Big Data

Lecture 1 – Introduction and MapReduce programming

Gianluca Quercini

gianluca.quercini@centralesupelec.fr

Centrale DigitalLab, 2021

Organization of the course

- **Lecture 1**. Introduction and MapReduce programming (03/05).
- **Tutorial 1.** MapReduce programming (03/05)
- **Lecture 2**. Hadoop and its ecosystem: HDFS (05/05).
- **Lecture 3**. Introduction to Apache Spark (05/05).
- **Tutorial 2.** Introduction to Spark programming (05/05).
- **Lecture 4**. Structured APIs and Structured Streaming (10/05).
- **Tutorial 3.** Introduction to DataFrames and Spark SQL (10/05).
- **Lab assignment 1.** Apache Spark programming (12/05).
- **Tutorial 4.** Structured Streaming (12/05).
- **Lecture 5**. Distributed databases and NoSQL (17/05).
- **Lecture 6**. Document oriented databases: MongoDB (17/05).
- **Lab assignment 2.** MongoDB (19/05).

Class material

Available on ▸ This website

- Slides of the lectures.
- Tutorials and lab assignments.
- References (books and articles).

Evaluation

- **Lab assignments**. Lab assignments 1 and 2 will be graded.
  - Submission: code source + written report.

Contact

Email: gianluca.quercini@centralesupelec.fr

What you will learn

In this lecture you will learn:

- **Big data** notions, motivations and challenges.
- Basic notions of **Hadoop** and its ecosystem.
- The **MapReduce** programming paradigm.

What is data?

Definition (Data)

Data are *raw symbols* that represent the properties of objects and events and in this sense data has no meaning of itself, it simply exists (Russell L. Ackoff, 1989). ▸ Source

- **Example.** {"John", "Smith", 30000}
- **Information.** Data + *meaning*.
  - {(first name, "John"), (last name, "Smith"), (salary, 30000)}

Definition (Dataset)

A **dataset** is a collection of data.

- Data can be categorized based on their **structure**.

Structured data

Definition (Structured data)

Structured data describe the *properties* (e.g., the name, address, credit card number and phone number) of *entities* (e.g., customer, products) following a fixed *template* or *model*.

- Records stored in the tables of a **relational database**.
- Each property is easily distinguishable from the others.
    - It fits one unit of the structure (e.g., a column of a table).

Unstructured data

Definition (Unstructured data)

Unstructured data describe entities that lack a clear structure due to their properties not being immediately distinguishable.

- **Text** is unstructured.
  - Description of entity properties drowned in a rich context.
  - No direct access to these properties.

Semi-structured data

Definition (Semi-structured data)

Semi-structured data have a structure in which the entities and their properties are easily distinguishable, but the organization of the structure is not as rigorous as in a table of a relational database.

- Examples: XML, JSON, HTML documents, spreadsheets...

Example (XML document)

```
<book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
</book>
```

What is Big Data?

Definition (Big Data)

The term **Big Data** refers to an accumulation of data that is too large and complex for processing by traditional database management tools. ‣ Source

The term Big Data also refers to:

- The (hardware and software) **solutions** developed to manage large volumes of data.
- The **branch of computing** studying solutions to manage large volumes of data.

Other sources define Big Data in terms of its characteristics.

The 3 Vs of Big Data

Definition (3V)

Big data is high-**Volume**, high-**Velocity** and high-**Variety** information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making (Gartner).

- **Volume**, the size of a dataset.
- **Velocity**, the necessity of processing data as they arrive.
- **Variety**, the heterogeneous nature of data (structured, unstructured, semi-structured).

The 4 Vs of Big Data

Definition (4V)

Big Data consists of extensive datasets primarily in the characteristics of **Volume**, **Variety**, **Velocity**, and/or **Variability** that require a **scalable** architecture for efficient storage, manipulation, and analysis (NIST).

- Subtle difference between **variety** and **variability**.
  - **Variety**: a bakery that sells ten types of bread.
  - **Variability**: a bakery that sells only one type of bread that tastes differently every day.

