

POLYTECHNIQUE MONTRÉAL

LOG8415 : LAB B

ADVANCED CONCEPTS IN CLOUD COMPUTING

MapReduce with Hadoop on AWS (or Azure)

Authors

Anis ZOUATENE (1963304)
Aleksandar STIJELJA (1959772)
Amin GHADESI (2121658)
Reza ROUHGHALANDARI (2153395)

November 1, 2022



1 Abstract

The programming model known as “MapReduce” facilitates concurrent processing by splitting petabytes of data into smaller chunks and processing them in parallel on Hadoop commodity servers. In the end, it aggregates all the data from multiple servers to return a consolidated output back to the application.

However, we have different ways to manage Big Data sets, such as Apache Hadoop or Apache Spark. In this paper, we will explore both software, compare their differences and evaluate their performances by conducting a few experiments.

Keywords: Big Data, Azure, MapReduce, Spark, Hadoop, Big Data.

2 Introduction

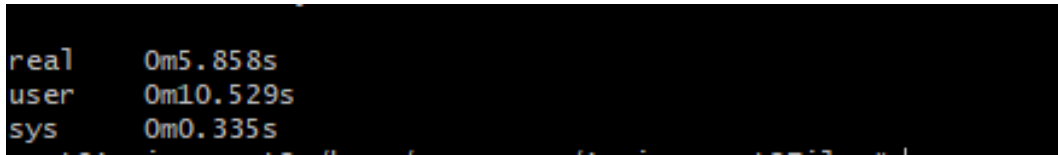
As such, in this lab, we compared the performance of the algorithm on Linux, Hadoop, and Spark with different experiments. We began by comparing their performances in a simple WordCount program and noted their differences. The WordCount program counts the occurrence of every single word in the specified document. To do this, we chose to use Azure instead of AWS. We then created a VM in it, then we used Hadoop and MapReduce to solve a social networking problem and process bigger data sets.

In this lab, we will be presenting the experiments done with the WordCount program, the performance comparison of Hadoop vs. Linux, the performance comparison of Hadoop vs. Spark on Azure, and the “people you might” know algorithm problem. In addition, we explain precisely each step we took for performing this assignment on the GitHub page.

3 Experiments with WordCount program

Before getting into the performance experiments amongst Hadoop, Spark, and Linux, we began by experimenting with the WordCount program java example given with both Hadoop and Spark. After installing the required Java packages and setting up all environment variables required, we proceeded to run the WordCount example on a copy of pg4300.txt. We made use of the TIME command to be able to extract the time performance of the run. It would provide us with the “real”, “user”, and “sys” duration of the command executed.

- **real** or **total** or **elapsed**: It is the time from start to finish of the command. In other words, it is the time from the moment we hit the Enter key until the moment the command is completed.
- **user**: The amount of CPU time spent in the user mode.
- **system** or **sys**: The amount of CPU time spent in the kernel mode.



```
real    0m5.858s
user    0m10.529s
sys     0m0.335s
```

Figure 1: The time consumption of Hadoop WordCount.java on pg3000.txt

We have to mention that we utilize real metric for our experiments and result figures. However, it seems we have to use the sum of user and sys times instead of the real time.

```

root@Assignment2:~# hdfs dfs -head output/part-0000
"Come 1
"Defects," 1
"I 1
"Information 1
"J" 1
"Plain 2
"Project 5
"Right 1
"Viator" 1
"YOU 1
#4300] 1
$5,000) 1
% 2
&c, 2
&c. 1
'46. 1
'92 1
'AS-IS' 1
'Slife, 1
'TWAS 1
'Tis 8
'Tis, 1
'Twas 5
'Twixt 1
'em. 2
'mid 1
'neath 1
'pon 1
's 3
'tis 4
'twas 4
'twas. 1
'twere, 1
("the 1

```

Figure 2: Result of Hadoop WordCount.java on pg3000.txt

Figure 1 and 2 illustrate the time and the output of Hadoop WordCount.java on pg3000.txt, respectively.

4 Performance comparison of Hadoop vs. Linux

Now that we experimented using Hadoop with the word count example talked about previously, we can now start running Hadoop WordCount on all the datasets provided and compare them with Linux. The first step was done in the last section, but for the Linux part, we do not need additional tools except regular pipelining through command line commands. For implementation, we created an Azure VM where we had to do 3 tests on each to get an average for both.

