data
e-commerce	• products & prices • customer data • transactions	• product reviews • phone transcripts • social media mentions
banking	• financial transactions • customer data	• customer communication • regulations & compliance • financial news

Examples of structured/unstructured data

Industry	Structured data	Unstructured data
healthcare	• patient records • medical billing data • genomic data	• clinical reports • radiology imagery • clinical speech
seismology	• satellite images • seismic wave sensor data	• historic records

Forecast: Big Data in 2025

Data Phase	Astronomy	Twitter	YouTube	Genomics
Acquisition	25 zetta-bytes/year	0.5–15 billion tweets/year	500–900 million hours/year	1 zetta-bases/year
Storage	1 EB/year	1–17 PB/year	1–2 EB/year	2–40 EB/year
Analysis	In situ data reduction	Topic and sentiment mining	Limited requirements	Heterogeneous data and analysis
	Real-time processing	Metadata analysis		Variant calling, ~2 trillion central processing unit (CPU) hours
	Massive volumes			All-pairs genome alignments, ~10,000 trillion CPU hours
Distribution	Dedicated lines from antennae to server (600 TB/s)	Small units of distribution	Major component of modern user's bandwidth (10 MB/s)	Many small (10 MB/s) and fewer massive (10 TB/s) data movement

doi:10.1371/journal.pbio.1002195.t001

This table[3] shows the projected annual storage and computing needs in four domains (astronomy, social media, genomics).

[3] Stephens ZD et al. "Big Data: Astronomical or Genomical?" In: *PLoS Biol* (2015).

The V's of Big Data: additional dimensions

Three more "V"s to be considered:

- **V**eracity (quality or trustworthiness of data)
- **V**alue (economic value of the data)
- **V**ariability (change over time of any of the aforementioned characteristics)

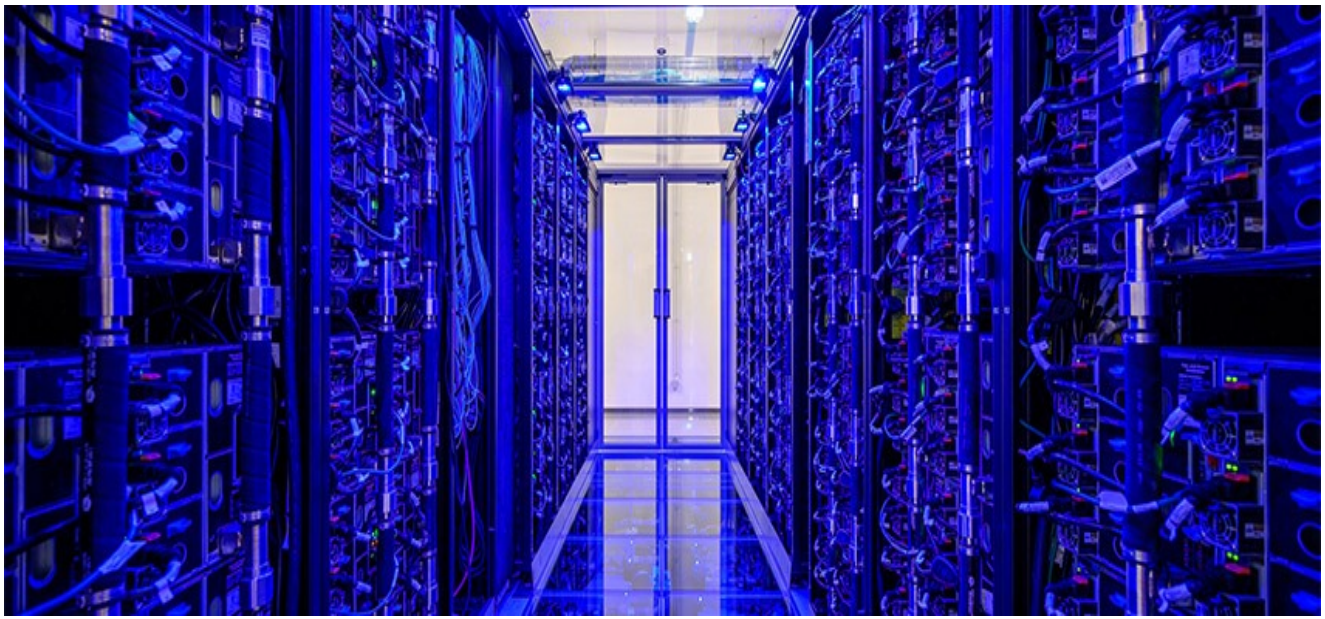
The challenges of Big Data

When working with large amounts of data you will sooner or later face one or more of these challenges:

- disk and memory space
- processing speed
- hardware faults
- network capacity and speed
- need to optimize resources use

Big Data software tools address these challenges.

Distributed computing for Big Data

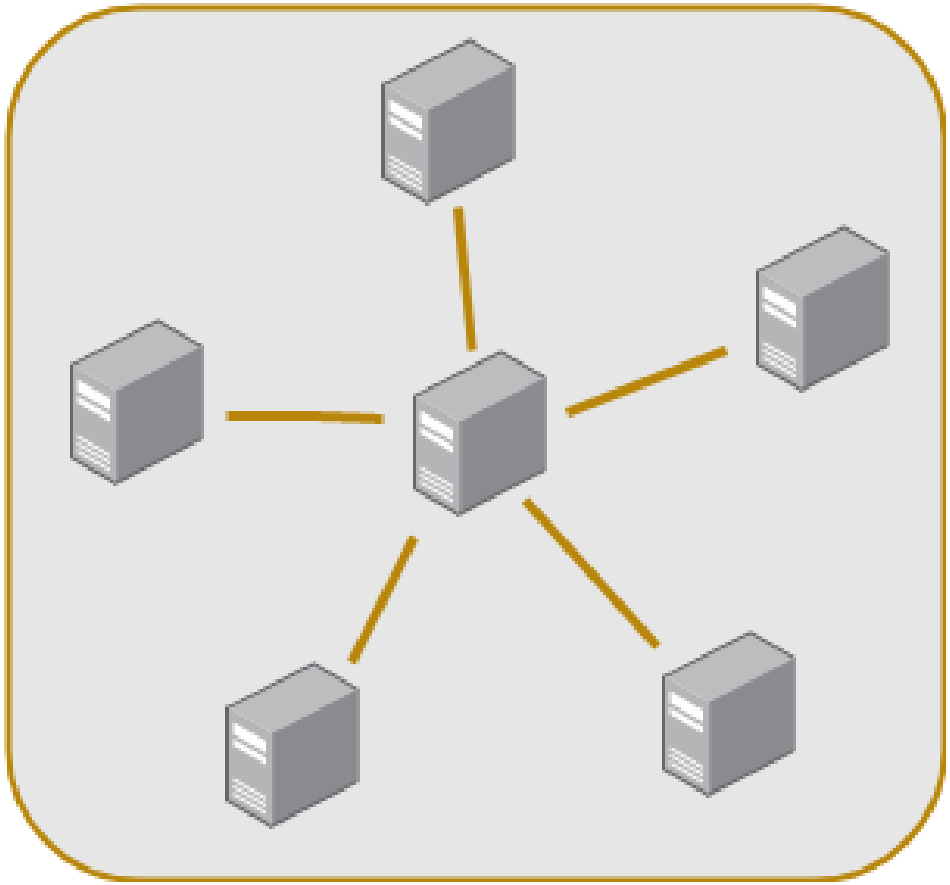


Source: VSC-4 ©Matthias Heisler

Traditional data processing tools are inadequate for large amounts of data.

Distributed computation makes it possible to work with Big Data optimizing time and available resources.

What is distributed computing?



A distributed computer system consists of several interconnected *nodes*. Nodes can be physical as well as virtual machines or containers.

When a group of nodes provides services and applications to the client as if it were a single machine, then it is also called a *cluster*.

Benefits of distributed computing

- **Performance**: supports intensive workloads by spreading tasks across nodes
- **Scalability**: new nodes can be added to increase capacity
- **Fault tolerance**: resilience in case of hardware failures

Outline/next

- Introduction
- What is Big Data?
- The Hadoop distributed computing architecture
- HDFS hands-on exercises
- The YARN resource manager
- MapReduce
- MapReduce hands-on
- Concluding remarks

The Hadoop distributed computing architecture

What is Hadoop?

Hadoop is a framework for running jobs on clusters of computers that provides a good abstraction of the underlying hardware and software.

What is Hadoop?

"*Stripped to its core, the tools that Hadoop provides for building distributed systems—for data storage, data analysis, and coordination—are simple. If there's a common theme, it is about raising the level of abstraction—to create building blocks for programmers who just happen to have lots of data to store, or lots of data to analyze, or lots of machines to coordinate, and who don't have the time, the skill, or the inclination to become distributed systems experts to build the infrastructure to handle it.*[4]"

[4] White T. *Hadoop: The Definitive Guide*. O'Reilly, 2015.

