



PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

Introduction to Hadoop (part 1)

Dr. Giovanna Roda

Vienna Technical University - IT Services (TU.it)



What is Big Data?

The large amounts of data that are available nowadays cannot be handled with traditional technologies. "Big Data" has become the catch-all term for massive amounts of data as well as for frameworks and R&D initiatives aimed at working with it efficiently.

The “three V’s” which are often used to characterize Big Data:

- ▶ Volume (the sheer volume of data)
- ▶ Velocity (rate of flow of the data and processing speed needs)
- ▶ Variety (different sources and formats)

The three V's of Big Data

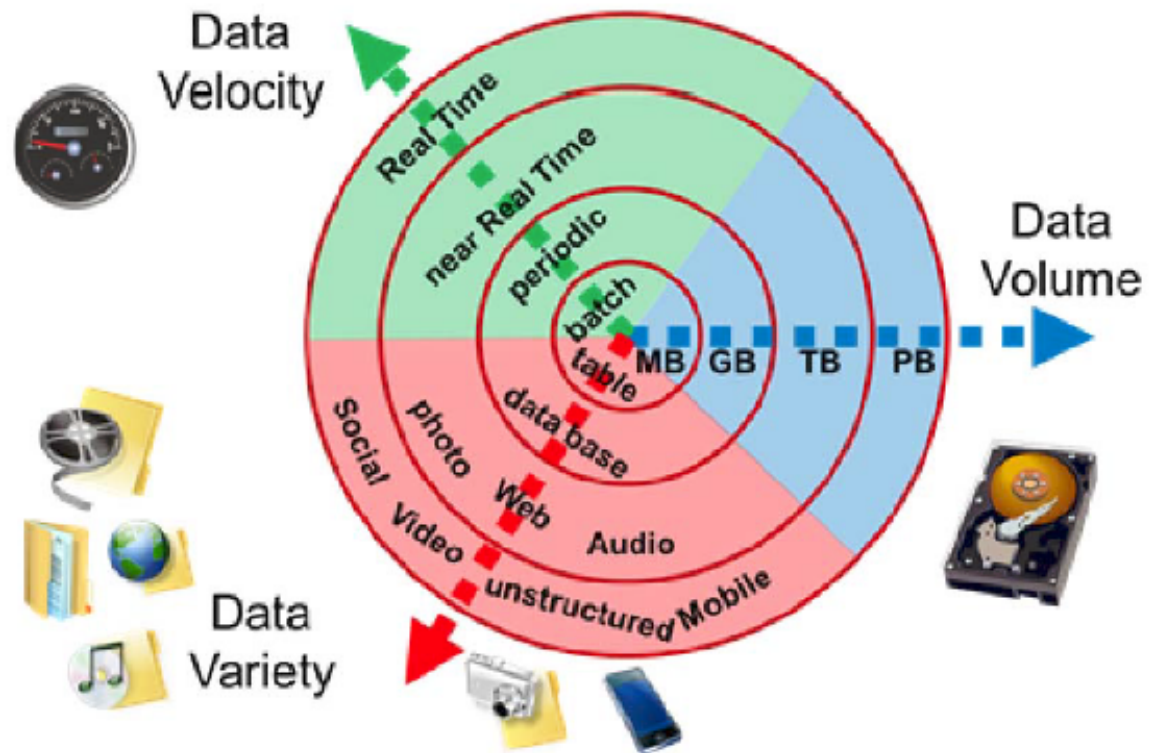


Image source: [Wikipedia](#)



The three V's of Big Data

Additionally some other characteristics to keep in mind when dealing with Big Data and with data in general:

- ▶ Veracity (quality or trustworthiness of data)
- ▶ Value (economic value of the data)
- ▶ Variability (general variability in any of the data characteristics)



Challenges posed by Big Data

Here are some challenges posed by Big Data:

- ▶ disk and memory space
- ▶ processing speed
- ▶ hardware faults
- ▶ network capacity and speed
- ▶ optimization of resources usage

In the rest of this seminar we are going to see how Hadoop tackles them.

Big Data and Hadoop

[Apache Hadoop](#) is one of the most widely adopted frameworks for Big Data processing.



Some facts about Hadoop:

- ▶ project of the [Apache Open Source Foundation](#)
- ▶ open source
- ▶ facilitates distributed computing
- ▶ initially released in 2006. Last version is 3.3.0 (Jul. 2020), stable [3.2.1](#) (Sept. 2019)
- ▶ originally inspired by Google's MapReduce and the proprietary GFS (Google File System)



Some Hadoop features explained

- ▶ *fault tolerance*: the ability to withstand hardware or network failures
- ▶ *high availability*: this refers to the system minimizing downtimes by eliminating *single points of failure*
- ▶ *data locality*: task are run where the data is located to reduce the costs of moving large amounts of data around



How does Hadoop address the challenges of Big Data?

