# Cloud Computing Lab Session 2 Report

Hichem Lamraoui[1], Ahmed Haj Yahmed[1], Bouchoucha Rached[1], Foalem Patrick Loic[1]
[1] *Dept. of Computer Engineering, Polytechnique Montreal, Canada*

November 4, 2022

**Abstract**

The objective of this task is to become acquainted with the MapReduce programming paradigm by (1) utilizing several commonly used technologies such as Hadoop and Spark and (2) solving fundamental and advanced problems. We were instructed in this lab session to first gain hands-on experience running MapReduce on AWS and to explore two well-known big data tools: Hadoop and Spark. Then, we solved a classical word count problem using the MapReduce paradigm and compared the performance of Hadoop, Linux, and Spark on AWS. Finally, utilizing the MapReduce paradigm, we employed Hadoop to tackle an advanced problem known as "the social network problem".
In this report, we discuss how we implemented the MapReduce paradigm using Hadoop and Spark, how we compared the performance of Hadoop, Linux, and Spark, how we used MapReduce jobs to solve the social network problem, and what conclusions and observations we found after each experiment.

## 1 Introduction

MapReduce [1] is a programming paradigm and implementation for processing large data sets on a cluster using a parallel, distributed schema. The model is a variant of the split-apply-combine data analysis technique, and it is commonly used because it allows for easy-to-scale data processing across several computer nodes. Hadoop [2] and Spark [3] are two popular big data solutions that leverage the MapReduce paradigm. Apache Hadoop is a set of open-source software tools that helps address problems involving big volumes of data whereas Apache Spark is an open-source engine for big data processing. Hadoop is built to handle batch processing effectively. Spark is intended to handle real-time data in an efficient manner.

Our second lab session's goal is to gain hands-on experience with the MapReduce paradigm and its accompanying commonly used big data technologies. To be more specific, we begin by investigating MapReduce on AWS with Hadoop and Spark. We initially used the MapReduce paradigm to solve a first classical problem, the word count problem, and compared the performance of Hadoop, Linux, and Spark on AWS. Then, we used Hadoop to solve a more

advanced problem known as "the social network problem". The idea is to implement a "People You Might Know" social network friendship algorithm that recommends two people connect with each other if they have a large number of mutual friends. Finally, in this report, we reported our findings and observation of each experiment.

The remainder of our report is arranged as follows: Section 2 introduces the technologies used, Section 3 presents the approach used to fulfill the lab objectives, and Section 4 provides the entire findings of our experiments. Section 5 brings the lab report to a close.

# 2 Background

In the following, we briefly describe the technologies used during this lab assignment.

## 2.1 MapReduce:

MapReduce [1] is a programming model for processing large amounts of data on a cluster using a parallel, distributed algorithm. The model is a variant of the split-apply-combine data analysis technique. A MapReduce program is made up of a mapping technique that filters and sorts data, and a reduction method that performs summary operations. The MapReduce System is in charge of orchestrating processing by conducting several jobs in parallel, handling all connections and data transfers between system components, and providing redundancy and fault tolerance. MapReduce libraries have been built in a variety of programming languages with varying degrees of optimization. Apache Hadoop and Apache Spark are two of the most popular of these libraries.

## 2.2 Apache Hadoop:

Apache Hadoop [2] is a suite of open-source software products that allow you to tackle issues requiring enormous volumes of data and processing by leveraging a network of many machines. The core of Apache Hadoop is made up of a storage component known as Hadoop Distributed File System (HDFS) and a processing component that uses the MapReduce programming style. Hadoop divides files into large blocks and distributes them among cluster nodes. It then distributes packed code to nodes, which process the data in parallel. This method makes use of data locality, in which nodes alter the data to which they have access.

## 2.3 Apache Spark:

Apache Spark [3] is an open-source data processing engine. It provides a programming interface for clusters with implicit data parallelism and fault tolerance. For quick analytic queries against massive data, it uses in-memory caching and improved query execution. Spark may be deployed in a variety of approaches, and it supports SQL, streaming data, machine learning, and graph processing. It also has native bindings for Java, Scala, Python, and R programming languages. Spark applications are made up of two major components;

A driver turns the user's code into many tasks that may be spread across worker nodes, and executors operate on those nodes and execute the tasks assigned to them. To arbitrate between the two, some sort of cluster manager is required.

# 3 Approach

This section outlines the tests we ran and the tools, and settings we used. It's important to note that we test each scenario three times so that we can get consistent results and get rid of noise in our studies.

## 3.1 Experiments with Word Count

The goal here is to calculate the frequency of each word in a given dataset using different tools. First, we compare the performance (real execution time) between Hadoop and Linux and second, we compare it between Hadoop and Spark.