Hadoop: some facts

- open-source project of the Apache Software Foundation
- its core components are implemented in Java
- initially released in 2006. Last stable version is 3.3.4 from August 2022
- originally inspired by Google's MapReduce[5] and the proprietary GFS (Google File System)

[5] J. Dean and S. Ghemawat. "MapReduce: Simplified data processing on large clusters." In: *Proceedings of Operating Systems Design and Implementation (OSDI)*. 2004. url: https://www.usenix.org/legacy/publications/library/proceedings/osdi04/tech/full_papers/dean/dean.pdf.

Hadoop's features

Hadoop addresses many of the challenges of Big Data by providing:

- scalability
- fault tolerance
- high availability
- distributed cache/data locality
- cost-effectiveness (no high-end hardware is needed)
- provides a good abstraction of the underlying hardware
- easy to learn
- data can be queried trough SQL-like endpoints (Hive, Cassandra, SparkSQL)

Mini-glossary of Hadoop's main features

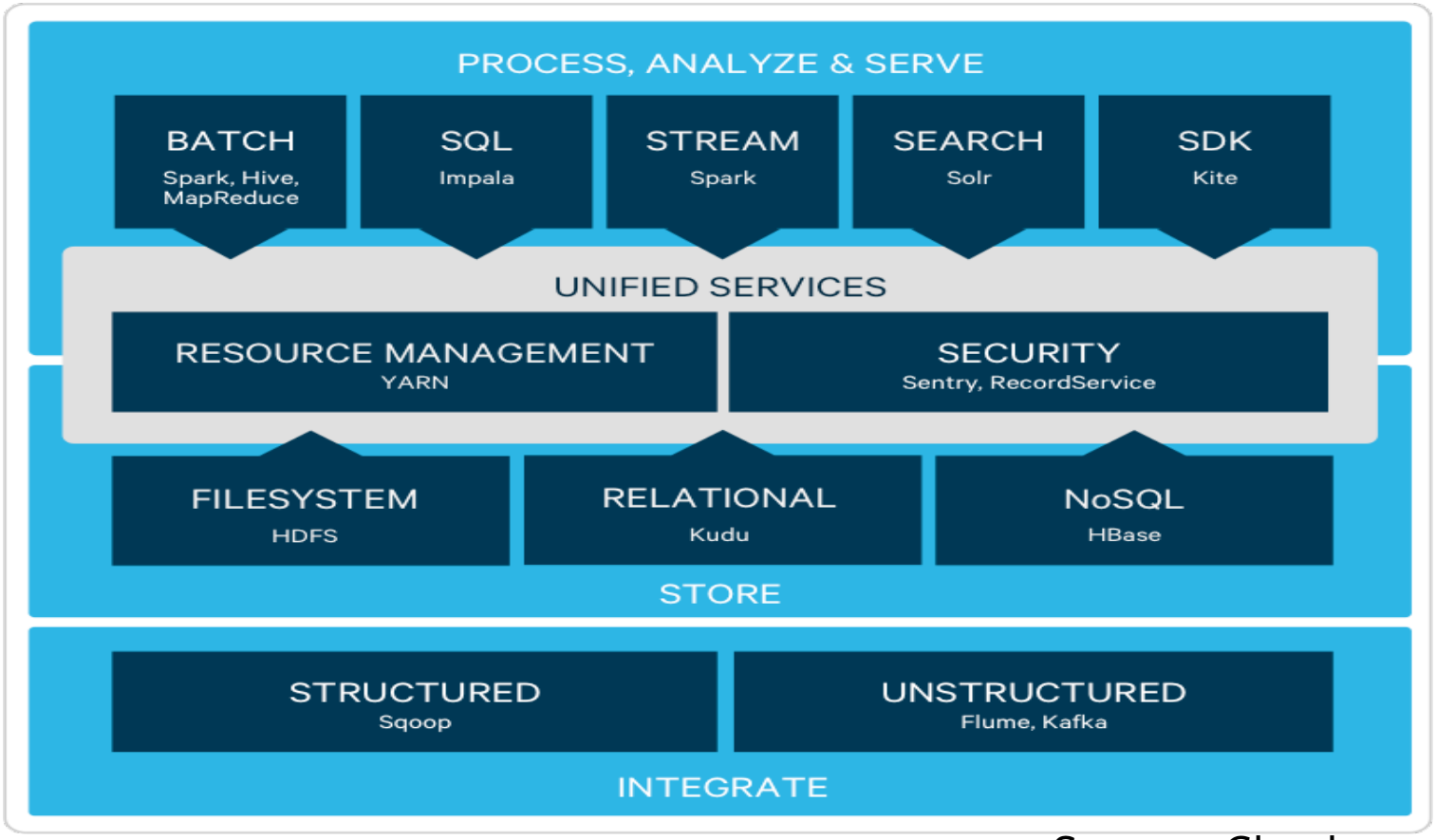
- ***fault tolerance***: the ability to withstand hardware or network failures (also: *resilience*)
- ***high availability***: this refers to the system minimizing downtimes by eliminating single points of failure
- ***data locality***: task are run on the node where data are located, in order to reduce time-consuming transfer of data

The Hadoop core

The core of Hadoop consists of:

- Hadoop common, the core libraries
- HDFS, the Hadoop Distributed File System
- MapReduce
- the YARN (Yet Another Resource Negotiator) resource manager

The Hadoop ecosystem



Source: Cloudera

There's a whole constellation of open source components for collecting, storing, and processing big data that integrate with Hadoop.

Some useful tools that integrate with Hadoop

Just to mention a few:

- **Spark** in-memory computation engine superseding MapReduce
- **Kafka** a distributed streaming system that allows to integrate multiple streams of data for real-time processing
- **Zookeeper** synchronization tool for distributed systems
- **Hbase** a noSQL database (key-value store) that runs on the Hadoop distributed filesystem
- **Hive** a distributed datawarehouse system
- **Presto** a distributed SQL query engine
- **Oozie** a workflow scheduler

All these tools are open source.

The Hadoop Distributed File System (HDFS)

HDFS stands for Hadoop Distributed File System and it takes care of partitioning data across a cluster.

In order to prevent data loss and/or task termination due to hardware failures HDFS uses either

- replication (creating multiple copies —usually 3— of the data)
- erasure coding

Data redundancy (obtained through replication or erasure coding) is the basis of Hadoop's fault tolerance.

Replication vs. Erasure Coding

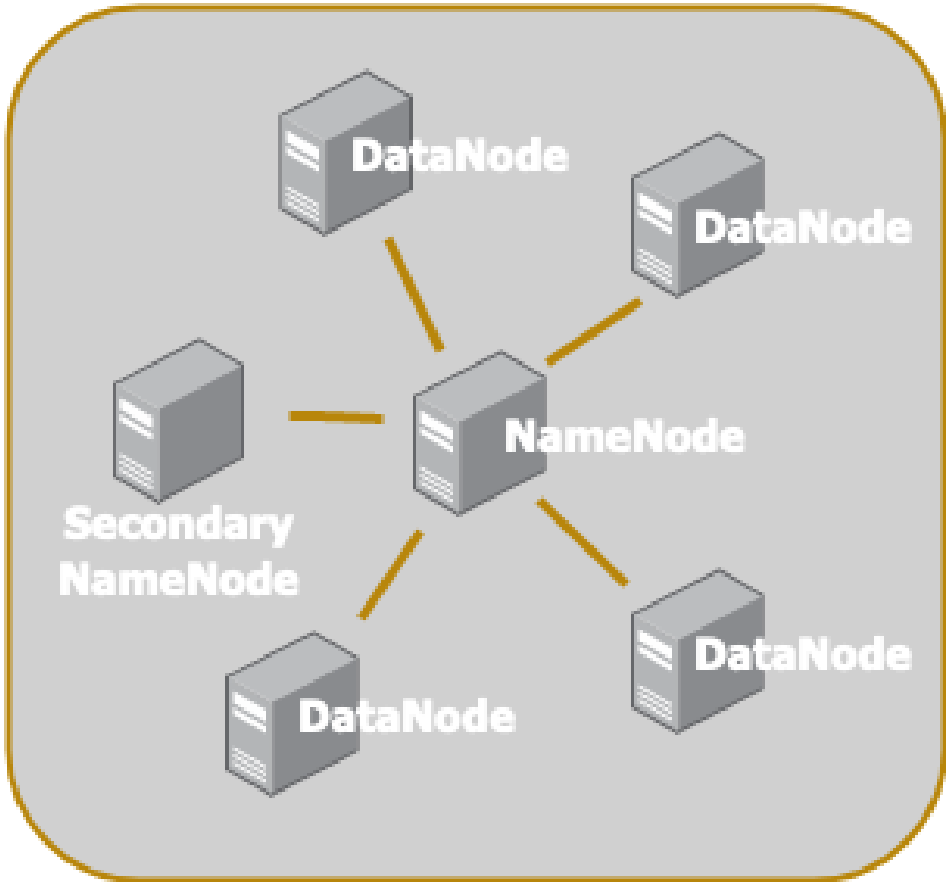
In order to provide protection against failures one introduces:

- data redundancy
- a method to recover the lost data using the redundant data

Replication is the simplest method for coding data by making n copies of the data. n-fold replication guarantees the availability of data for at most $n-1$ *failures and it has a storage overhead of 200% (this is equivalent to a storage efficiency of 33%).*

Erasure coding provides a better storage efficiency (up to to 71%) but it can be more costly than replication in terms of performance.

