



Rapport du MOOC

# Big Data Essentials: HDFS, MapReduce and Spark RDD

Author: Mohamed Sneiba HAMOUD

Supervisor: Nicolas BRUNEL

# Table of Contents

# Why I chose this MOOC

I have chosen this MOOC to expand my knowledge and skills in the field of data engineering and data analysis on large scale. In this course some of the most used technologies in the world in terms au large scale data processing are introduced. Learning how to use them would be very beneficial.

# Organization and specificities of this course

This course is part of the "Big Data for Data Engineers Specialization" program offered by Yandex (a russian multinational corporation specialized in Internet related products and services) in which i enrolled to.
The specialization includes 5 courses:

- Course 1: **Big Data Essentials: HDFS, MapReduce and Spark RDD**
- Course 2: **Big Data Analysis: Hive, Spark SQL, DataFrames**
- Course 3: **Big Data Applications: Machine Learning at Scale**
- Course 4: **Big Data Applications: Real-Time Streaming**
- Course 5: **Big Data Services: Capstone Project**

After finishing each course and the completion of the hands-on project, a certificate is delivered.

The first course requires 6 weeks of study with 6-8 hours/week.

The "Big Data Essentials" course is divided into 6 weeks and is organized as follow:

1. Week 1: Basic command line interface and introduction to HDFS.
2. Week 2: Solving problems with MapReduce
3. Week 3: Practice week
4. Week 4: Introduction to Apache Spark
5. Week 5: Practice week
6. Week 6: Real-World Applications

# Introduction

Big Data analytics is the often complex process of examining large and varied data sets to uncover information including hidden patterns, unknown correlations, market trends and customer preferences that can help organizations make informed business decisions.
Big Data is historically associated with three concepts:

1. **Volume**: The volume of data stored is expanding rapidly: digital data created worldwide is estimated to have increased from 1.2 zettabytes per year in 2010 to 1.8 zettabytes in 201143, then 2.8 zettabytes in 2012 and will rise to 40 zettabytes in 2020.
2. **Variety**: The type and nature of the data is sometimes heterogenous. This adds a new challenge because mainstream relational databases are not well suited for unstructured or semi-structured data.
3. **Velocity**: It represents the frequency with which data is generated, captured, shared and updated.

The objective of this course is to:

- Learn some basic technologies of the modern Big Data landscape, namely: HDFS, MapReduce and Spark.
- Learn about distributed file systems, why they exist and what function do they serve.
- Grasp the MapReduce framework, a workhorse for many modern Big Data applications.
- Apply the framework to process texts and solve sample business cases.
- Learn about Spark, a computational framework.
- Build a strong understanding of Spark basic concepts.

# Unix Command Line Interface (CLI)

The prerequisite for this lesson is to have a minimum knowledge of Linux command line and file system management. The first part of week 1 was dedicated to a refresher on the use of the bash interpreter, on Linux process management and on classic file systems.

# Hadoop Distributed File System (HDFS)

**Hadoop** is an open-source framework which is dedicated to facilitating the creation of distributed applications.

The Hadoop core consists of a storage part: HDFS (Hadoop Distributed File System), and a

processing part called MapReduce. Hadoop splits the files into large blocks and distributes them through the nodes of the cluster. To process the data, Hadoop transfers the code to each node and each node processes the data it has. This allows all data to be processed more quickly and efficiently than in a more traditional supercomputer architecture based on a parallel file system where calculations and data are distributed over high-speed networks.

## Distributed File Systems, HDFS Architecture and Scalability problems

In order to store a high quantity of data, two approaches are possible: Horizontal scaling and Vertical Scaling.