- ▶ performance: allows processing of large amounts of data through distributed computing
- ▶ *data locality*: tasks are run where the data is located
- ▶ cost-effectiveness: it runs on commodity hardware
- ▶ scalability: new and/or more performant hardware can be added seamlessly
- ▶ offers fault tolerance and high availability
- ▶ good abstraction of the underlying hardware and easy to use
- ▶ provides SQL and other abstraction frameworks like Hive, Hbase



The Hadoop core

The core of Hadoop consists of:

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



The Hadoop core is written in Java.



Next:

The core of Hadoop consists of:

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



The Hadoop core is written in Java.



What is HDFS?

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

In order to prevent data loss and/or task termination due to hardware failures HDFS uses *replication*, that is simply making multiple copies —usually 3— of the data (*).

The feature of HDFS being able to withstand hardware failure is known as *fault tolerance* or *resilience*.

(*) Starting from version 3, Hadoop provides erasure coding as an alternative to replication



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. See for instance the white paper [“Comparing cost and performance of replication and erasure coding”](#).



HDFS architecture

A typical Hadoop cluster installation consists of:

- ▶ a **NameNode**

responsible for the bookkeeping of the data partitioned across the DataNodes, managing the whole filesystem metadata, load balancing,

- ▶ a **secondary NameNode**

this is a copy of the NameNode, ready to take over in case of failure of the NameNode.

A secondary NameNode is necessary to guarantee *high availability* (since the NameNode is a *single point of failure*)

- ▶ multiple **DataNodes**

here is where the data is saved and the computations take place



HDFS architecture: internal data representation

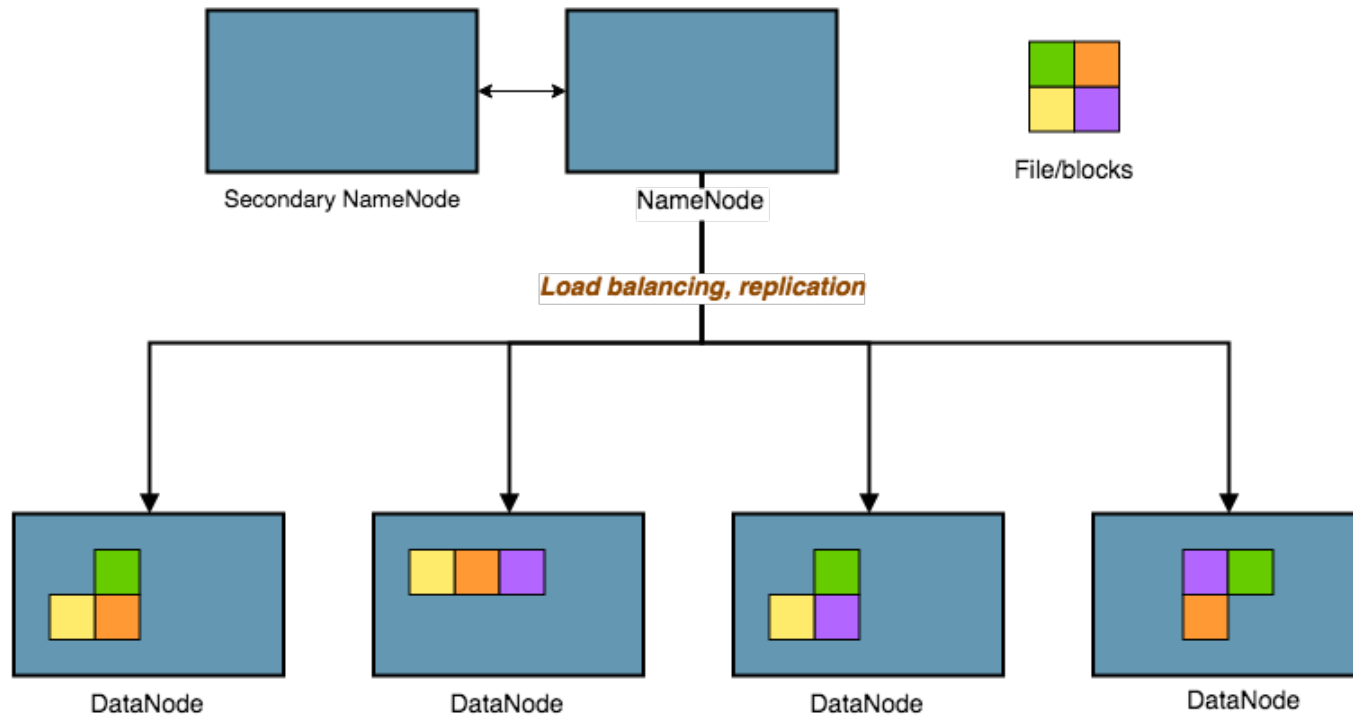
HDFS supports working with very large files.