Hadoop vs Linux								
Document	Hadoop				Linux			
	1st time	2nd time	3rd time	Average	1st time	2nd time	3rd time	Average
buchanj-midwinter-00-t.txt	4.829	4.661	4.864	4.785	0.118	0.118	0.114	0.117
carman-farhorizons-00-t.txt	4.111	3.607	4.146	3.955	0.019	0.022	0.021	0.021
colby-champlain-00-t.txt	4.997	3.612	3.681	4.097	0.046	0.049	0.046	0.047
cheyneyp-darkbahama-00-t.txt	4.803	4.443	4.738	4.661	0.093	0.107	0.113	0.104
delamare-bumps-00-t.txt	3.827	3.589	3.671	3.696	0.02	0.021	0.026	0.022
charlesworth-scene-00-t.txt	4.81	4.79	4.547	4.716	0.067	0.071	0.067	0.068
delamare-lucy-00-t.txt	3.693	3.812	3.404	3.636	0.021	0.016	0.018	0.018
delamare-myfanwy-00-t.txt	3.936	3.453	3.351	3.580	0.018	0.019	0.02	0.019
delamare-penny-00-t.txt	3.945	3.726	3.785	3.819	0.013	0.017	0.013	0.014

Figure 3: Average run time of Hadoop vs. Linux on each dataset

4.1 Results

Figure 3 displays the comparison of the consumption time between Hadoop and Linux on each document. Also, for simplicity, we created a bar chart (Figure4) that shows the average times. We have to mention that all times are in the second unit. By looking at the bar chart, we can see that Linux outperformed Hadoop by a LOT. One of the reasons for this kind of result is that Hadoop needs to take time to convert the input data into HDFS format then proceed with the processing, whereas Linux directly uses the RAM to read the input line by line in order to do the process, cutting off that preparation period that Hadoop needs. This does not mean that Hadoop is inferior, because if we are dealing with large quantities of data, then we expect Hadoop to outperform easily Linux. Technically, in-memory processing is faster as no time is spent in moving the data/processes in and out of the disk, memory and catch. In addition, Hadoop does not suit for small data. (HDFS) Hadoop distributed file system lacks the ability to efficiently support the random reading of small files because of its high capacity design.

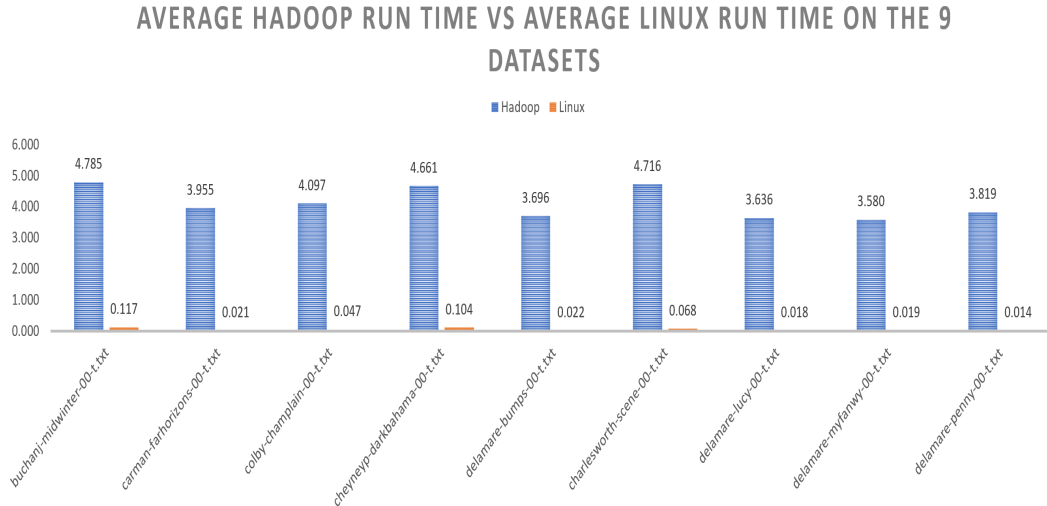


Figure 4: Result of Hadoop vs. Linux runtime

5 Performance comparison of Hadoop vs. Spark on Azure

Using the same Azur VM used previously, we used the same method that was used for Hadoop vs. Linux. As such, to properly evaluate both Hadoop and Spark, we ran the WordCount with the *time* command 3 times on each dataset for both. For the spark implementation, we used a python code which known as *SparkWordCount.py*.

On the hand, the command used for Hadoop began with “time hadoop ...”, whereas that with Spark began with “time spark-submit ...”. This information is presented in more detail in our *readme.md* file.