### 3.1.1 Hadoop vs. Linux

The first experiment consists of counting the frequency of words in *James Joyce's Ulysses* book using Hadoop and Linux. For Hadoop, we ran a provided Java program (WordCount.java) that applies the MapReduce paradigm. For Linux, we ran an equivalent command to count word frequency:

```
tr <./input/pg4300.txt -c "[:graph:]" "" | sort | uniq -c
```

Where *pg4300.txt* is the text file containing the *James Joyce* book. Next, we compared the performance between the two tools by recording the real execution time.

### 3.1.2 Hadoop vs. Apache Spark

In the second experiment, we ran the same Java program (WordCount.java) but this time on Hadoop and Apache Spark. The execution is repeated three times for each dataset (we were provided with 9 different datasets) and for both tools. Then, we compared the performance between them by recording the average of real execution time.

## 3.2 Problem: The Social Network Friendship

**Problem Statement :** The challenge involves recommending new friends to a person based on the number of mutual friends between them. In other words, if two individuals share several friends but are not friends themselves, our recommander should inform each individual that they can connect and become friends. In fact, the recommendation system should be able to rank the list of recommanded friends depending on the number of mutual friends shared by they have.
**The goal :** Our objective is to use MapReduce in the Hadoop environment to provide to each individual a ranked list of people with whom they share mutual friends. The data used to solve this problem is a large file with each line comprising a person and a list of his friends in the format [Person][Tab][List of Friends]. The objective will be to specify how the

mapper and reducer of the Mapreduce technique should function in order to generate the list of mutual friends.

### 3.2.1 Mapper :

The mapper will receive each line of the file describing a person and his friends and transform it into many key-value data chunks. The data generated by the mapper will be sent to the reducer so that it can aggregate the data and provide the final results.
The inspiration for the mapper we created came from graph theory, where we related people in our data through types of edges, such as friends and mutual friends. For each line in the file, the mapper will execute the two steps below:

- In the first step, we extract individuals with a friendship relationship. For each line, we build numerous key-value pairs consisting of the [Person] as the key (noted as P) and a string formed of each element in the [List of Friends] (noted as $F_i$) followed by a flag equal to "0" to indicate the relation friend. In other words, the final list of key values obtained will have the following format: $\{(P, F_i\_0), F_i \in [List\ of\ Freinds]\}$. We didn't consider the other path $((F_i, P\_0))$ as we assume the dataset has an entry where it put $F_i$ as a friend with $P\_0$ (thus this relationship will be created there).

- The second phase involves the extraction of the list of mutual friends. To accomplish this, we'll relate each pair of elements in the [List of Friends] (we'll refer to them as $F_i$ and $F_j$) and create two relations that reflect that $F_i$ is a mutual friend with $F_j$ and the inverse, $F_j$ is a mutual friend with $F_i$, and we'll use the flag "1" to indicate the mutual freinds relation. This step's concluding data consists of numerous key-value pairs in the following formats: $\left\{(F_i, F_j\_1), (F_i, F_j) \in [List\ of\ Freinds]^2\right\}$.

### 3.2.2 Reducer :

The reducer will receive key-value chunks from the mapper. The key symbolises the friend $P$ to whom we wish to promote friendship. There are two sorts of relationships represented by the values: friendship and mutual friendship ($(F_i, 0)$ and $(F_i, 1)$). The mapper performs the following tasks:

- We loop over all the relationships and get the friend id $F_i$ and its relationship with $P$: friends (0) or mutual friends (1).

- We then build a dictionary in which we list each $F_i$ and its connection to $P$.

- $F_i$ will always have 0 as a value if he has a 0 relationship, even if he has other 1 relationships. Otherwise, his value will be the sum of his 1 relationships.

- Finally, we sort the dictionary in decreasing order and provide the list of friends whose values are not zero.

# 4 Results

In this part, we examine and analyze the results of each experiment.

## 4.1 Testing Environment

### 4.1.1 Azure:

We manually created an azure virtual machine of type Standard_D2s_v3, in which we installed Ubuntu 20.04 LTS. After a connection via SSH in the instance, we copy the following files setup.sh, WordCount.java, then FriendSocialNetwork.java and soc-LiveJournal1Adj.txt using the command `scp`. Once the copy of these files is finished we run setup.sh which contains all the necessary script to perform the different benchmark and to execute our program for the friend social network problem then we obtain two files in output one containing the different execution times for Wordcount and the other the list of friends for the given IDs. The results of this experiment are shown in Figure 2, 4.