Internally, files are split into *blocks*. One of the reason for that is that block objects have all the same size.

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

What is HDFS?

HDFS Architecture





HDFS architecture

Some notes:

- ▶ one host can act both as NameNode and DataNode. These are just services running on the nodes
- ▶ a minimal HDFS cluster should have 3 nodes to support the default replication of 3 (one of the nodes acts as both NameNode and DataNode)
- ▶ rebalancing of data nodes is not done automatically but it can be triggered with:

```
sudo -u hdfs hdfs balancer -threshold 5
```

(here we balance data on the DataNodes so that load differs by at most 5%, default is 10%)



Look at the current cluster

We're now going to look at the current cluster configuration.

In order to do that we must first:

- ▶ login to the system
- ▶ activate the appropriate Hadoop environment
- ▶ (optional) check environment variables



Login to the system and activate environment

- ▶ login to VSCFS and open a terminal. Instructions are in: GUI interface with NoMachine client:

https://wiki.hpc.fs.uni-lj.si/index.php?title=Dostop#Dostop_preko_grafi.C4.8Dnega_vmesnika,

SSH: https://wiki.hpc.fs.uni-lj.si/index.php?title=Dostop#Dostop_preko_SSH

- ▶ activate Hadoop module

```
module avail Hadoop # show available Hadoop installations (case-sensitive)
```

```
module load Hadoop # this will load the latest Hadoop (2.10)
```

```
module list
```

You should have the following modules 3 activated now:

```
GCCcore/8.3.0, Java/1.8.0_202, Hadoop/2.10.0-GCCcore-8.3.0-native
```



Check environment variables

These variables usually need to be defined before starting to work with Hadoop:

`JAVA_HOME, PATH, HADOOP_CLASSPATH, HADOOP_HOME`

Check with:

```
echo $JAVA_HOME
```

```
echo $PATH
```

```
echo $HADOOP_CLASSPATH
```

```
echo $HADOOP_HOME
```



Let's look at the current Hadoop configuration

- ▶ what are the NameNode(s)?

```
hdfs getconf -namenodes
```

- ▶ list all DataNodes

```
yarn node -list -all
```

- ▶ block size

```
hdfs getconf -confKey dfs.blocksize|numfmt --to=iec
```

- ▶ replication factor

```
hdfs getconf -confKey dfs.replication
```



Let's look at the current Hadoop configuration

- ▶ how much disk space is available on the whole cluster?

```
hdfs getconf -namenodes
```

- ▶ list all DataNodes

```
yarn node -list -all
```

```
yarn node -list -showDetails # show more details for each host
```

- ▶ block size

```
hdfs getconf -confKey dfs.blocksize|numfmt --to=iec
```

- ▶ replication factor

```
hdfs getconf -confKey dfs.replication
```



Basic HDFS filesystem commands

You can regard HDFS as a regular file system, in fact many HDFS shell commands are inherited from the corresponding bash commands. Here's three basic commands that are specific to HDFS.

command	description
<code>hadoop fs -put</code> <code>hdfs dfs -put</code>	Copy single src, or multiple srcs from local file system to the destination file system
<code>hadoop fs -get</code> <code>hdfs dfs -get</code>	Copy files to the local file system
<code>hadoop fs -usage</code> <code>hdfs dfs -usage</code>	get help on <code>hadoop fs</code>



Basic HDFS filesystem commands

Notes

1. You 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).
2. The full list of Hadoop filesystem shell commands can be found here:
<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html>



Basic HDFS filesystem commands that also exist in bash

bash	HDFS	description
<code>mkdir</code>	<code>hadoop fs -mkdir</code> <code>hdfs dfs -mkdir</code>	create a directory
<code>ls</code>	<code>hadoop fs -ls</code> <code>hdfs dfs -ls</code>	list files
<code>cp</code>	<code>hadoop fs -cp</code> <code>hdfs dfs -cp</code>	copy files
<code>mv</code>	<code>hadoop fs -mv</code> <code>hdfs dfs -mv</code>	move files
<code>cat</code>	<code>hadoop fs -cat</code> <code>hdfs dfs -cat</code>	concatenate files and print them to standard output
<code>rm</code>	<code>hadoop fs -rm</code> <code>hdfs dfs -rm</code>	remove files