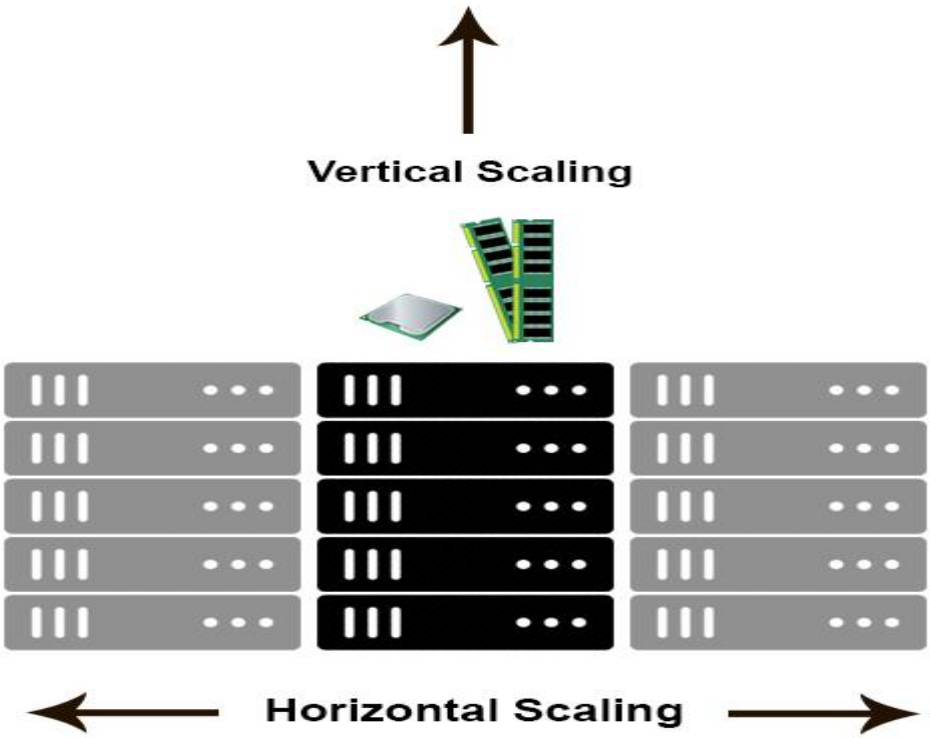


*Figure 1 Horizontal scaling vs Vertical Scaling*

Horizontal scaling means that you scale by adding more machines into your pool of resources whereas Vertical scaling means that you scale by adding more power (CPU, RAM) to an existing machine.

When having a very quantity of data to store, the horizontal scaling approach is by far the most used because of its fault-tolerant, flexible and scalable. Using the horizontal scaling adds more complexity and it comes with an overhead in the form of cluster setup. In order to manage that, we inevitably need a distributed file system to access and store data across multiples nodes.

## HDFS Architecture

Hadoop HDFS architecture consists of a Master/Slave architecture in which the master is NameNode that stores meta-data and the slave is DataNode that stores the actual data. HDFS Architecture consists of single NameNode and all the other nodes are DataNodes.