### 4.1.2 AWS:

With the help of `Botocore` library we have automated the process of creating our EC2 instance of type M4.large and installing ubuntu 22.04 LTS, then we have used the `paramiko` library to automate the process of connection via SSH, copying different files and executing the setup.sh script. The results are then analyzed and presented in Figure 1, 3.

## 4.2 Hadoop vs. Linux

The results of the first experiment show a big gap between Hadoop and Linux regarding execution time. Indeed, counting the frequency of words in the book *Ulysses* is much faster with Linux than with Hadoop. This discrepancy is explained by the fact that Hadoop is not used to its full potential. Indeed, it is only installed on a single machine (standalone mode) which prevents us from benefiting from the parallelism of the counting operations that we would have found in a distributed system. In addition, the Linux command is executed directly on the dataset line by line and it aggregates the results as it is processed, which allowed a significant gain in terms of execution time. The results obtained on AWS and Azure were similar, which reinforces their validity (see figure 1 and 2).

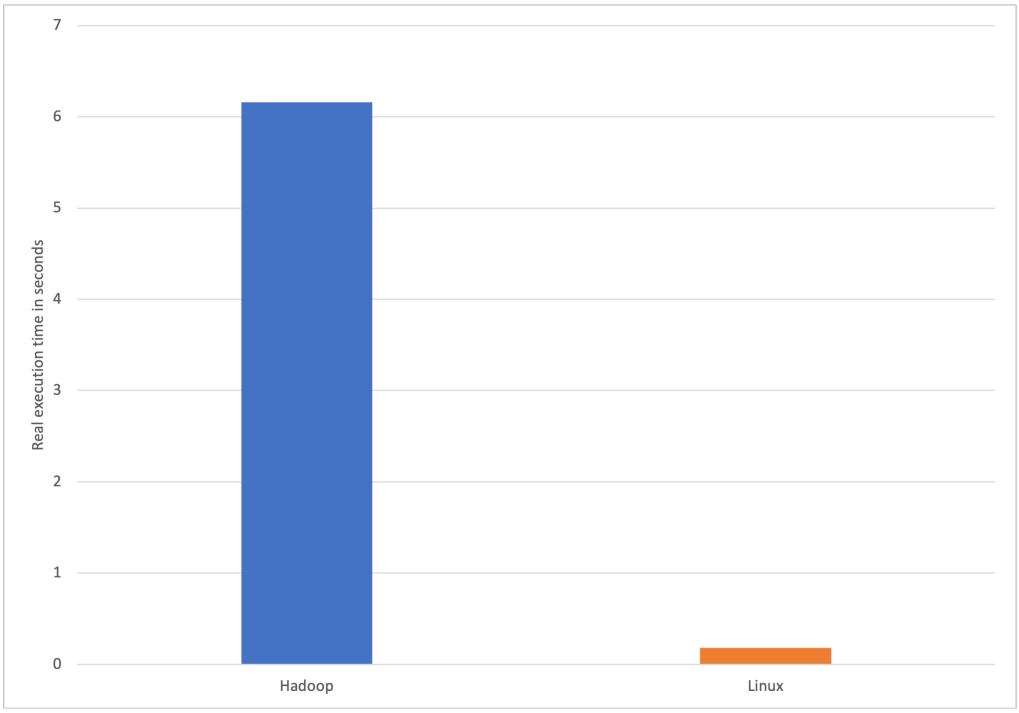


Figure 1: Comparison between Hadoop and Linux execution times on AWS.