Exercise: basic HDFS commands

1. List your HDFS home with: `hdfs dfs ls`
2. create a directory `small_data` in your HDFS share (use relative path, so the directory will be created in your HDFS home)
3. Upload a file from the local filesystem to `small_data`. If you don't have a file, use `/home/campus00/public/data/fruits.txt`
4. list the contents of `small_data` on HDFS to check that the file is there
5. compare the space required to save your file in the local filesystem and on HDFS using `du`
6. clean up: (use `hadoop rm -r small_data`)



Next:

The core of Hadoop consists of:

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



The Hadoop core is written in Java.

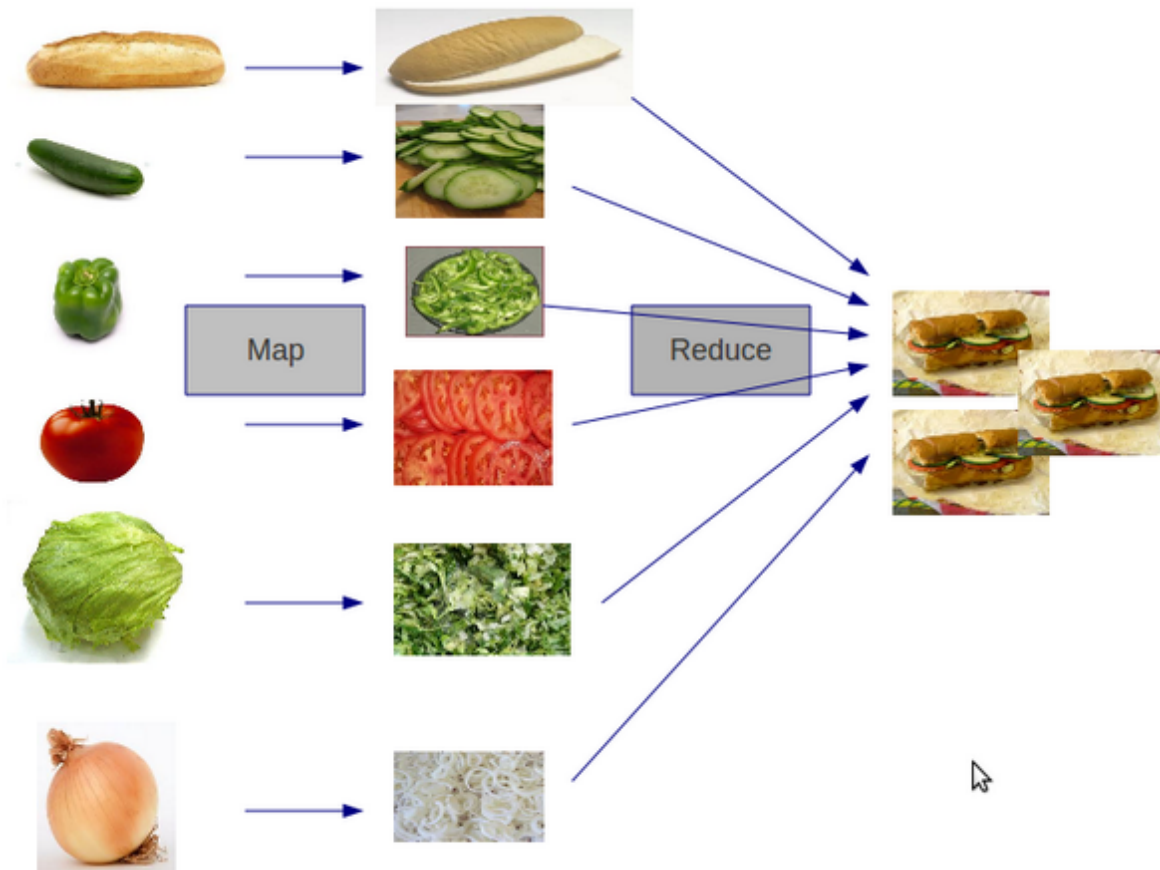


MapReduce: the origins

The seminal article on MapReduce is: ["MapReduce: Simplified Data Processing on Large Clusters"](#) by Jeffrey Dean and Sanjay Ghemawat from 2004.

In this paper, the authors (members of the Google research team) describe the methods used to split, process, and aggregate the large amount of data at the basis of the Google search engine.