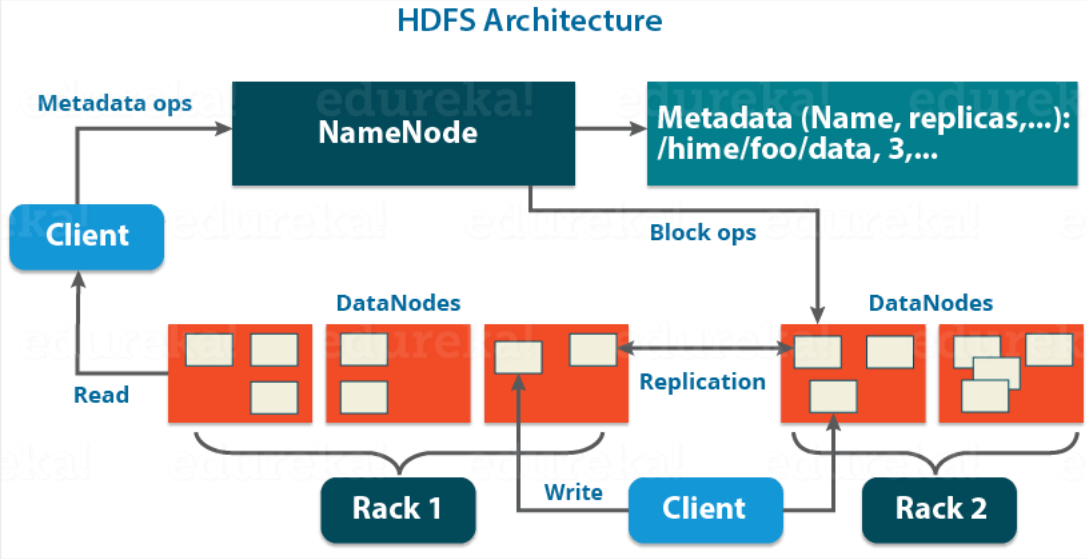


*Figure 2. HDFS Architecture*

In the figure above, we can notice that Hadoop splits large files into chunks of data of the same size called *blocks*. These blocks are placed into different data nodes. In order to guarantee fault-tolerance, HDFS will duplicate the blocks multiple times and store it into different datanodes.
This will allow to launch a recovery process when a block is lost due to a data node failure. With this mechanism HDFS provides a very robust fault-tolerant system.

## Rack Awareness

In order to improve the network traffic while reading/writing HDFS file, the NameNode of the cluster takes into account the topology of the cluster to decide in which DataNode to store each block and replica. In fact the NameNode makes sure that all replicas and blocks are not stored on the same rack. This allows HDFS to maintain a high availability of the data but also to improve the cluster's performance.

## HDFS Write Operation

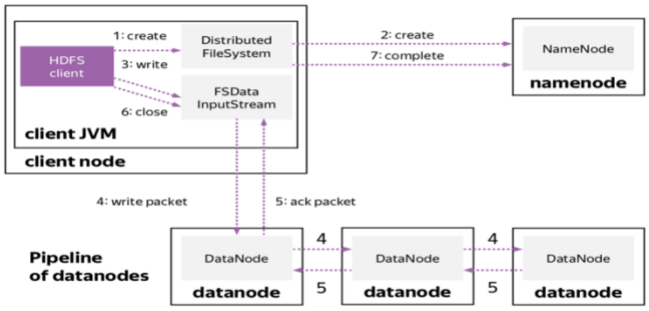


*Figure 3. Write operation*

When a client wants to write a file to HDFS, it communicates to NameNode for metadata. The NameNode responds with a number of blocks, their location, replicas and other details. Based on information from the NameNode, the client splits the file into multiple blocks. After that, it starts sending them to first DataNode. The client first sends block A to DataNode 1 with other two DataNodes details. When DataNode 1 receives block A from the client, DataNode 1 copy same block to DataNode 2. DataNode 2 copies the same block to DataNode 3.
When the DataNode receives the blocks from the client, it sends the write confirmation to the NameNode.
The same process is repeated for each block of the file.