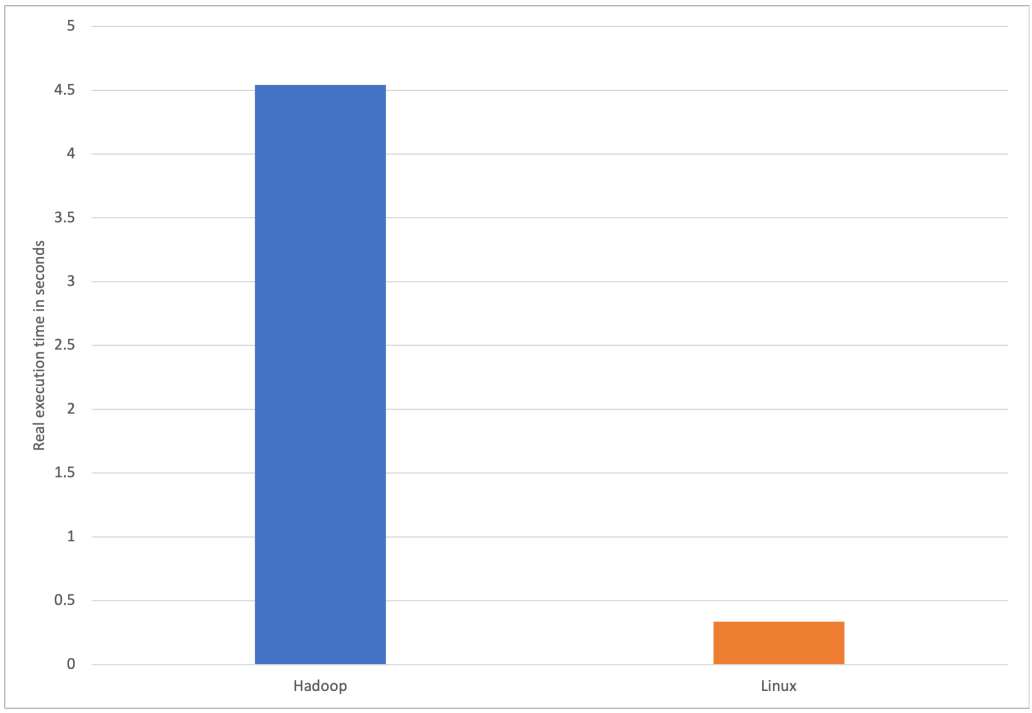


Figure 2: Comparison between Hadoop and Linux execution times on Azure.

## 4.3 Hadoop vs. Apache Spark

The results of the second experiment show a slight advantage for Hadoop compared to Apache Spark regarding execution time. This result is a bit counterintuitive where we expected that Spark would be faster than Hadoop. This is explained by the fact that Spark needs to copy the data into memory in order to process it, which requires additional time compared to Hadoop which processes the data directly from the hard disk. The results would be different if Hadoop and Spark were installed on a distributed system where the time lost in copying data into memory would be compensated by gains in processing time thanks to the parallelism of counting operations. The results obtained on AWS and Azure were similar, which reinforces their validity. Note that the datasets on the graphs are numbered from 1 to 9 respecting the same order of appearance of their download links in the statement of this laboratory (see figure 3 and 4).

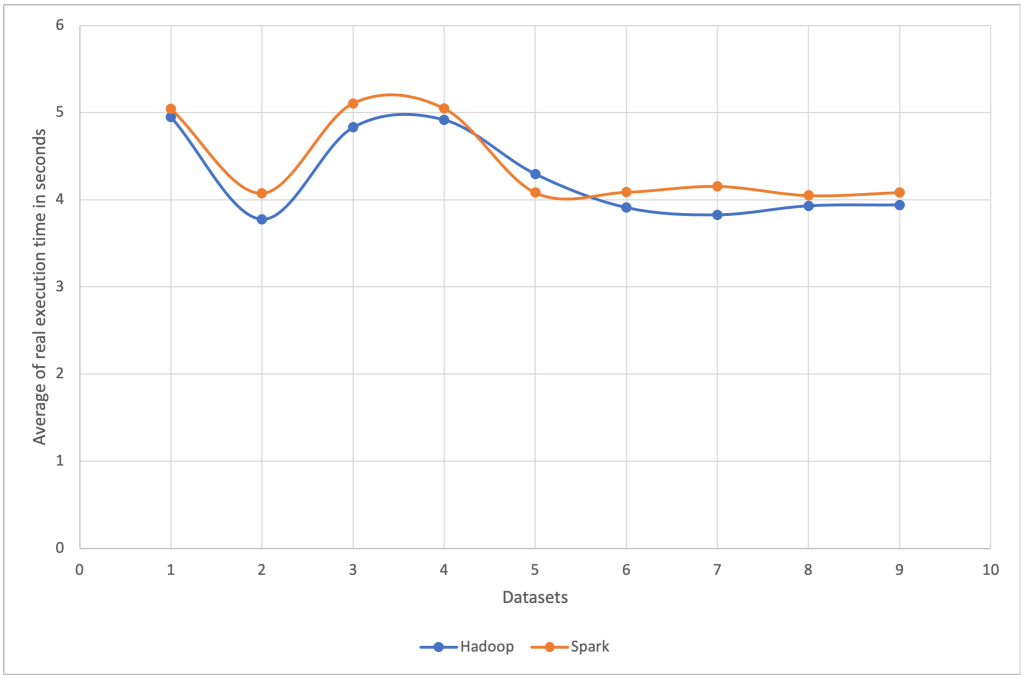


Figure 3: Comparison between Hadoop and Spark execution times on AWS.