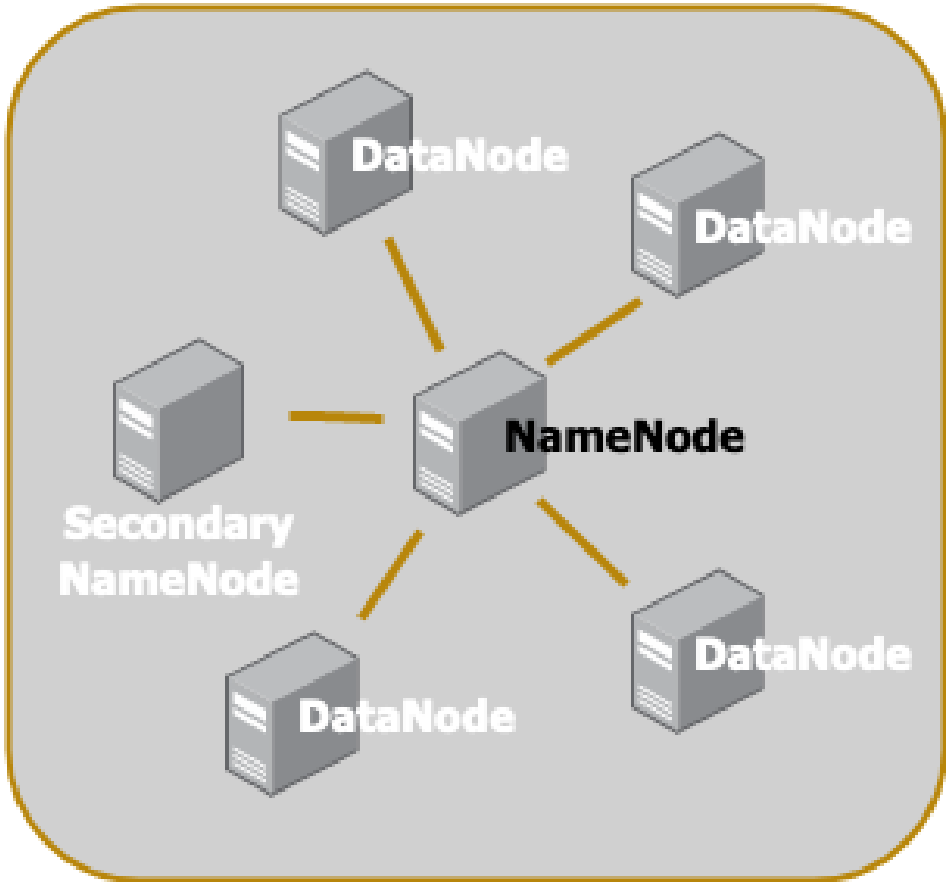
HDFS architecture



A typical Hadoop cluster installation consists of:

- a NameNode
- a secondary NameNode
- multiple DataNodes

HDFS architecture: NameNode

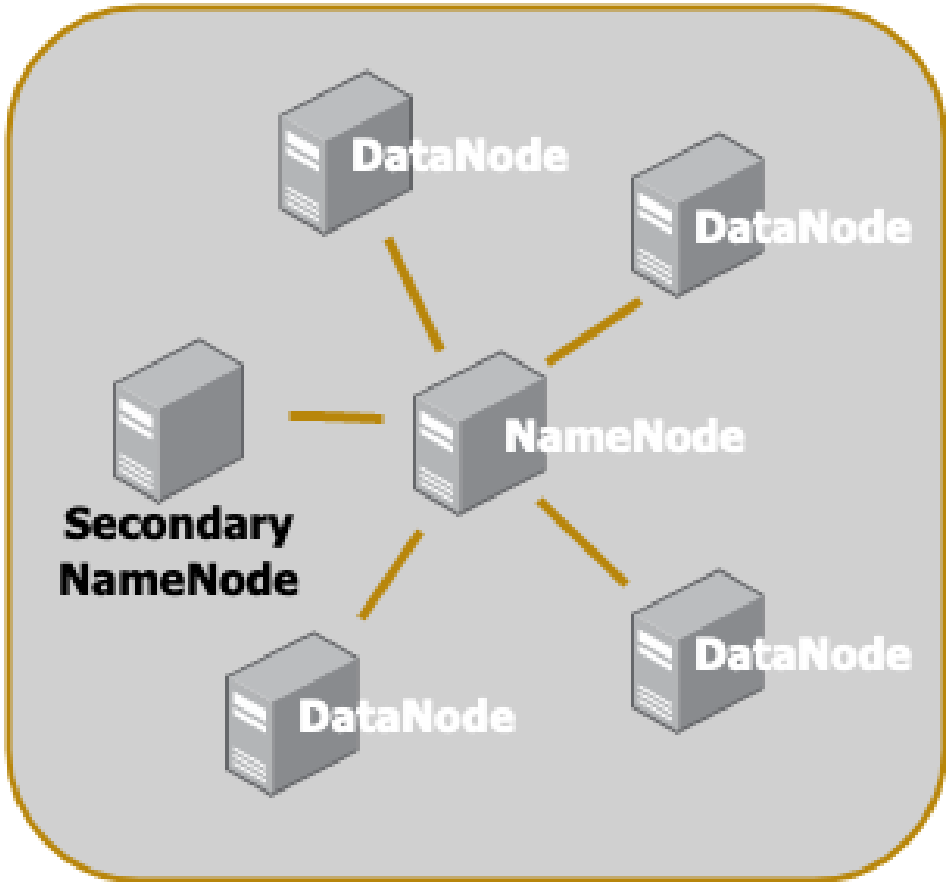


NameNode

The NameNode is the main point of access of a Hadoop cluster. It is

responsible for the bookkeeping of the data partitioned across the DataNodes, manages the whole filesystem metadata, and performs load balancing

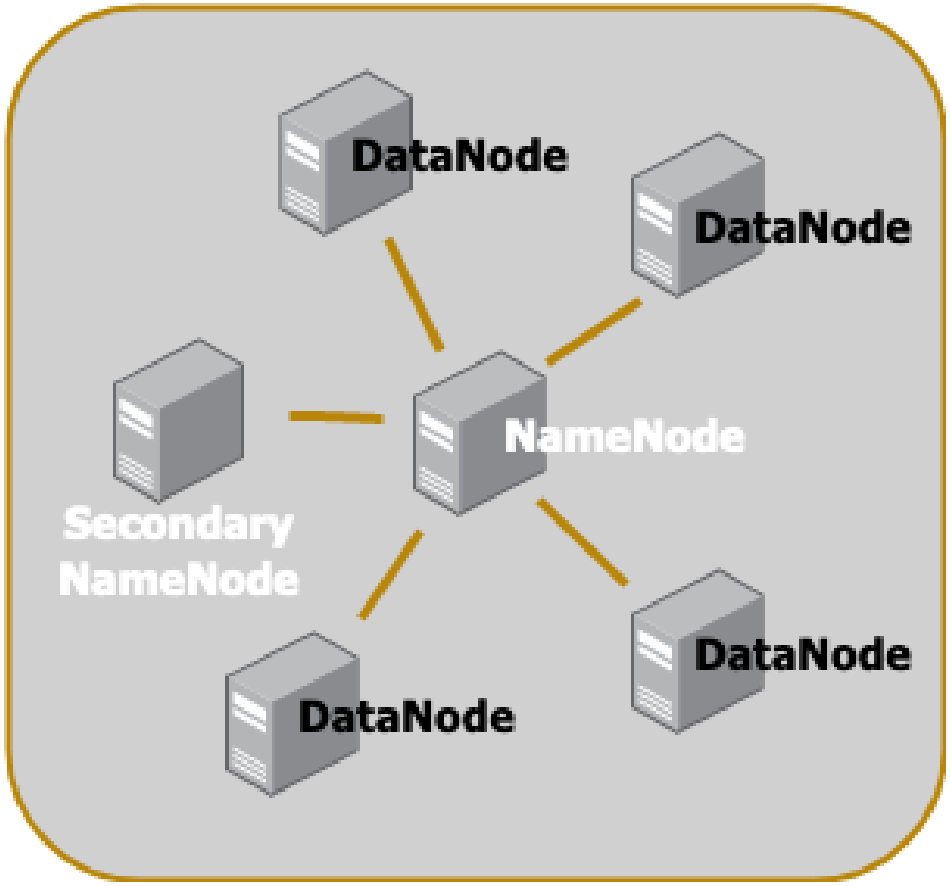
HDFS architecture: Secondary NameNode



Secondary NameNode

Keeps track of changes in the NameNode performing regular snapshots, thus allowing quick startup.
An additional *standby node* is needed to guarantee high availability (since the NameNode is a single point of failure).