- **Scalability**: the ability of a system architecture to manage growing amounts of data, without a significant decrease of its performance

The 4 Vs of Big Data: example

Example (4V)

- *Sentiment analysis* system that processes tweets to derive the general mood about a political candidate.
- Language analysis: positive/negative/neutral sentiment?
- **Volume**. Millions of tweets.
- **Velocity.** Constant stream of data (7,500 tweets/second).
- **Variety.** Text, images and links to Web pages.
- **Variability.** The meaning of each word changes depending on the context.
  - *I'm* deeply *satisfied about the candidate* ☺
  - *I'm* deeply *offended by the candidate* ☹

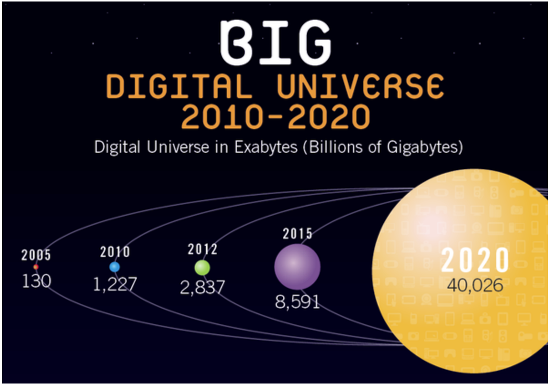
More Vs

- **Veracity.** Data might not correspond to the truth.
  - Fake news retweeted multiple times.
  - Uncertainty. The example of *Google Flu Trends*.
- **Value.** Separating the wheat from the chaff.
  - Lot of data available.
  - Identify the data that can have some value.
  - Discard the other data.

Scalability

- **Scalability**: the ability of a system to handle growing amounts of data, without a significant decrease of its performances.
- Two techniques:
  - **Vertical scaling** (**scale-up**).
    - Upgrade the existing infrastructure (more memory, computing power. . .).
  - **Horizontal scaling** (**scale-out**).
    - Add machines to the existing infrastructure.
    - Distribute the data and the workload across several machines.
- **Advantages** of vertical scaling:
  - **Easier** to maintain a single machine than many.
  - **Centralized control** over the data and the computations.
- **Advantages** of horizontal scaling:
  - **Limitless upgrade** of the computing power of a system.
  - **Fault tolerance**.

Where does Big Data come from?



Source: IDC, 2014

- The Web: social networks, blogs, wikis.
- Sensors: surveillance cameras, medical devices, cellphones.
- Companies (e.g., Amazon, UPS, Spotify, Netflix).

Big Data applications

- **Communications, media entertainment.**
  - Recommendation systems, social network analysis ...
- **Web search engines.**
- **Banking industry.**
  - Fraud detection, anti-money laundering ....
- **Healthcare industry.**
  - Diagnostics, medical research ....
- **Government agencies.**
  - Processing of unemployment claims, homeland security...

Recommendation systems

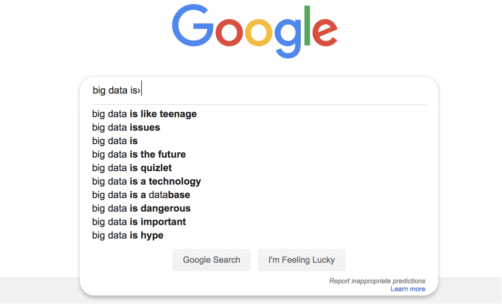
- Suggest items (songs, movies, books,...) to people based on:
  - their own tastes (**content-based filtering**);
  - similar people's tastes (**collaborative filtering**);
  - both (**hybrid systems**).

- Several companies and applications use a recommendation system.
  - Amazon, Spotify, Netflix, MovieLens...

- **Recommendation**: outcome of the analysis of **large volumes** of data.
  - How to **store** all these data?
    - Centralized database or distributed database?
    - Which data model?
  - How to **process** all these data?

Big Data: search engines

- Return a list of Web pages related to a search query.
- Need to **index** all Web pages.
    - **Inverted index:** for each word, list the Web pages containing that word.
- Need to **rank** all Web pages.
    - *wikipedia.org* (supposedly) more important than *myblog.com*.



Big Data challenges

- In this course, we study two main challenges: **processing** and **storage**.

Processing.

- Parallelize the computation across machines.
- Parallel Databases.
- Distributed processing frameworks (e.g., Hadoop MapReduce/Spark)

Storage.