MapReduce explained



Source: Stackoverflow



MapReduce explained

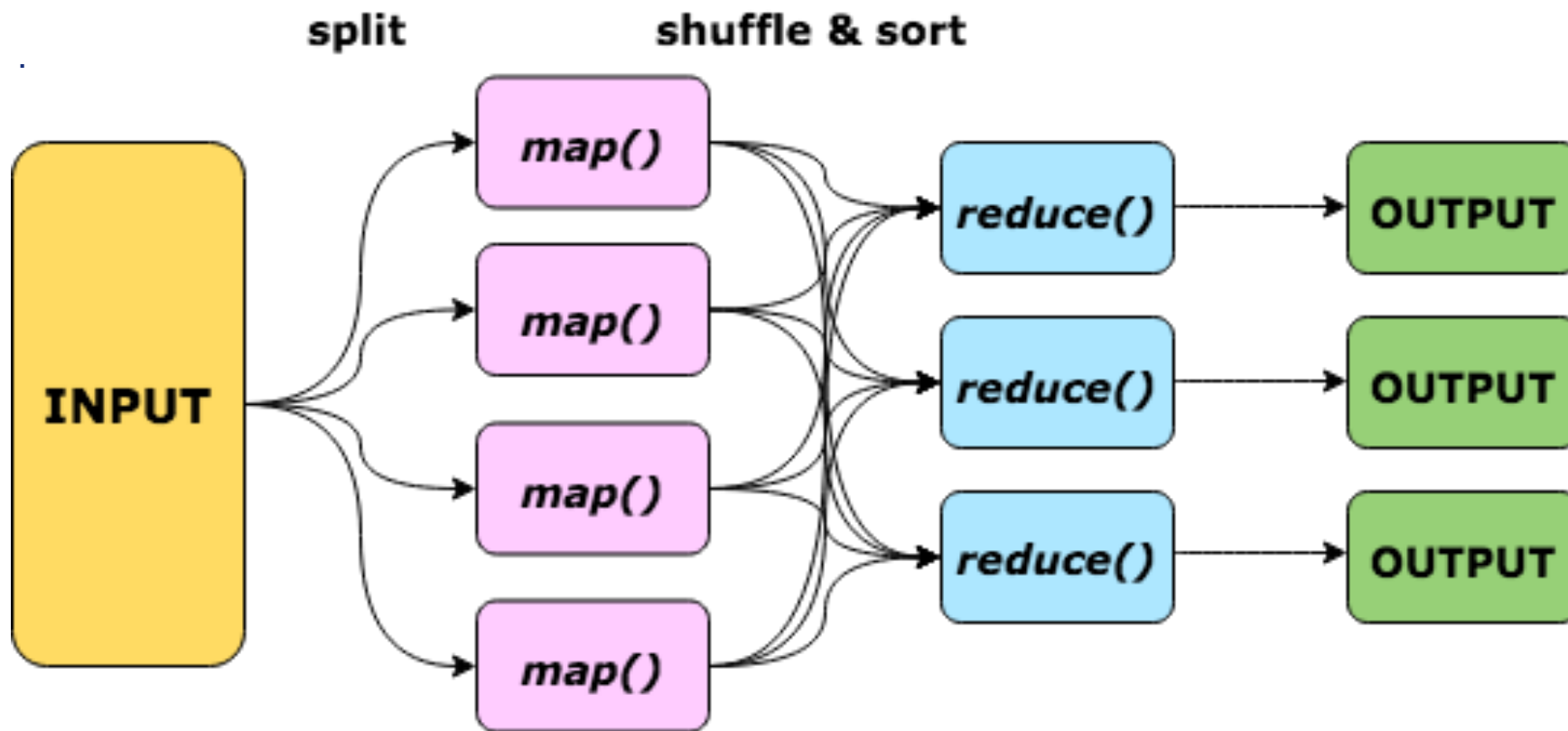
The phases of a MapReduce job

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

Notes

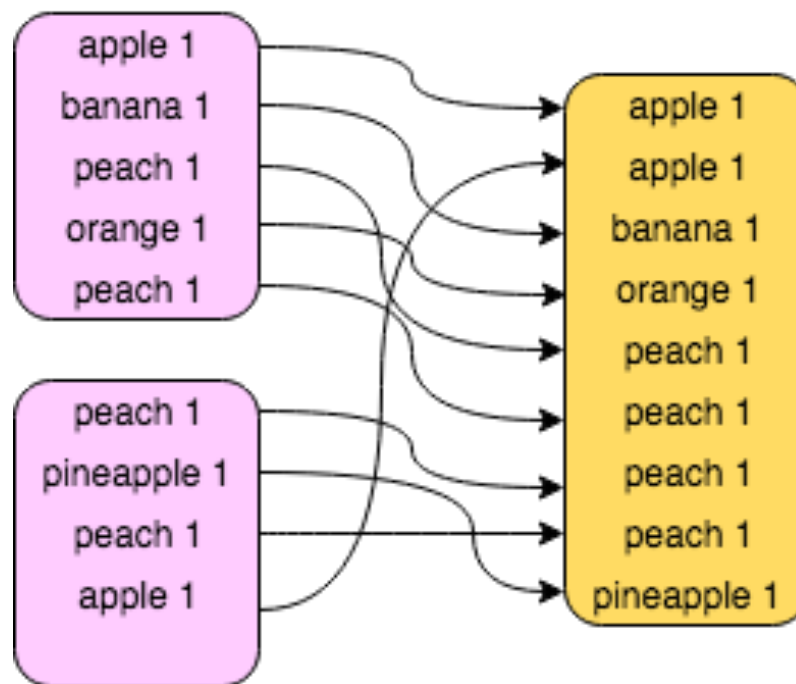
- ▶ Note 1: the same map (and reduce) function is applied to all the chunks in the data.
- ▶ Note 2: the map and reduce computations can be carried out in parallel because they're completely independent from one another.
- ▶ Note 3: the split is not the same as the internal partitioning into blocks

MapReduce explained



MapReduce explained

shuffling & sorting





Look at some Mapreduce configuration values

- ▶ default number of map tasks

```
hdfs getconf -confKey mapreduce.job.maps
```

- ▶ default number of reduce tasks

```
hdfs getconf -confKey mapreduce.job.reduces
```

- ▶ the total amount of memory buffer (in MB) to use when sorting files

```
hdfs getconf -confKey mapreduce.task.io.sort.mb
```



A simple example

We're now going to run a simple example on the cluster using MapReduce and the Hadoop's streaming library.

Check/set environment variables

These variables usually need to be defined before starting to work with Hadoop:

JAVA_HOME, PATH, HADOOP_CLASSPATH

```
echo JAVA_HOME
```

```
echo PATH
```

```
echo HADOOP_CLASSPATH
```



Hadoop streaming

The mapreduce streaming library allows to use any executable as mappers and reducers.

The requirements on the mapper and reducer executables is that they are able to:

- ▶ read the input from `stdin` (line by line) and
- ▶ emit the output to `stdout`.

Here's the documentation for streaming: <https://hadoop.apache.org/docs/r2.10.0/hadoop-streaming/HadoopStreaming.html>



Hadoop streaming