HDFS architecture: DataNode



DataNode

Here is where the data is saved and the computations take place (data nodes should actually be called "data and compute nodes").

HDFS architecture: internal data representation

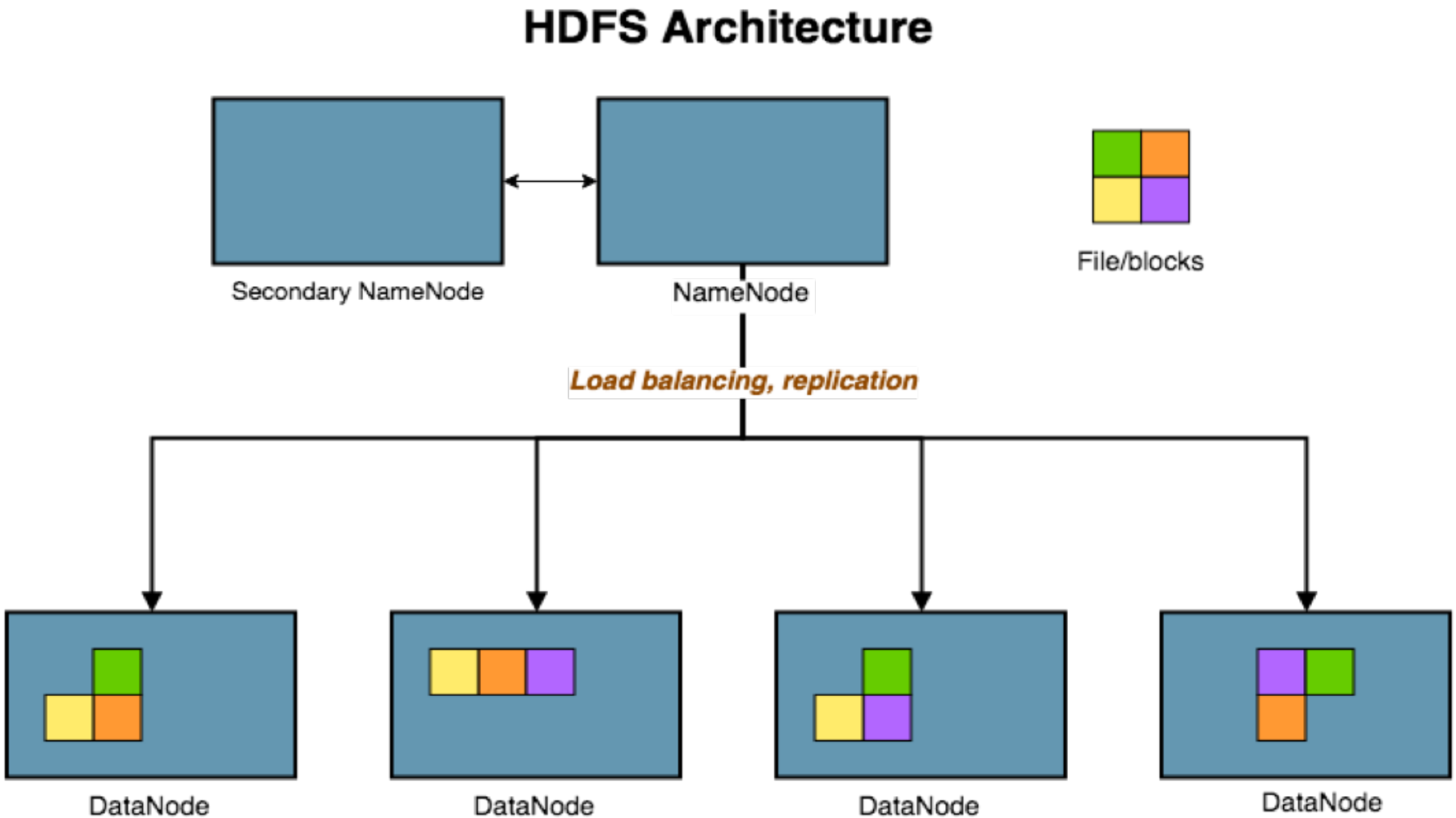
HDFS supports working with very large files.

Internally, data are split into *blocks*. One of the reason for splitting data into blocks is that in this way block objects all have the same size.

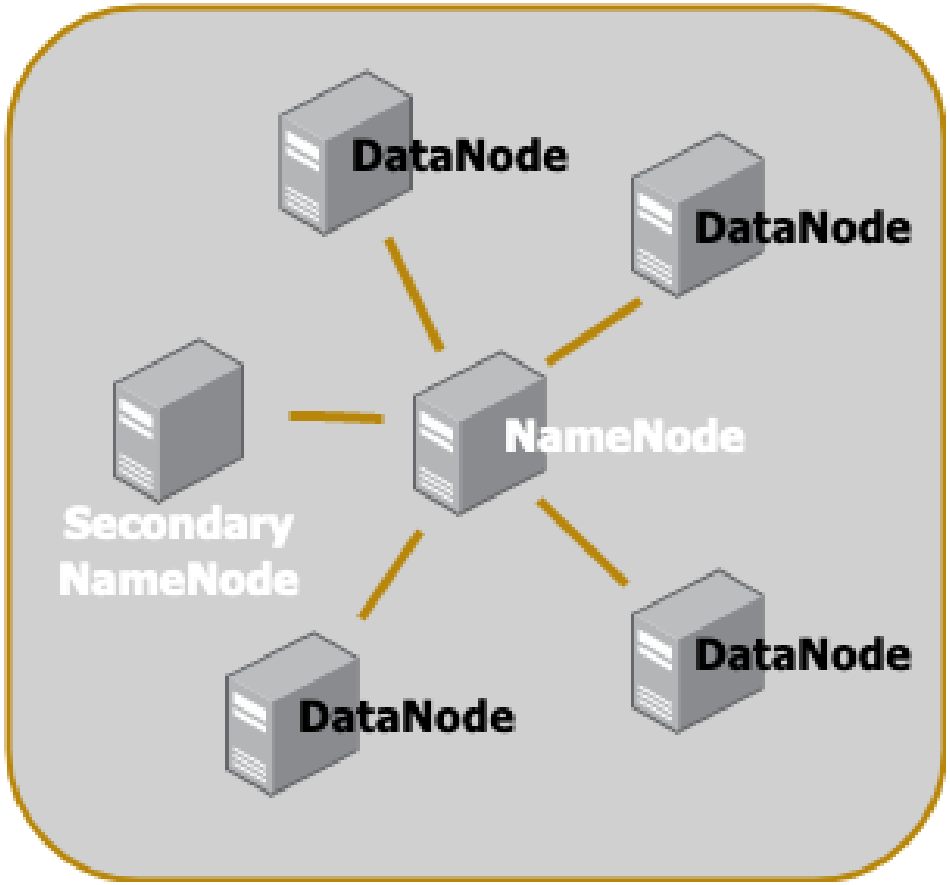
The block size in HDFS can be configured at installation time and it is by default **128MiB** (approximately **134MB**).

Note: data for Hadoop can be a single file or a directory. When giving a directory as input, all files in that directory will be processed.

HDFS architecture



Management of DataNode failures



Each DataNode sends a *heartbeat* message to the NameNode periodically.

Whenever a DataNode becomes unavailable (due to network or hardware failure), the NameNode stops sending requests to that node and creates new replicas of the blocks stored on that node.

Blocks versus partitions

In the next part of the course you will hear about *data partitioning*.

File partitions are logical divisions of the data and should not be confused with blocks, that are physical chunks of data (i.e. each block has a physical location on the hardware).

The WORM principle of HDFS

The Hadoop Distributed File System relies on a simple design principle for data known as Write Once Read Many (WORM).

"*A file once created, written, and closed need not be changed except for appends and truncates. Appending the content to the end of the files is supported but cannot be updated at arbitrary point. This assumption simplifies data coherency issues and enables high throughput data access.*[6]"

[6] Apache Software Foundation. *Hadoop*. url: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

Data immutability paradigm

The data immutability paradigm is also discussed in Chap.2 of "Big Data".[7]

What differentiates Big Data from the world of relational databases:

- *raw data*: unstructured better than structured, unnormalized better than normalized, raw data is more open to new questions
- *immutability*: human-fault tolerance, simplicity of the data model

[7] Warren J. and Marz N. *Big Data*. Manning publications, 2015.

Etymology and grammar of the word "data"

The word "data" comes from the Latin "*datum*", which means "given"—something that can't be derived from anything else. This meaning is reflected in Hadoop data immutability design.

In case you're wondering whether "data" should be considered a plural count noun (singular "datum") or a singular count noun, the answer is: both are allowed.

The correct English form is the plural one ("these data") but the singular form ("Big Data is") is also commonly used (see[8]).