**HDFS Read Operation**

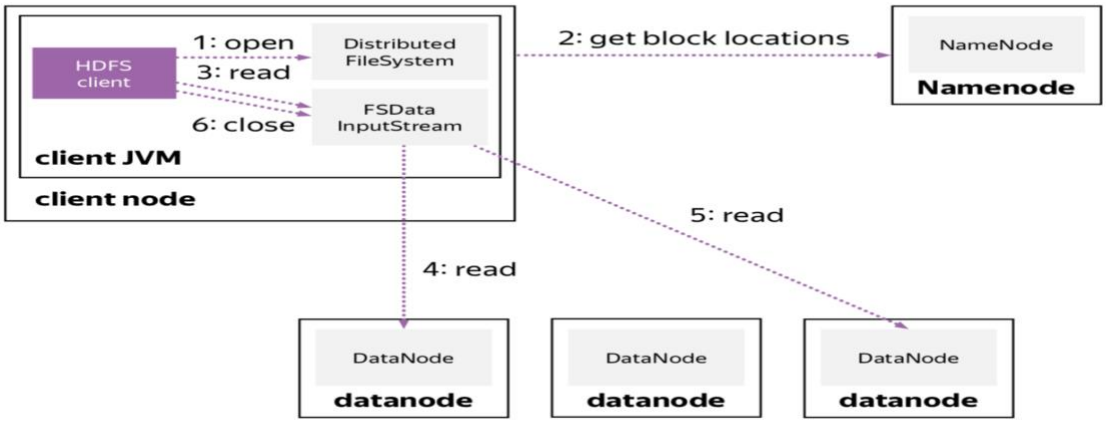


*Figure 4. Read Operation*

To read from HDFS, the first client communicates to NameNode for metadata. A client comes out of name node with the name of files and its location. The NameNode responds with number of blocks, their location, replicas and other details.
Now client communicates with the DataNodes. The client starts reading data in parallel from the DataNodes based on the information received from the NameNode. When the client or application receives all the block of the file, it combines these blocks into the form of an original file.

## Data Modeling and File Formats

Most often in Big Data, data comes in a unstructured format. Usually this mean that the data is "not structured enough for a task" for example:

Ex1: Logs = Line per request with all related data
Ex2: Video = Sequence of frames

In order to improve our applications performance, correctness, computation complexity and resource usage, we have to correctly decide on a ***data model*** and a ***storage format***.

A data model is a way we have to think about our data elements and ask ourselves the following questions about our data in the data model decision making process:

- What are they ?
- What domain they come from ?
- How different elements relate to each other ?
- What are they composed of ?

There is many different data models for example:

- Relational data model
- Graph data model

A file format defines the physical data layout and different choices of file format can lead to tradeoffs in complexity. The primary function of a file format is to make transformations between raw bytes and programmatical data structures.

There is many different file formats and they can differ in:

- Space efficiency
- Encoding and decoding speed
- Supported data types
- Splittable/monolithic structure
- Extensibility

# Apache Hadoop MapReduce

MapReduce is a programming model specifically designed to read, process and write very large volumes of data. Programs adopting this model are automatically parallelized and executed on computer clusters. MapReduce consists of two functions ***map()*** and ***reduce()***.

"***map***" is the name of the function that applies a given operation to each of the elements in a list and returns a list. For example:

```
>>> map(lambda x: x*x, [1,2,3,4])
[1,  4,  9,  16]
```

"***reduce***" is the name of the function that applies a given function to all items in a list and returns a single list. For example:

```
>>> reduce(operator.sum, [1,  4,   9,   16])
>>> reduce(operator.sum, [5 = 1 + 4,  9,  16])
>>> reduce(operator.sum, [14,   16])
>>> 30
```

Apache Hadoop MapReduce implements the following functionalities:

- Automatic parallelization of Hadoop programs:
  - HDFS is responsible for data distribution and replication
  - The master divides the work into parallel jobs and distributes them
  - The master collects the results and manages node failures
- Transparent management of the distributed mode.
- Fault tolerance

In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machine instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount.

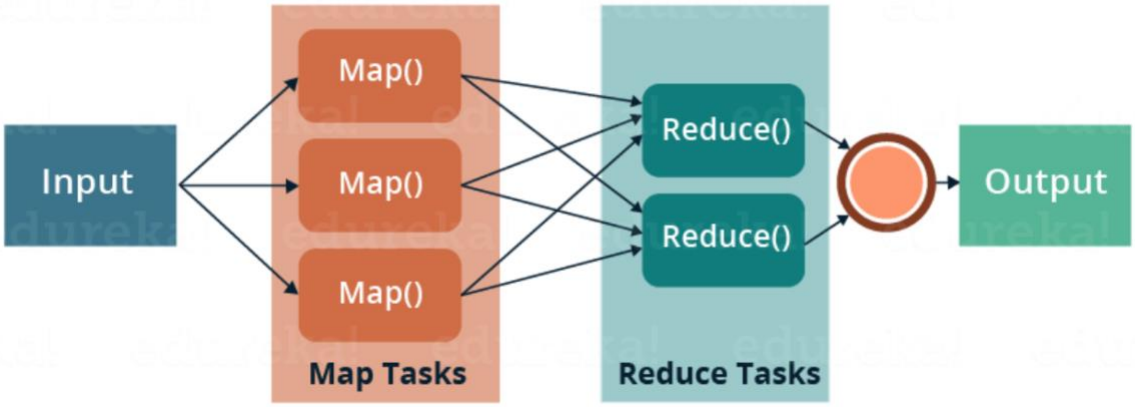


*Figure 5. MapReduce Architecture*

In order to get a much better understanding of the MapReduce paradigm, let's see a typical use case. We have a very large text file that contains some words and we would like to compute the number of occurrences of each word. This is how the workflow of the program will look like:

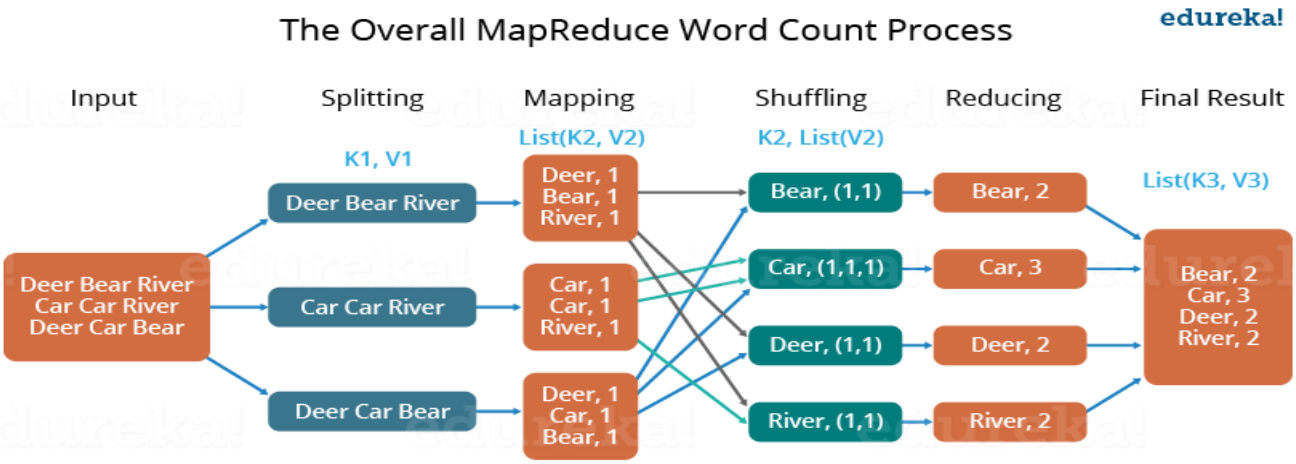


*Figure 6. Word count MapReduce process*

- First, we divide the input in three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mapper and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing, but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After mapper phase, a partition process takes place where sorting and shuffling happens so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1], etc.
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

## Hadoop MapReduce streaming applications in Python

In this course, we used the Python programming language in streaming mode. In fact Hadoop provides a possibility to write mappers and reducers in any language, which is called streaming. We will use YARN (Yet Another Resource Negotiator) as the Hadoop cluster manager. Yarn helps launch our applications in streaming. Let's image we want to count the number of lines in a large file.

Our input file is a wikipedia article and has the following format:


*Figure 7. File format*

Here is the python code of the mapper and the reducer:

```
import sys
import re

reload(sys)
sys.setdefaultencoding('utf-8') # required to convert to unicode

for line in sys.stdin:
    try:
        article_id, text = unicode(line.strip()).split('\t', 1)
    except ValueError as e:
        continue
    words = re.split("\W*\s+\W*", text, flags=re.UNICODE)
    for word in words:
        print >> sys.stderr, "reporter:counter:Wiki stats,Total words,%d" % 1
        print "%s\t%d" % (word.lower(), 1)
```

*Figure 8. mapper.py*

```
for line in sys.stdin:
    try:
        key, count = line.strip().split('\t', 1)
        count = int(count)
    except ValueError as e:
        continue
    if current_key != key:
        if current_key:
            print "%s\t%d" % (current_key, word_sum)
        word_sum = 0
        current_key = key
    word_sum += count

if current_key:
    print "%s\t%d" % (current_key, word_sum)
```

*Figure 9. reducer.py*

```
yarn jar $HADOOP_STREAMING_JAR \
        -files mapper.py,reducer.py \
        -mapper 'python mapper.py' \
        -reducer 'python reducer.py' \
        -input /data/wiki/en_articles \
        -output word_count
```

*Figure 10. Yarn command for streaming*

We can notice that with Hadoop we can launch MapReduce application in streaming easily by specifying the mapper, the reducer, the input file and the output file.