- Distributed file systems.
- Distributed (relational/NoSQL) databases.

Processing big data

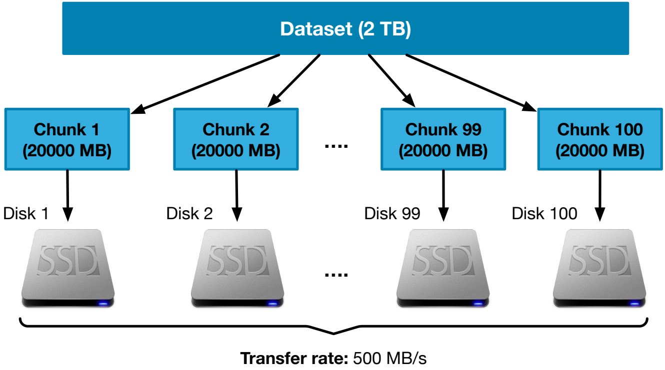
Why processing Big Data is challenging?

- **Disk storage** capacities have increased rapidly over the years.
  - A typical disk from 1990 could store **1,370 MB of data** (cf. Seagate ST-41600n).
  - A typical disk (SSD) today can store **2 TB of data** (cf. Seagate Barracuda 120).
- **Disk access** increased much slower.
  - A typical disk from 1990 had a **transfer speed** of 4.4 MB/s.
  - A typical disk (SSD) from today has a **transfer speed** of 500 MB/s.

- In 1990 it could take 5 minutes to read all the data from a disk.
- In 2020 it takes more than one hour to read all the data from a disk.

Processing big data: parallelization

- Split the data into many smaller chunks stored on separate disks.
- Read **in parallel** from each disk.



☞ Each disk may contain chunks from different datasets.

Parallelization: challenges

- **Hardware failure.** Lots of disks → higher chances of failures.
  - Use **redundancy**. Replicate data across several disks.

> ☞ This is where **HDFS (Hadoop Distributed File System)** comes into play.

- **Combine data** from different sources.

> ☞ This is where **MapReduce** comes into play.

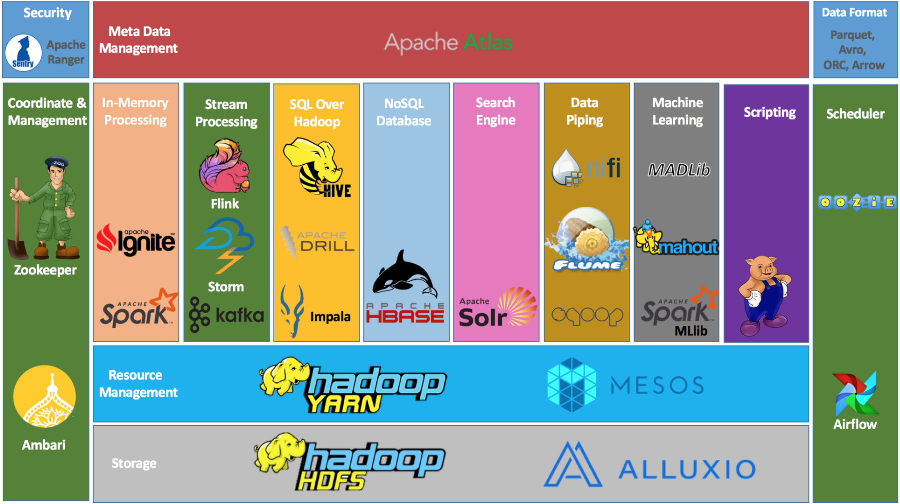
- **Apache Hadoop** is a system that provides:
  - A reliable shared **storage**: HDFS.
  - A **processing** system: MapReduce.

Apache Hadoop

Hadoop infobox

- **Key people**: Doug Cutting, Mike Cafarella.
- **Original project:** Apache Nutch, open-source web search engine.
- **Timeline:**
  - **2002.** Apache Nutch was started (crawler and search system).
  - **2003.** Google published the Google File System (GFS).
  - **2004.** Nutch implemented GFS as the Nutch Distributed File System (NDFS).
  - **2004.** Google published MapReduce.
  - **2005.** Nutch integrated its own implementation of MapReduce.
  - **2006.** NDFS and MapReduce moved out to another project: Hadoop.
  - **2008.** Hadoop used in the Yahoo! search engine.
  - **2008.** Hadoop made top-level project at Apache.
  - **2008.** Hadoop sorted 1 TB of data in 209 seconds.