[8] J. Aronson. *A Word About Evidence: 7. Data—etymology and grammar*. https://blogs.bmj.com/bmjebmspotlight/2018/07/01/a-word-about-evidence-7-data-etymology-and-grammar/.

Data biases

Whenever one works with data, one should keep in mind that data is inherently biased.

For instance, in data harvested from the web some categories of people or themes could be underrepresented due to social, cultural, economic conditions.

And if that's not enough, alone choosing what kind of data to focus on introduces bias.

A good starting point for thinking about biases is.[9]

[9] Ricardo Baeza-Yates. "Bias on the web." In: *Communications of the ACM* 61.6 (2018), pp. 54–61.

Outline/next

- Introduction
- What is Big Data?
- The Hadoop distributed computing architecture
- HDFS hands-on exercises
- The YARN resource manager
- MapReduce
- MapReduce hands-on
- Concluding remarks

HDFS hands-on exercises

Essential module commands

Module is a Python library used to manage software environments.

```
# show available Hadoop installations
module avail Hadoop
# load the default Hadoop installation
module load Hadoop
# show loaded modules
module list
# unload all modules
module purge
# show currently loaded Hadoop version
module show Hadoop
```

We are going to use these commands to load Java and Hadoop modules.

Basic HDFS filesystem commands

One can regard HDFS as a regular file system, in fact many HDFS shell commands are inherited from the corresponding bash commands.

To run a command on an Hadoop filesystem use the prefix `hdfs dfs`, for instance use:

```
hdfs dfs -mkdir myDir
```

to create a new directory `myDir` on HDFS.

Note: One can use interchangeably `hadoop` or `hdfs dfs` when working on a HDFS file system. The command `hadoop` is more generic because it can be used not only on HDFS but also on other file systems that Hadoop supports (such as Local FS, WebHDFS, S3 FS, and others).

Basic HDFS filesystem commands

Basic HDFS filesystem commands that also exist in bash

`hdfs dfs -mkdir`	create a directory
`hdfs dfs -ls`	list files
`hdfs dfs -cp`	copy files
`hdfs dfs -cat`	print files
`hdfs dfs -tail`	output last part of a file
`hdfs dfs -rm`	remove files

Basic HDFS filesystem commands

Here's three basic commands that are specific to HDFS.

`hdfs dfs -put`	Copy single src, or multiple srcs from local file system to the destination file system
`hdfs dfs -get`	Copy files to the local file system
`hdfs dfs -usage`	get help on hadoop fs

Basic HDFS filesystem commands

To get more help on a specific `hdfs` command use: `hdfs -help <command>`

```
$ hdfs dfs -help tail
# -tail [-f] <file> :
#   Show the last 1KB of the file.

#   -f  Shows appended data as the file grows.
```

Some things to try

```
# create a new directory called "input" on HDFS
hdfs dfs -mkdir input
# copy local file wiki_1k_lines to input on HDFS
hdfs dfs -put wiki_1k_lines input/
# list contents of directory ("-h" = human)
hdfs dfs -ls -h input
# disk usage
hdfs dfs -du -h input
# get help on "du" command
hdfs dfs -help du
# remove directory
hdfs dfs -rm -r input
```

Some things to try

What is the size of the file `wiki_1k_lines`? What is its disk usage?

```
# show the size of wiki_1k_lines on the regular filesystem
ls -lh wiki_1k_lines
# show the size of wiki_1k_lines on HDFS
hdfs dfs -put wiki_1k_lines
hdfs dfs -ls -h wiki_1k_lines

# disk usage of wiki_1k_lines on the regular filesystem
du -h wiki_1k_lines
# disk usage of wiki_1k_lines on HDFS
hdfs dfs -du -h wiki_1k_lines
```

Disk usage on HDFS

The command `hdfs dfs -help du` will tell you that the output is of the form:

`size disk space consumed filename`.

You'll notice that the space on disk is larger than the file size (38.6MB versus 19.3MB):

```
hdfs dfs -du -h wiki_1k_lines
# 19.3 M  38.6 M  wiki_1k_lines
```

This is due to replication. You can check the replication factor using:

```
hdfs dfs -stat 'Block size: %o Blocks: %b Replication: %r'
    input/wiki_1k_lines
# Block size: 134217728 Blocks: 20250760 Replication: 2
```

Disk usage on HDFS

From the previous output:

```
Block size: 134217728 Blocks: 20250760 Replication: 2
```

we can see that the HDFS filesystem currently supports a replication factor of 2.

Note that the Hadoop block size is defined in terms of *mebibytes*, in fact 134217728 bytes corresponds to 128MiB and 134MB. One MiB is larger than a MB since one MiB is $1024^2 = 2^{20}$ bytes, while one MB is 10^6 bytes.

Where to find the notebooks

For this part of the training we will work with the Jupyter notebooks in the directory:

`~/hadoop_training/HDFS`

Outline/next

- Introduction
- What is Big Data?
- The Hadoop distributed computing architecture
- HDFS hands-on exercises
- The YARN resource manager
- MapReduce
- MapReduce hands-on
- Concluding remarks

The YARN resource manager

YARN: Yet Another Resource Negotiator

Hadoop jobs are usually managed by YARN (acronym for Yet Another Resource Negotiator), that is responsible for allocating resources and managing job scheduling. Basic resource types are:

- memory (memory-mb)
- virtual cores (vcores)

YARN supports an extensible resource model that allows to define any countable resource. A countable resource is a resource that is consumed while a container is running, but is released afterwards. Such a resource can be for instance:

- GPU (gpu)

YARN architecture

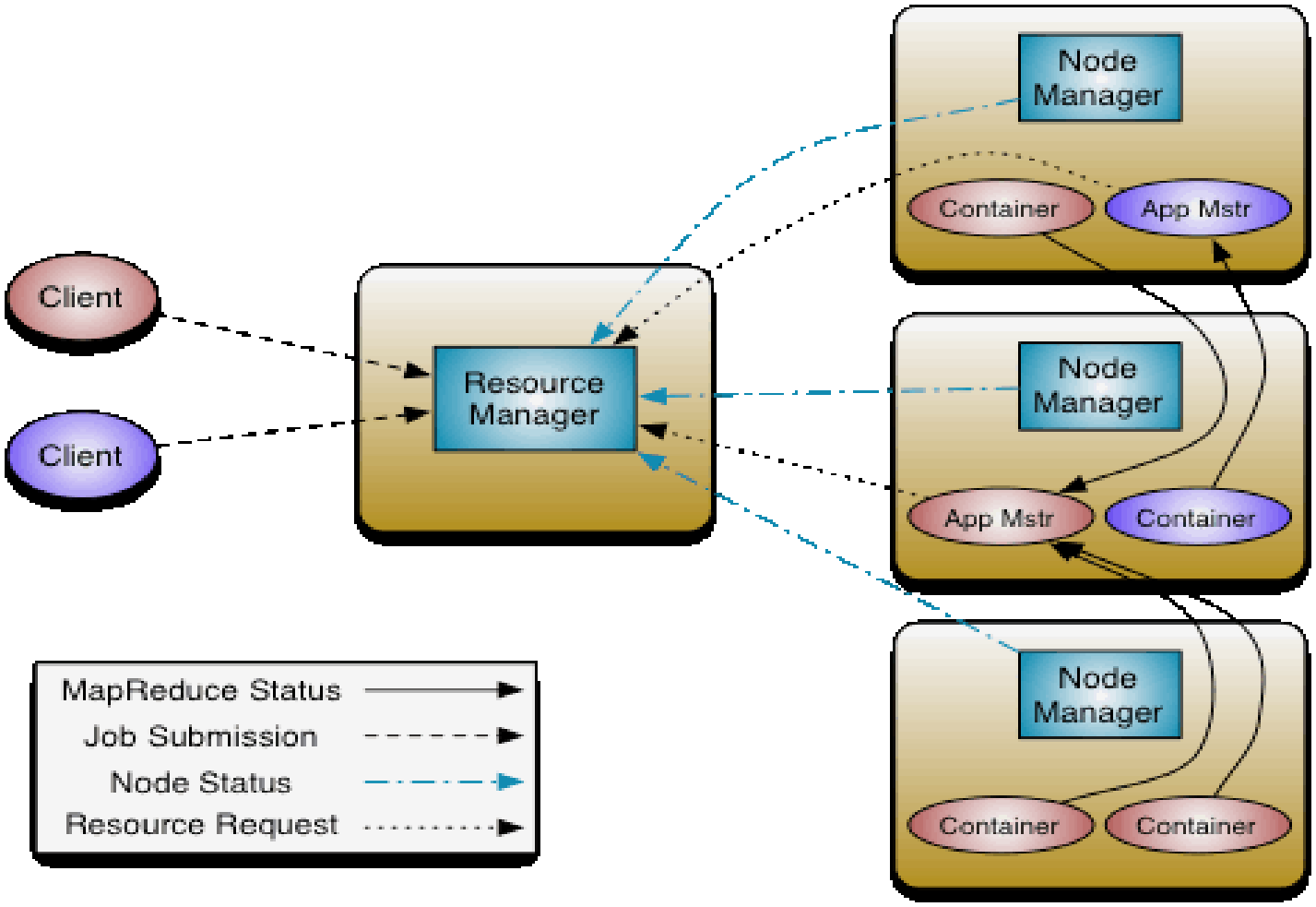


Image source: Apache Software Foundation

YARN architecture

Each job submitted to the Yarn is assigned:

- a ***container***: this is an abstract entity which incorporates resources such as memory, cpu, disk, network etc. Container resources are allocated by YARN's *Scheduler*.
- an ***ApplicationMaster*** service assigned by the Application Manager for monitoring the progress of the job, restarting tasks if needed

YARN architecture

The main idea of Yarn is to have two distinct daemons for job monitoring and scheduling, one *global* and one *local* for each application:

- the **Resource Manager** is the global job manager, consisting of:
  - **Scheduler**: allocates resources across all applications
  - **Applications Manager**: accepts job submissions, restart Application Masters on failure
- an **Application Master** is the local application manager, responsible for negotiating resources, monitoring status of the job, restarting failed tasks

Dynamic resource pools

Sharing computing resources fairly can be a big issue in multi-user environments.