To start a MapReduce streaming job use:

```
hadoop jar hadoop-streaming.jar  
-input myInputDirs  
-output myOutputDir  
-mapper mapper_executable  
-reducer reducer_executable
```

Note 1: in the input files and output files are located by default on HDFS

Note 2: in Hadoop 3 you can use the shorter command `mapred streaming -input ...` in place of `hadoop jar ...` to launch a MapReduce streaming job.



Find the streaming library path

Use one of the following commands to find the location of the streaming library:

```
which hadoop
```

```
echo $HADOOP_HOME
```

```
alternatives --display hadoop
```

```
find /opt/pkg/software/Hadoop/2.10.0-GCCcore-8.3.0-native/ -name
```

```
'hadoop-streaming*.jar'
```

```
export STREAMING_PATH=/opt/pkg/software/Hadoop/2.10.0-GCCcore-8.3.0-  
native/share/hadoop/tools/lib
```



Run a simple MapReduce job

We are going to use two bash commands as executables:

```
export STREAMING_PATH=/opt/pkg/software/Hadoop/2.10.0-GCCcore-8.3.0-  
native/share/hadoop/tools/lib
```

```
hadoop jar ${STREAMING_PATH}/hadoop-streaming-2.10.0.jar \  
  -input data/wiki321MB \  
  -output simplest-out \  
  -mapper /usr/bin/cat \  
  -reducer /usr/bin/wc
```



Run a simple MapReduce job – look at the output

What did we just do?

- ▶ the mapper just outputs the input as it is
- ▶ the reducer counts the lines in the input text

Where to find the output?

It is in `simplest-out`

Since the output folder is on HDFS, so we list it with

```
hdfs dfs -ls simplest-out
```

If the output folder contains a file named `_SUCCESS`, then the job was successful. The actual output is in the file(s) `part-*`. Look at the output with `hdfs dfs -cat simplest-mr-job/part-*` and compare with the output of `wc` on the local file.