Hadoop ecosystem



Source

Why do we need Hadoop?

Example (Mining weather data ▸ Source)

- We want to analyze a dataset with weather data.
- The dataset contains a file for each year (between 1901 and 2001).
- Each file contains temperature readings from different weather stations.
- **Objective.** Compute the maximum temperature for each year.

First approach:

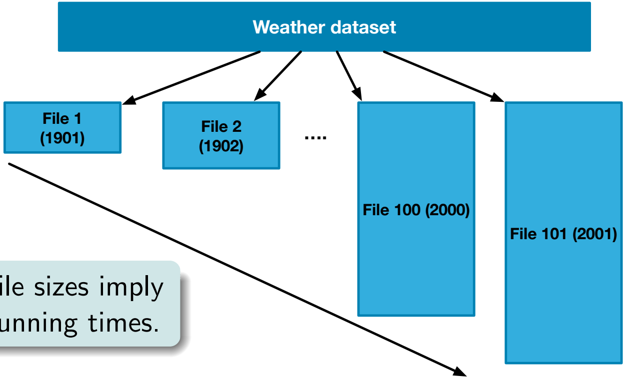
- Create a Linux script or a Python program.
- Run it on a **single machine**.

☞ The computation may take a long time depending on the dataset size.

Why do we need Hadoop?

- **Second approach:** run parts of the script in **parallel**.
  - Using all available threads on a **single machine**.

Problem: how do we split the input data?

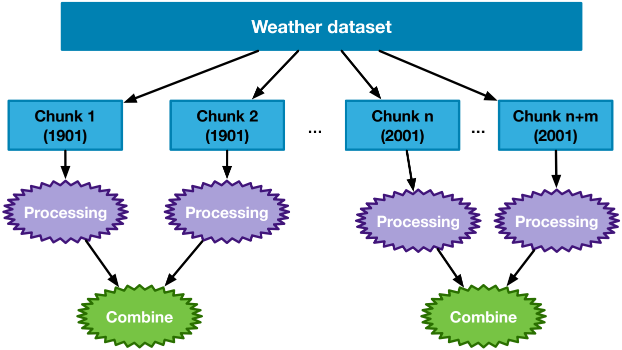


Different file sizes imply different running times.

Why do we need Hadoop?

- **Second approach:** run parts of the program in **parallel**.
  - Using all available threads on a **single machine**.

Problem: how do we split the input data?



Why do we need Hadoop?

- Many datasets cannot be handled on a single machine.
- **Third approach:** run parts of the program in **parallel** on a **cluster** of machines.

Challenges

- **Coordination**.
  - On which machine each process goes?
  - How the results are combined?
- **Reliability.**
  - What happens if some processes fail?

- Hadoop takes care of all these challenges.
- Hadoop offers programmers an **abstraction** to run **parallel programs**: **MapReduce**.

MapReduce

Definition (MapReduce)

MapReduce is a programming model for data processing that abstracts the problem from disk reads and writes transforming it into a computation over sets of **keys** and **values**. ▸ Source

- The computation consists of two parts: **map** and **reduce**.
- The same program can run on one or multiple machines.

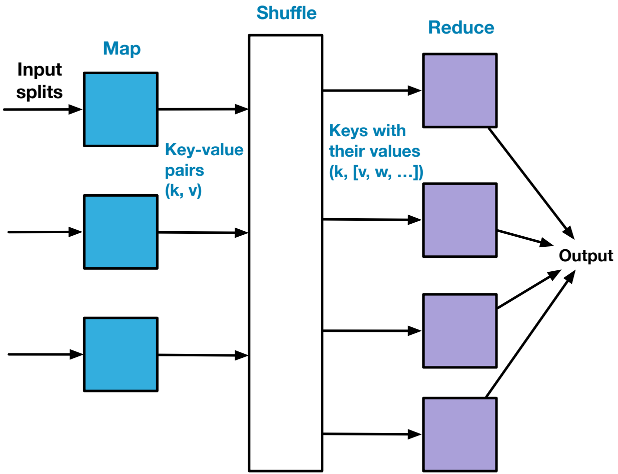
What can you do with MapReduce?