YARN supports *dynamic resource pools* for scheduling applications.

A resource pool is a given configuration of resources to which a group of users is granted access. Whenever a group is not active, the resources are *preempted* and granted to other groups.

Groups are assigned a priority and resources are shared among groups according to these priority values.

Additionally, resource configurations can be scheduled for specific intervals of time.

Outline/next

- Introduction
- What is Big Data?
- The Hadoop distributed computing architecture
- HDFS hands-on exercises
- The YARN resource manager
- MapReduce
- MapReduce hands-on
- Concluding remarks

MapReduce

MapReduce: Idea

The MapReduce paradigm is inspired by the computing model commonly used in functional programming.

Applying the same function independently to items in a dataset either to transform (*map*) or collate (*reduce*) them into new values, works well in a distributed environment.

MapReduce: Idea

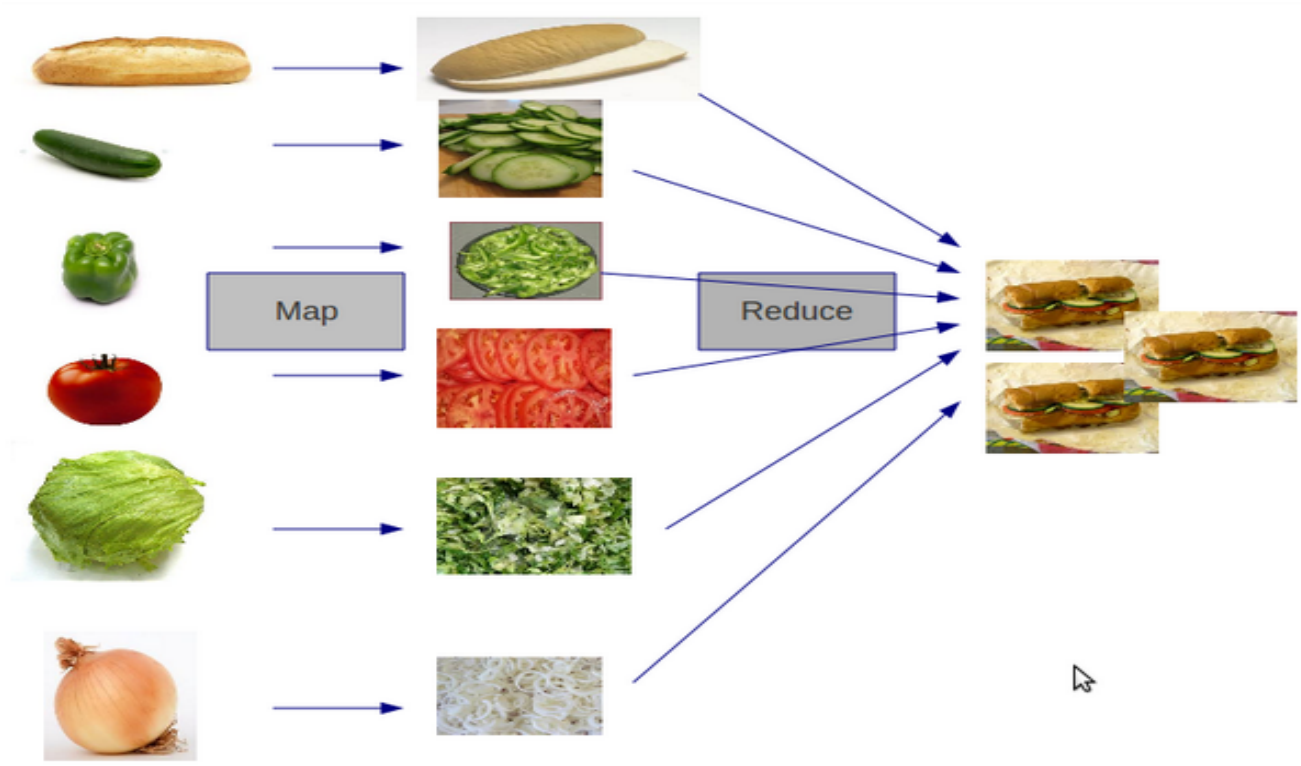


Image source: Stack Overflow

The origins of MapReduce

The 2004 paper "*MapReduce: Simplified Data Processing on Large Clusters*" by two members of Google's R&D team, Jeffrey Dean and Sanjay Ghemawat, is the seminal article on MapReduce.

The article describes the methods used to split, process, and aggregate the large amount of data for the Google search engine.

The open-source version of MapReduce was later released within the Apache Hadoop project.

The phases of MapReduce

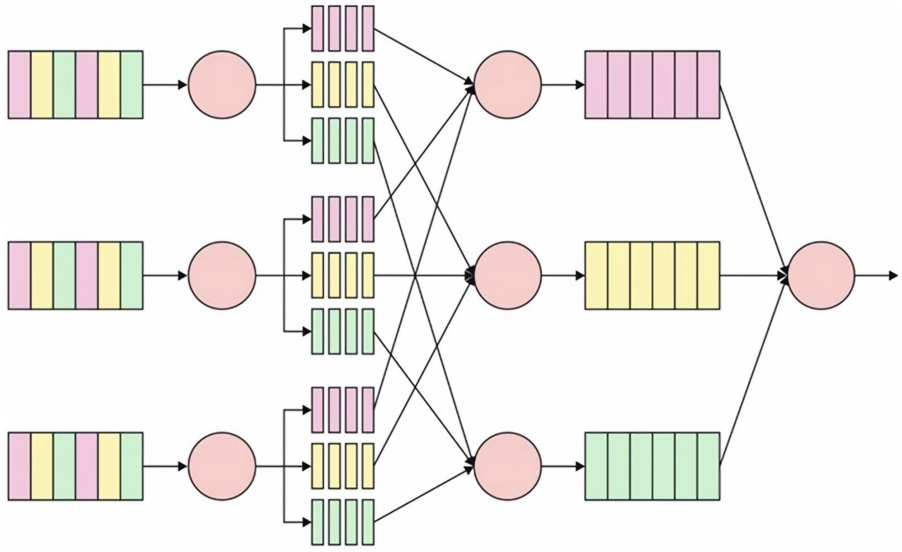
The phases of a MapReduce job:

- **split**: data is partitioned across several computer nodes
- **map**: apply a map function to each chunk of data
- **sort & shuffle**: the output of the mappers is sorted and distributed to the reducers
- **reduce**: finally, a reduce function is applied to the data and an output is produced

The phases of MapReduce

The phases of a MapReduce job:

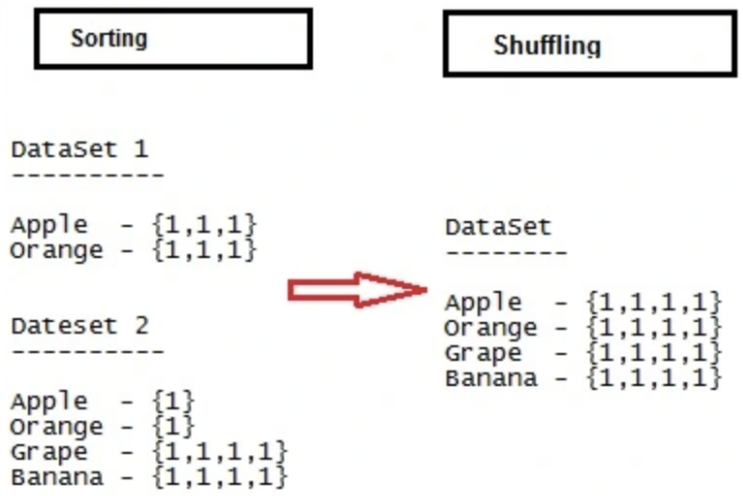
- **split**: partition and distribute data
- **map**: apply map function
- **sort & shuffle**: sort and distribute data
- **reduce**: apply reduce function