Hadoop vs Spark								
Document	Hadoop				Spark			
	1st time	2nd time	3rd time	Average	1st time	2nd time	3rd time	Average
buchanj-midwinter-00-t.txt	4.477	4.374	4.518	4.456	10.426	10.541	10.551	10.506
carman-farhorizons-00-t.txt	3.898	3.586	3.561	3.682	10.669	10.25	10.243	10.387
colby-champlain-00-t.txt	3.821	3.53	3.471	3.607	10.417	10.236	9.726	10.126
cheyney-darkbahama-00-t.txt	4.611	4.598	4.467	4.559	10.342	10.225	9.66	10.076
delamare-bumps-00-t.txt	3.751	3.489	3.296	3.512	10.667	9.544	9.694	9.968
charlesworth-scene-00-t.txt	4.443	3.312	4.603	4.119	9.662	9.733	9.438	9.611
delamare-lucy-00-t.txt	3.829	3.445	3.741	3.672	10.067	9.956	9.639	9.887
delamare-myfanwy-00-t.txt	3.701	3.57	3.474	3.582	10.013	10.014	10.022	10.016
delamare-penny-00-t.txt	3.452	3.452	3.406	3.437	9.489	10.025	9.929	9.814

Figure 5: Average run time of Hadoop vs. Spark on each dataset

5.1 Results

Figure 5 depicts the comparison of the consumption time between Hadoop and Spark on each document. Also, for simplicity again, we created a bar chart (Figure6) that shows the difference in average times between Hadoop and Spark software (all times are in the second unit). It is obvious amongst all datasets Spark took more time for the word counting application in comparison to the Hadoop software. In large datasets, Spark outperforms Hadoop due to the Apache Spark process and parallelization strategy. However, the situation in our assignment is different. We think this problem was due to the different steps of initialisation that Spark needs to set up before beginning to its process. But we have to mention that, Spark supports memory facility but Hadoop does have this feature.

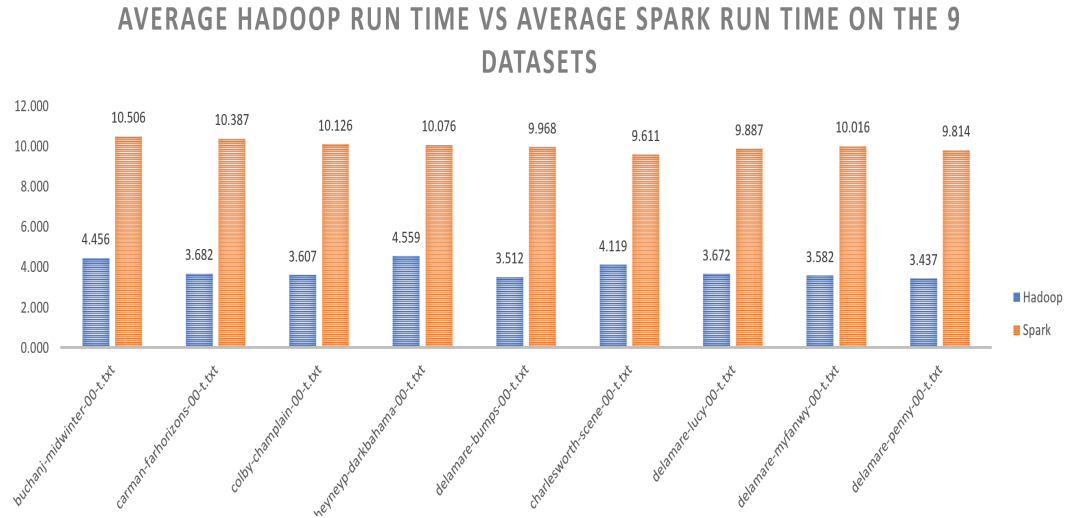


Figure 6: Result of Hadoop vs. Spark on each dataset

6 People you might know problem

In the last part of the assignment, we want to implement a simple social network friendship recommendation system. The system should suggest the right persons for each *user id*. In this case, our input is a list of users and their associated list of friends, and then we recommend to each user a list of people they might know according to their mutual friends. This could be something similar to the "suggested friends" section when you browse your Facebook feed. In the next section, we discussed a strategy we utilized for the Map and Reduce functions.

6.1 MapReduce and algorithm

The MapReduce model works based on the key-value approach. Keys are unique, and in this problem, are User IDs. Also, values are the list of User IDs of friends that each user has.

Through that, in the mapping process, we will create a key and value. key contains the User ID, and the value is a list that has potential new friends they are not already friends with, as long as they have at least one friend in common. For creating this list, we eliminated User IDs that have mutual friends with the key, in advance.

In the reducer function, we just focused on the frequency of User IDs in the value part for each key and returns ten recommenders for each User ID.

6.2 Results

The below table shows the outcome of our recommender system for each User ID.

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

Table 1: FRIEND SUGGESTIONS

7 Github repository

The following public GitHub repository includes all the files that we used. All codes have comments in themselves and in the *readme.md* file, we precisely explain each step we took for our implementations.

Link: <https://github.com/ghadesi/LOG8415—Advanced-Concepts-in-Cloud-Computing/tree/main/Assignment2>

8 Bibliography

1. Apache spark. <http://spark.apache.org>.
2. Apache spark examples. <http://spark.apache.org/examples.html>. Ac
3. Hadoop tutorial. <http://dzone.com/articles/getting-hadoop-and-running>.
4. Mapreduce tutorial. <https://hadoop.apache.org/docs/r1.2.1/mapred-tutorial.html>.