Search indexes, image analysis, graph-based problems, machine learning...

☞ MapReduce cannot solve any problem!

MapReduce: principles

- The input is divided into **splits**. In a text file, **split** = **set of lines**.
- Each split is a set of **records**. In a text file, **record** = **one line**.



Example: word count

- **Input.** A text file f (text over several lines).
- **Output.** $\{(w, o_w)\ \forall w \in f\}$, w = word, o_w = occurrences of w in f.
- We have to define the two functions **map** and **reduce**.

What we know so far

map : $\textit{line} \rightarrow$ sequence of (k, v)
reduce : $(k, L) \rightarrow$ output value

What we need to determine

- What are k and v?
- What is the output value?

☞ It's easier to determine the output value, as we already know what we want to obtain as a result of the computation!

Example: word count

What we know so far

map : *line* $\rightarrow$ sequence of (k, v)
reduce : $(k, L) \rightarrow$ output value

- The output is a pair (w, o_w).
- Therefore, the key k is a word w!

Example: word count

What we know so far

map : *line* $\rightarrow$ sequence of (k, v)
reduce : $(k, L) \rightarrow$ output value

- The output is a pair (w, o_w).
- Therefore, the key k is a word w!

Definition of map and reduce

map : *line* $\rightarrow$ sequence of (w, v)
reduce : $(w, L) \rightarrow (w, o_w)$

- What does L contain ?

Example: word count

What we know so far

map : *line* $\rightarrow$ sequence of (k, v)
reduce : $(k, L) \rightarrow$ output value

- The output is a pair (w, o_w).
- Therefore, the key k is a word w!

Definition of map and reduce

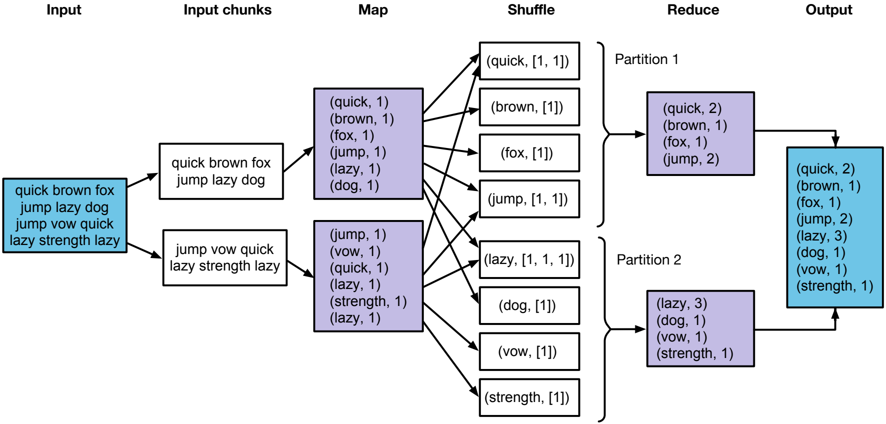
map : *line* $\rightarrow$ sequence of (w, v)
reduce : $(w, L) \rightarrow (w, o_w)$

- What does L contain ?

Definition of map and reduce

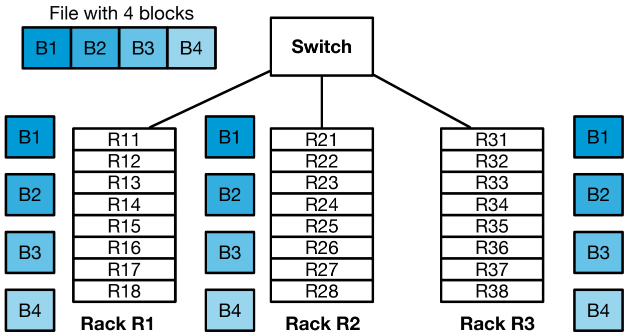
map : *line* $\rightarrow \{(w, 1)\ \forall w \in line\}$ w is a word.
reduce : $(w, L = [1, \ldots, 1]) \rightarrow (w, sum(L))$

Example: word count



MapReduce implementation: data storage

- For large inputs, we use a **cluster** of machines (**scale out**).
  - Machines in a cluster are referred to as **nodes**.
- Data is stored in a distributed file systems (e.g., HDFS).
- Each file is split into a set of **fixed-size blocks** (64 MB or 128 MB).
- Each block is **replicated** across machines (for **reliability**).