Source: Nature

MapReduce: shuffling and sorting

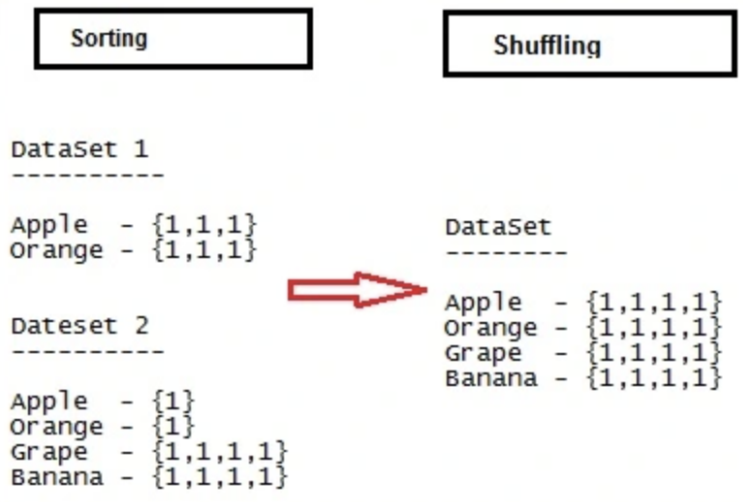
- Values are **grouped** by keys
- Often the most **costly** in a MapReduce job as mapping outputs have to be merged and sorted in order to transfer them to the reducer(s).



Source: creativeme1807

MapReduce: shuffling and sorting

- Splitting, sorting and shuffling usually done by the **framework**
- **Customization** of splitting, sorting and shuffling phases
  $\rightarrow$ e.g. by managing the amount of splitting or defining the sorting comparator



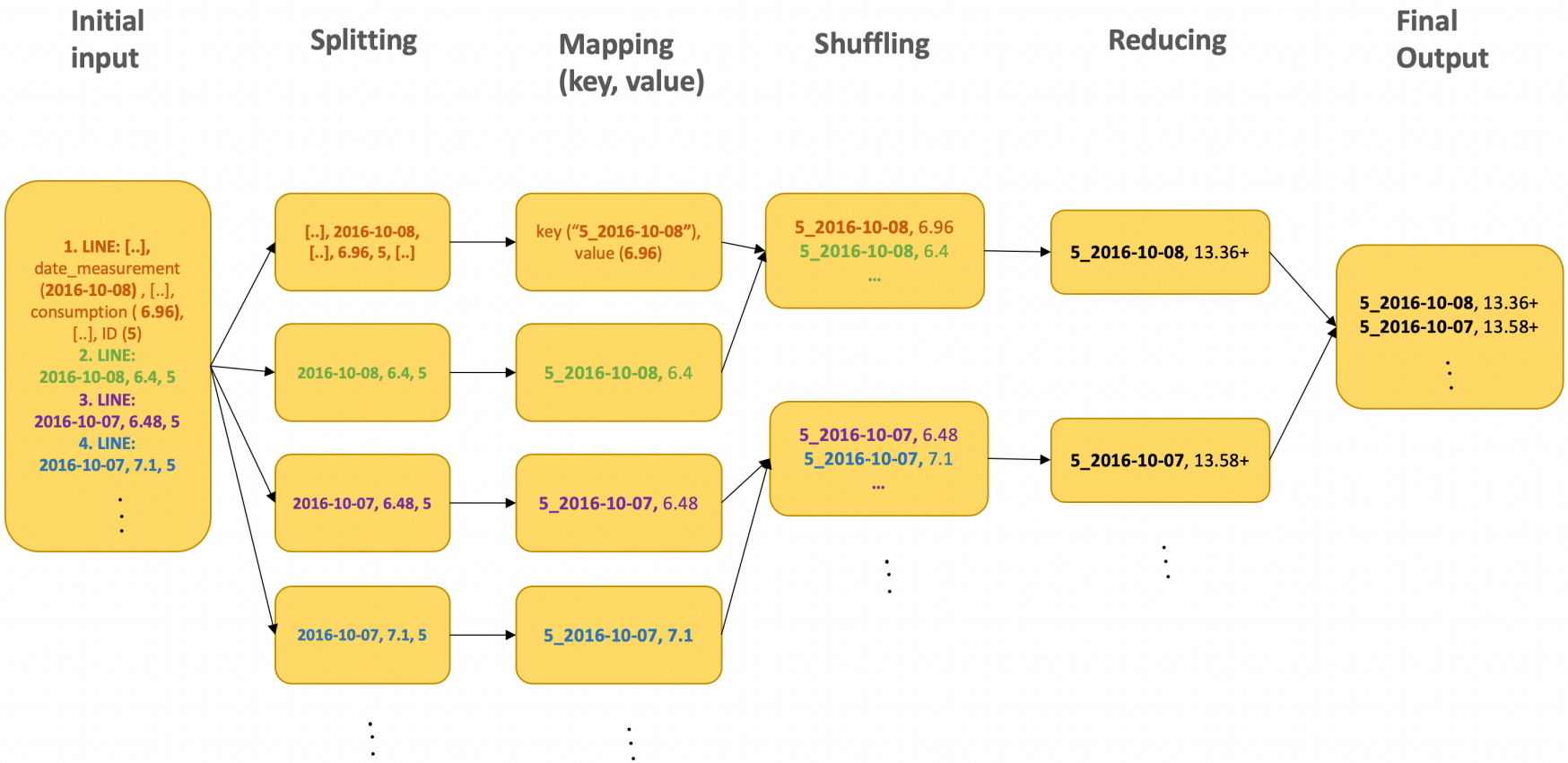
Source: creativeme1807

MapReduce Example: Electricity Consumption

Goal: Average daily electricity consumption per year per consumer **Data:** 2.9 millions of rows of electricty consumptions at different time points

NA	date_time	region	date_measurement	time_txt	validity	consumption	year	month	day	day_name	week	v3	v4	id_new
1	2016-10-08 23:15:00	02	2016-10-08	231500	PD0	6.96	2016	10	8	Saturday	40	92	0.2	512132
2	2016-10-08 23:30:00	02	2016-10-08	233000	PD0	6.4	2016	10	8	Saturday	40	92	0.2	512132
3	2016-10-08 23:45:00	02	2016-10-08	234500	PD0	6.48	2016	10	8	Saturday	40	92	0.1	512132
4	2016-10-09 00:00:00	02	2016-10-09	000000	PD0	7.76	2016	10	9	Sunday	40	92	0.1	512132
5	2016-10-09 00:15:00	02	2016-10-09	001500	PD0	6.8	2016	10	9	Sunday	40	93	0.1	512132
6	2016-10-09 00:30:00	02	2016-10-09	003000	PD0	6.96	2016	10	9	Sunday	40	93	0.1	512132
7	2016-10-09 00:45:00	02	2016-10-09	004500	PD0	6.52	2016	10	9	Sunday	40	94	0	512132
8	2016-10-09 01:00:00	02	2016-10-09	010000	PD0	6.8	2016	10	9	Sunday	40	94	0	512132
9	2016-10-09 01:15:00	02	2016-10-09	011500	PD0	7.52	2016	10	9	Sunday	40	93	0.1	512132
10	2016-10-09 01:30:00	02	2016-10-09	013000	PD0	6.56	2016	10	9	Sunday	40	93	0.1	512132
11	2016-10-09 01:45:00	02	2016-10-09	014500	PD0	8.4	2016	10	9	Sunday	40	93	0	512132
12	2016-10-09 02:00:00	02	2016-10-09	020000	PD0	6.76	2016	10	9	Sunday	40	93	0	512132
13	2016-10-09 02:15:00	02	2016-10-09	021500	PD0	7.2	2016	10	9	Sunday	40	94	0.2	512132
14	2016-10-09 02:30:00	02	2016-10-09	023000	PD0	7.88	2016	10	9	Sunday	40	94	0.2	512132
15	2016-10-09 02:45:00	02	2016-10-09	024500	PD0	7.16	2016	10	9	Sunday	40	94	0.4	512132
16	2016-10-09 03:00:00	02	2016-10-09	030000	PD0	6.36	2016	10	9	Sunday	40	94	0.4	512132
17	2016-10-09 03:15:00	02	2016-10-09	031500	PD0	6.56	2016	10	9	Sunday	40	94	0.1	512132
18	2016-10-09 03:30:00	02	2016-10-09	033000	PD0	6.76	2016	10	9	Sunday	40	94	0.1	512132
19	2016-10-09 03:45:00	02	2016-10-09	034500	PD0	7.4	2016	10	9	Sunday	40	94	0	512132
20	2016-10-09 04:00:00	02	2016-10-09	040000	PD0	6.64	2016	10	9	Sunday	40	94	0	512132
21	2016-10-09 04:15:00	02	2016-10-09	041500	PD0	7.12	2016	10	9	Sunday	40	94	0	512132
22	2016-10-09 04:30:00	02	2016-10-09	043000	PD0	6.72	2016	10	9	Sunday	40	94	0	512132
23	2016-10-09 04:45:00	02	2016-10-09	044500	PD0	8.28	2016	10	9	Sunday	40	94	0	512132

MapReduce Example: Electricity Consumption



MapReduce Use Cases

- **Pre-processing** such as filtering ("Grepping"), simple transformations,..
  - Use Case: ETL (Extraction - Transformation - Load) processes
- **Cross-Correlation**
  - Use Case: Pattern Recognition as in Text-, Marktanalyse,..
- **Relational Operations** such as Union, Intersection

MapReduce: Additional Notes

- Usually a single mapper and reducer do not suffice for a task -> Chaining MapReduce Jobs
- Output key-value pair can contain custom input format or custom data types in case e.g more or special objects have to be passed.