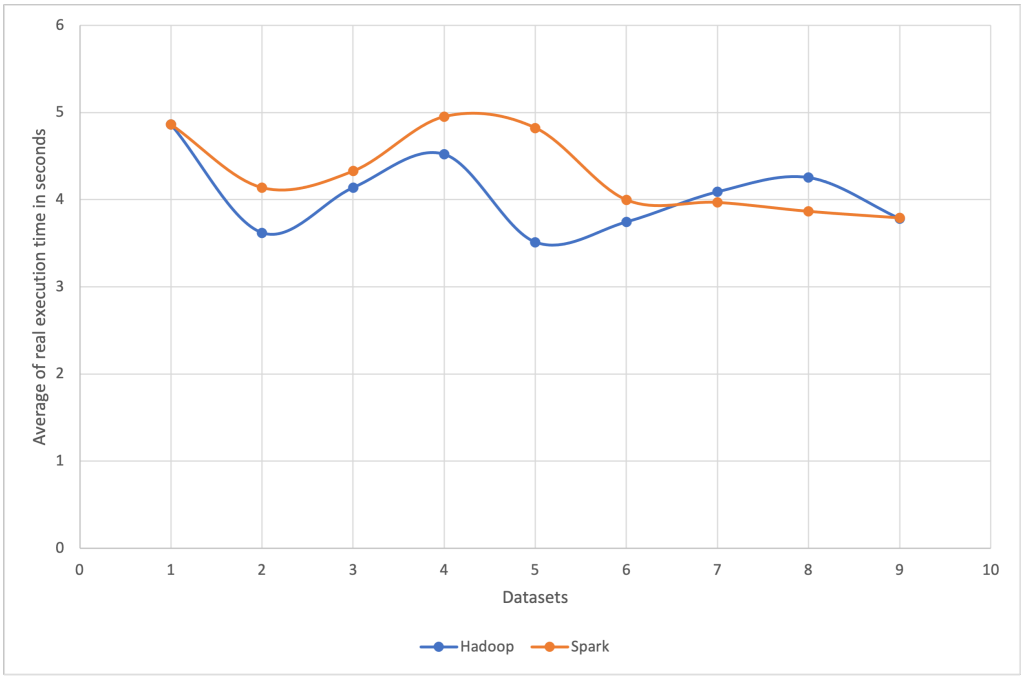


Figure 4: Comparison between Hadoop and Spark execution times on Azure.

## 4.4 Problem: The Social Network Friendship

The initial step in fixing the problem was setting the environment. Our experiments took place in two environments, namely AWS and AZURE. First, we used the same AWS environment described in the preceding section of WordCount. The enviroenement was created through an automated program that created an EC2 instance and deployed our MapReduce algorithm running on HDFS in an S3 bucket that will be accessed by the EC2 instances. Then, we put our method to the test on an AZURE B1S machine. For this section, we manually configured the machine because AWS instance creation automation was not required. Due to the large amount of lines in the file, the execution of the Social Network Friendship Mapreduce algorithm took about 2 minutes to finish all the experiment. Then, using the acquired findings, we examined the list of friends for the following user ids { 924, 8941, 8942, 9019, 9020, 9021, 9022, 9990, 9992, 9993 } which were as presented in figure 5.

| User ID | Recommended Freinds IDs |
|---|---|
| 924 | 439,2409,6995,45881,11860,43748,15416 |
| 8941 | 8944,8943,8940 |
| 8942 | 8939,8940,8944,8943 |
| 9019 | 9022,317,9023 |
| 9020 | 9021,9017,9016,9022,317,9023 |
| 9021 | 9020,9017,9016,9022,317,9023 |
| 9022 | 9019,9021,9020,9017,9016,317,9023 |
| 9990 | 13877,13134,13478,37941,34642,34485,34299 |
| 9992 | 9989,9987,35667,9991 |
| 9993 | 9991,13877,13134,13478,37941,34642,34485,34299 |

Figure 5: The results of The Social Network Friendship problem.

# 5 Conclusion

The goal of this lab is to learn how to use the MapReduce paradigm and the big data technologies that go with it. We begin by investigating MapReduce on AWS and Azure with Hadoop and Spark. We first used the MapReduce model to solve a first classical problem, the word count problem, and compared the performance of Hadoop, Linux, and Spark on AWS and Azure. Then, we used Hadoop to solve a more advanced problem known as "the social network problem". The idea is to implement a "People You Might Know" social network friendship algorithm that recommends two people connect if they have a large number of mutual friends. Finally, we reported our findings and observations from each experiment.

# References


[1] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online." in *Nsdi*, vol. 10, no. 4, 2010, p. 20.

[2] T. White, *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*, 2010.