Run a simple MapReduce job – look at the logging messages

Let look at some of the logging messages of MapReduce:

- ▶ how many mappers were executed?

MapReduce automatically sets the number of map tasks according to the size of the input (in blocks). The minimum number of map tasks is determined by `mapreduce.job.maps`

- ▶ how many reducers?



MapReduce: word count

Wordcount is the classic MapReduce application. We are going to implement in in Python and learn some more about MapReduce along the way.

We need to create two scripts:

- ▶ mapper.py
- ▶ reducer.py



MapReduce: word count – the mapper

The mapper needs to be able to read from input and emit a series of `<key, value>` pairs. By default, the format of each mapper's output line is a tab-separated string where anything preceding the tab is regarded as the key.

```
#!/bin/python3
import sys
for line in sys.stdin:
    words = line.strip().split()
    for word in words:
        print("{}\t{}".format(word,1))
```



MapReduce: word count – test the mapper

Test the mapper on a local file, for instance `wiki321MB`

```
FILE=/home/campus00/hadoop/data/wiki321MB  
chmod 755 mapper.py  
head -2 $FILE | ./mapper.py | less
```

If everything went well, you should get an output that looks like this:

```
8753      1  
Dharma    1  
.  
.  
.
```



MapReduce: word count –the reducer

```
#!/bin/python3
import sys
current_word, current_count = None, 0
for line in sys.stdin:
    word, count = line.strip().split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print("{}\t{}".format(current_word, current_count))
            current_count = count
            current_word = word
        if current_word == word:
            print("{}\t{}".format(current_word, current_count))
```



MapReduce: word count – test the reducer

Test the reducer on a local file, for instance `wiki321MB` (as an alternative use `fruits.txt` or any other text file).

```
FILE=/home/campus00/hadoop/data/wiki321MB
chmod 755 reducer.py
head -2 $FILE | ./mapper.py | sort | ./reducer.py | less
```

If everything went well, you should get an output that looks like this:

```
"Brains,      1
"Chapter     1
. . .
```



MapReduce: word count – sort the output

The reducer is emitting an output sorted by key (in this case keys are words).

To view the output sorted by count use:

```
head -2 $FILE | ./mapper.py | sort | ./reducer.py | sort -k2nr | head
```

Note: `sort -k2nr` sorts numerically by the second field in reverse order.

What is the most frequent word?



MapReduce: word count – Hadoop it up

Now that we've tested our mapper and reducer we're ready to run the job on the cluster.

We need to tell Hadoop to upload our mapper and reducer code to the datanodes by using the option `-file` ([here](#) you can find all Hadoop streaming options).

We just need two more steps:

- ▶ prepare a folder `wordcount_out` on HDFS where the output will be written

```
hdfs dfs -mkdir wordcount_out
```

- ▶ upload the data to HDFS

```
echo $FILE # /home/campus00/hadoop/data/wiki321MB
```

```
hdfs dfs -put $FILE
```



MapReduce: word count – Hadoop it up

Finally, start the MapReduce job.

```
hadoop jar ${STREAMING_JAR}/hadoop-streaming-2.10.0.jar \  
  -file mapper.py \  
  -file reducer.py \  
  -input wiki321MB \  
  -output wordcount_out \  
  -mapper mapper.py \  
  -reducer reducer.py
```



MapReduce: word count – check the output

- ▶ does the output folder contain a file named `_SUCCESS?`
- ▶ check the output with

```
hdfs dfs -cat wordcount_out/part-00000 |head
```

Of course we would like to have the output sorted by value (the word frequency).

Let us just execute another MapReduce job acting on our output file.



MapReduce: word count – second transformation

The second transformation will use a mapper that swaps key and value and no reducer.

Let us also filter for words with frequency >100.

Mapper is a shell script. Call it `swap_keyval.sh`.

```
#!/bin/bash
while read key val
do
    if (( val > 100 )); then
        printf "%s\t%s\n" "$val" "$key"
    fi
done
```



MapReduce: word count – second transformation - run

Run MapReduce job

```
hadoop jar ${STREAMING_PATH}/hadoop-streaming-2.10.0.jar \  
-file swap_keyval.sh \  
-input wordcount_out \  
-output wordcount_out2
```

Check the output. Does it look right?



MapReduce: configure sort with KeyFieldBasedComparator

Map output by default is sorted in ascending order by key.

We can control how a mapper is going to sort by configuring the comparator directive to use the special class `KeyFieldBasedComparator`.