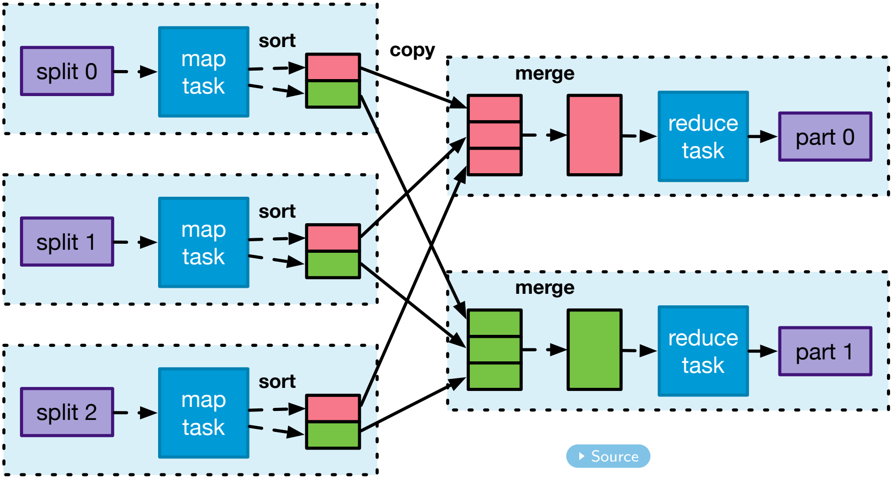
MapReduce implementation: terminology

Definition (MapReduce job)

A **MapReduce** job is a unit of work that a client application wants to be performed: it consists of the input data, the MapReduce program and configuration information. ▸ Source

- Hadoop runs a *job* by dividing it into **tasks** (*map* and *reduce* tasks).
- Hadoop divides the input data into **fixed-size** pieces called **splits**.
- Hadoop creates a **map task** for **each input split**.
- A map task runs the function *map* on each record of the split.

MapReduce implementation: data flow



Source

MapReduce implementation: data flow

- **Data locality optimization.** Whenever possible, Hadoop runs a map task on a node where the input split resides.
  - If the node is not available the map task is run on an available node in the same rack.
  - If no node in a rack is available, the map task is run on a node in another rack.
- The output of map task is stored to the local disk, not HDFS.
  - The output of map is temporary and doesn't need to be replicated.
- Reduce tasks don't have the advantage of data locality.
  - The different partitions are transferred from the nodes where the map tasks are running.

Combiner functions

- **Problem.** Lots of data is transferred on the network between map and reduce tasks.
- **Solution.** Use a **combiner function**.

Combiner function

Input. The output of the **map** function grouped by key.
Output. The input where values associated with the same key are combined somehow.

Example (Word count)

First map produces (cat, 1), (cat, 1), (cat, 1).
Second map produces (cat, 1), (cat, 1).
The reduce is called on (cat, [1, 1, 1, 1, 1]).
The combiner produces (cat, 3) and (cat, 2) respectively.
The reduce is called on (cat, [3, 2]).

Combiner functions

Word count

map: $line \rightarrow \{(w, 1)\ \forall w \in line\}$ w is a word.
combine: $(w, c_w = [1, \ldots, 1]) \rightarrow (w, sum(c_w))$.
reduce: $(w, c_w = \{o_{1,w}, \ldots, o_{m,w}\}) \rightarrow (w, sum(c_w))$ $o_{i,w}$: occurrences of w in one or more lines.

☞ We can use a combiner only if the function that we want to implement is **commutative** and **associative**.

☞ The combiner function doesn't replace the reduce function.

References

- White, Tom. *Hadoop: The definitive guide*. "O'Reilly Media, Inc.", 2012. Click here