MapReduce: Key Takeaways

- the same map (and reduce) function is applied to all the chunks in the data
- the mapping and reduce functions have to be defined, custom splitting or sorting are optional as they are given by most MapReduce libraries.
- the map computations can be carried out in parallel because they're completely independent from one another.

Outline/next

- Introduction
- What is Big Data?
- The Hadoop distributed computing architecture
- HDFS hands-on exercises
- The YARN resource manager
- MapReduce
- MapReduce hands-on
- Concluding remarks

MapReduce hands-on

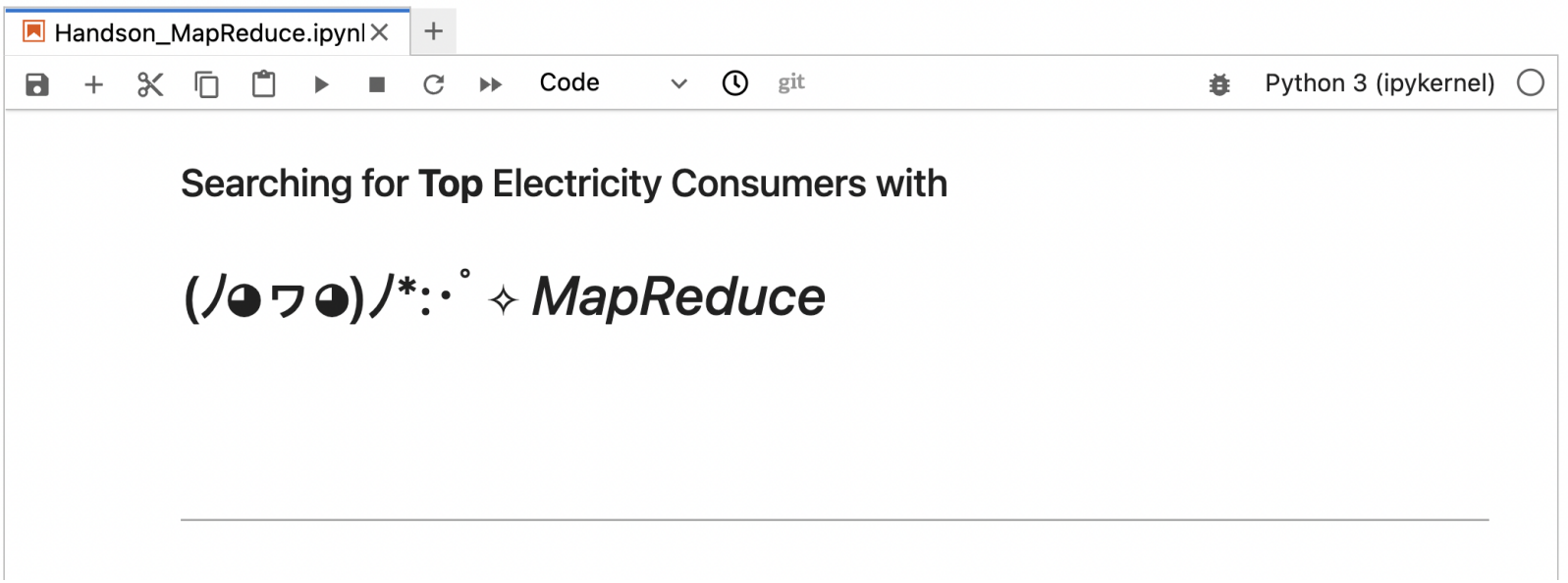
MapReduce Handson: Electricity Consumption

Goal: Daily average of electricity consumption per consumers **Data:** 2.9 millions of rows of consumer data

NA	date_time	region	date_measurement	time_txt	validity	consumption	year	month	day	day_name	week	v3	v4	id_new
1	2016-10-08 23:15:00	02	2016-10-08	231500	PD0	6.96	2016	10	8	Saturday	40	92	0.2	512132
2	2016-10-08 23:30:00	02	2016-10-08	233000	PD0	6.4	2016	10	8	Saturday	40	92	0.2	512132
3	2016-10-08 23:45:00	02	2016-10-08	234500	PD0	6.48	2016	10	8	Saturday	40	92	0.1	512132
4	2016-10-09 00:00:00	02	2016-10-09	000000	PD0	7.76	2016	10	9	Sunday	40	92	0.1	512132
5	2016-10-09 00:15:00	02	2016-10-09	001500	PD0	6.8	2016	10	9	Sunday	40	93	0.1	512132
6	2016-10-09 00:30:00	02	2016-10-09	003000	PD0	6.96	2016	10	9	Sunday	40	93	0.1	512132
7	2016-10-09 00:45:00	02	2016-10-09	004500	PD0	6.52	2016	10	9	Sunday	40	94	0	512132
8	2016-10-09 01:00:00	02	2016-10-09	010000	PD0	6.8	2016	10	9	Sunday	40	94	0	512132
9	2016-10-09 01:15:00	02	2016-10-09	011500	PD0	7.52	2016	10	9	Sunday	40	93	0.1	512132
10	2016-10-09 01:30:00	02	2016-10-09	013000	PD0	6.56	2016	10	9	Sunday	40	93	0.1	512132
11	2016-10-09 01:45:00	02	2016-10-09	014500	PD0	8.4	2016	10	9	Sunday	40	93	0	512132
12	2016-10-09 02:00:00	02	2016-10-09	020000	PD0	6.76	2016	10	9	Sunday	40	93	0	512132
13	2016-10-09 02:15:00	02	2016-10-09	021500	PD0	7.2	2016	10	9	Sunday	40	94	0.2	512132
14	2016-10-09 02:30:00	02	2016-10-09	023000	PD0	7.88	2016	10	9	Sunday	40	94	0.2	512132
15	2016-10-09 02:45:00	02	2016-10-09	024500	PD0	7.16	2016	10	9	Sunday	40	94	0.4	512132
16	2016-10-09 03:00:00	02	2016-10-09	030000	PD0	6.36	2016	10	9	Sunday	40	94	0.4	512132
17	2016-10-09 03:15:00	02	2016-10-09	031500	PD0	6.56	2016	10	9	Sunday	40	94	0.1	512132
18	2016-10-09 03:30:00	02	2016-10-09	033000	PD0	6.76	2016	10	9	Sunday	40	94	0.1	512132
19	2016-10-09 03:45:00	02	2016-10-09	034500	PD0	7.4	2016	10	9	Sunday	40	94	0	512132
20	2016-10-09 04:00:00	02	2016-10-09	040000	PD0	6.64	2016	10	9	Sunday	40	94	0	512132
21	2016-10-09 04:15:00	02	2016-10-09	041500	PD0	7.12	2016	10	9	Sunday	40	94	0	512132
22	2016-10-09 04:30:00	02	2016-10-09	043000	PD0	6.72	2016	10	9	Sunday	40	94	0	512132
23	2016-10-09 04:45:00	02	2016-10-09	044500	PD0	8.28	2016	10	9	Sunday	40	94	0	512132

MapReduce Job

Notebook to Solve:
/MapReduce/Handson_MapReduce.ipynb



Job Output

All directories created upon processing the job must be deleted on each run (-rm -r output), since Hadoop does not delete or overwrite existing data according to the WORM principle!

The option -D mapreduce.job.maps=4 directly after the directive jar (in this example -D mapreduce.job.maps=4) allows MapReduce properties to be changed at runtime.

The list of all MapReduce options are located in: mapred-default.xml∗

∗ This is the link to the latest stable version. There may be slight changes with respect to the version currently installed on the cluster.

Outline/next

- Introduction
- What is Big Data?
- The Hadoop distributed computing architecture
- HDFS hands-on exercises
- The YARN resource manager
- MapReduce
- MapReduce hands-on
- Concluding remarks

Concluding remarks

Where to find the course's material

These slides and all code used in the hands-on can be downloaded from:

https://bitbucket.org/bdtrain/resources

or from the course's home page

https://indico.ijs.si/event/1522/

Closing

Thank you for your attention.

Questions?