This class has some options similar to the Unix `sort` (`-n` to sort numerically, `-r` for reverse sorting, `-k pos1[,pos2]` for specifying fields to sort by).



MapReduce: configure sort with KeyFieldBasedComparator

```
COMPARATOR=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator
```

```
hadoop jar ${STREAMING_PATH}/hadoop-streaming-2.10.0.jar \  
-D mapreduce.job.output.key.comparator.class=$COMPARATOR \  
-D mapreduce.partition.keycomparator.options=-nr \  
-file swap_keyval.sh \  
-input wordcount_out \  
-output wordcount_out3
```



MapReduce exercise: change number of mappers or reducers

Map Try to change the number of map and/or reduce tasks using the options (here we use for instance 10 map tasks and 2 reduce tasks):

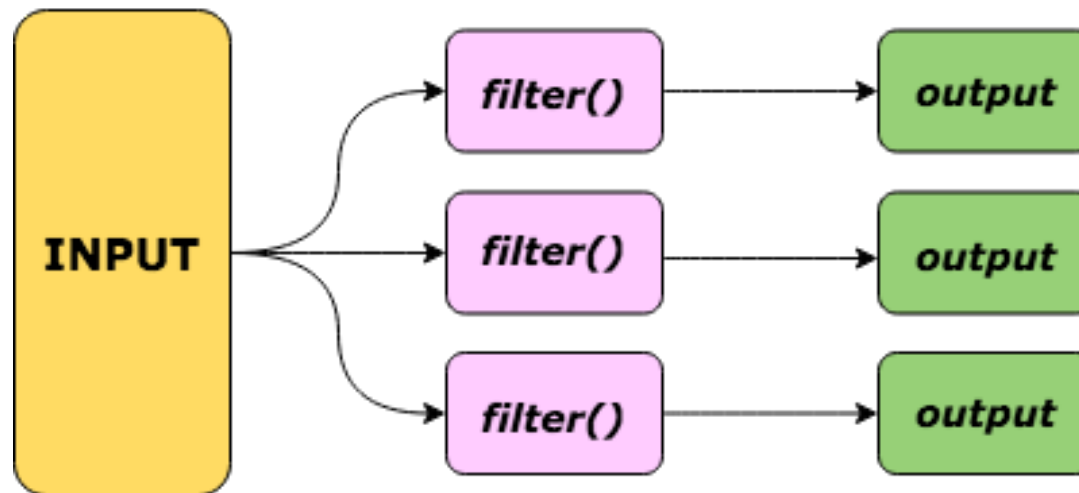
- D mapred.map.tasks=10
- D mapred.reduce.tasks=2

Does this improve the performance of your MapReduce job? How does the job output look like?

MapReduce patterns

What can MapReduce be used for? Data summarization by grouping, filtering, joining, etc. We have already seen the filtering pattern in our second transformation for wordcount:

split mappers (no reducers)



For more patterns see: [“MapReduce Design Patterns”](#), O’Reilly



Running Hadoop on your pc

Hadoop can run on a single node (of course in this case no replication is permitted):

[Hadoop: Setting up a Single Node Cluster](#)). In this case, no Yarn is needed.

By default, Hadoop is configured to run in a non-distributed mode as a single Java process. In a single node setup, it's possible to allow pseudo-distributed operation by allowing each Hadoop daemon to run as separate Java process.

It is also possible to use MapReduce without HDFS, using the local filesystem.



Why learn about HDFS and MapReduce?

HDFS and MapReduce are part of most Big Data curricula even though one ultimately will probably use higher level frameworks for working with Big Data.

One of the main limitations of Mapreduce is the disk I/O. The successor of Mapreduce, Apache Spark, offers better performance by orders of magnitude thanks to its in-memory processing. The Spark engine does not need HDFS.

A higher level framework like Hive allows to access data using HQL (Hive Query Language), a language similar to SQL.

Still, learning HDFS & MapReduce is useful because they exemplify in a simple way the foundational issues of the Hadoop approach to distributed computing.



Recap

- ▶ what is Big Data?
- ▶ what is Apache Hadoop?
- ▶ some Hadoop features explained
- ▶ Architecture of a Hadoop cluster
- ▶ HDFS, the Hadoop Distributed File System
- ▶ the MapReduce framework
- ▶ MapReduce streaming
- ▶ the simplest MapReduce job
- ▶ MapReduce wordcount
- ▶ using the Hadoop comparator class
- ▶ MapReduce design patterns
- ▶ why learn HDFS and MapReduce?
- ▶ can I run Hadoop on my pc?



PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

THANK YOU FOR YOUR ATTENTION

www.prace-ri.eu