There is more MapReduce application examples in my git repository for this courser:

https://github.com/sneibam/MOOC4-BigDataEngineering-Coursera-Yandex

# Apache Spark

Apache Spark is an open-source framework of distributed computing. It is a set of tools and software components structured according to a defined architecture. The framework is a Big Data processing application framework for performing complex analyses on large scale.

## Spark Ecosystem

Apache Spark contains multiple components on top of him:

### *Spark SQL*

Spark SQL allows us to execute queries in SQL languages to load and transform data. The SQL language is derived from relational databases, but in Spark, it can be used to process any data, regardless of its original format.

### *Spark Streaming*

Basically, across live streaming, Spark Streaming enables a powerful interactive and data analytics application. Moreover, the live streams are converted into micro-batches those are executed on top of spark core.

### *Spark MLlib*

It is machine learning library delivers both efficiencies as well as the high-quality algorithms. Moreover, it is the one of the most used frameworks by data scientist, since it

is capable of in-memory data processing, that improves the performance of iterative algorithm drastically.

### *Spark GraphX*

Basically, Spark GraphX is the graph computation engine built on top of Apache Spark that enables to process graph data at scale.

## RDD (Resilient Distributed Datasets)

At the core of Apache Spark is the notion of data abstraction as distributed collection of objects. This data abstraction enables us to make an efficient execution for a computation, and a flexible and convenient formalism to define computations. Formally, RDD is a read-only, partitioned collection of records.

## Transformations and Actions

There is two types of Apache Spark RDD operations:

- Transformations
- Actions

**Spark Transformation** is a function that produces new RDD from the existing RDDs. It takes RDD as input and produces one or more RDD as output. Each time it creates new RDD when we apply any transformation. Thus, the so input RDDs, cannot be changed since RDD are immutable in nature.

There is two types of transformations in Spark:

- Narrow transformations: all the elements that are required to compute the records in a single partition is in the single partition of parent RDD. A limited subset of partition is used to calculate the result.
- Wide transformations: In a wide transformation, all the elements that are required to compute the records in the single partition may be in many partitions of parent RDD.

The list of the transformations is in the Spark official documentation:

https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

An action is one of the ways of sending data from Executer to the driver. Executors are agents that are responsible for executing a task. While the driver is a JVM (Java Virtual Machine) process that coordinates workers and execution of the task.

The list of Spark actions us in the Spark official documentation:

https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions

## Conclusion

In short, this course allowed me to acquire basic knowledge in the field of Big Data and skills on the tools most used today on the market. After this lesson, i will focus on finishing the other lesson of the specialization in order to get all the certification.

My assignments and the slides of this lesson are all on my git repository.

https://github.com/sneibam/MOOC4-BigDataEngineering-Coursera-Yandex

# References


Hadoop HDFS Architecture Explanation and Assumptions:
https://data-flair.training/blogs/hadoop-hdfs-architecture/

Apache Hadoop: Wikipedia Article
https://en.wikipedia.org/wiki/Apache_Hadoop

Spark Programming Guide
https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html

Spark Tutorial – Learn Spark programming
https://data-flair.training/blogs/spark-tutorial/

Apache Spark: Wikipedia Article
https://en.wikipedia.org/wiki/Apache_Spark

MapReduce: Wikipedia Article
https://fr.wikipedia.org/wiki/MapReduce

Fundamentals of MapReduce With MapReduce Example
https://www.edureka.co/blog/mapreduce-tutorial/