LOG8415E - Advanced Concepts of Cloud Computing

Lab 2: MapReduce on AWS

1895231 - Anis Youcef Dilmi

2224721 - Mohammed Ridha Ghoul

1954607 - Victor Kim

2225733 - Yasser Benmansour

Presented to Vahid Majdinasab

November 18th, 2023

1. Introduction

This assignment focused on three primary objectives. Firstly, we aimed to assess the processing time of a basic WordCount task implemented with Hadoop in comparison to an approach utilizing Linux-native commands and pipes. Both implementations were executed on a single AWS EC2 instance. Secondly, we delved into the comparative performance analysis of Hadoop and Spark while processing WordCount problems on the same AWS EC2 instance. Lastly, we implemented a MapReduce algorithm using Hadoop to address a Social Network problem. The outcomes of our performance evaluations, coupled with a detailed analysis, are presented in this report. Additionally, we provide comprehensive insights into the MapReduce algorithm designed for solving the Social Network problem, accompanied by instructions for the seamless execution of our code.

2. Experiments with WordCount program

Our implementation utilizes the Terraform script, **main.tf**, to coordinate the deployment of an AWS EC2 instance configured with the m4.large instance type. After successfully creating the EC2 instance, we proceed to clone the repository **https://github.com/decorJim/log8415-tp2.git**, acquiring all necessary data for the new instance.

Subsequently, we systematically execute a series of bash scripts on the EC2 instance, covering a spectrum of computational tasks related to Hadoop, Linux, Spark, and addressing the social network problem. The execution of these scripts generates results that are stored in corresponding .txt files. This methodical process allows us to scrutinize and interpret the outcomes of each specific computational task, contributing to a comprehensive understanding of the performance and capabilities of the implemented technologies. Spark and Hadoop WordCount algorithms are very similar in their execution, they start by splitting each line at each whitespace, so the result is a single list of words. Then, they map each word to a tuple containing the word and the number 1. Finally, it reduces the tuples summing up the counts for each word which is the MapReduce algorithm. The Linux algorithm removes all punctuation, then split the remaining words at the whitespaces then puts them into a single column and count the occurrences of every identical line.

Performance comparison of Hadoop vs. Linux on EC2

In this section, we're running an experiment where we are timing the execution of a Hadoop MapReduce WordCount Java program that counts all occurrences of all words in a text file, we also time the execution of the same task with a simple Linux command. Then we compare the execution times of both methods. The experiment is made on an EC2 instance of type m4.large like mentioned before, the instance is running

Ubuntu 22.04. After the experiment, the chrono times of both methods are outputted to a text file so we can compare them in the following graph:

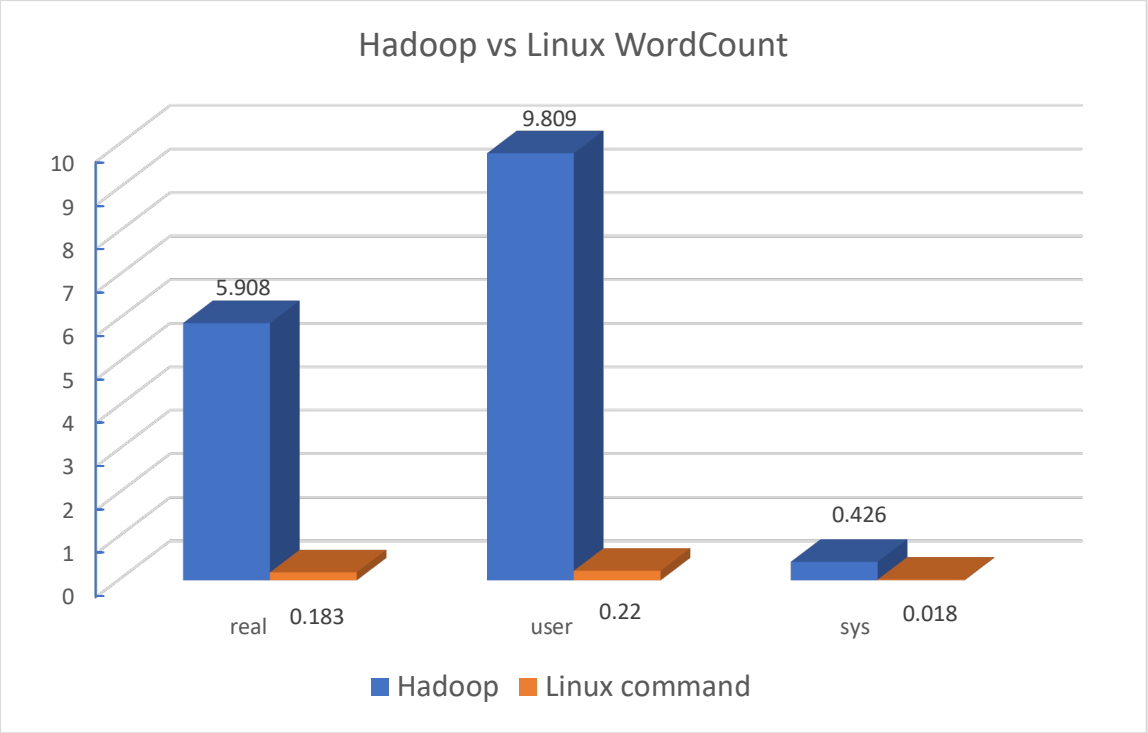

Figure 1: Hadoop vs Linux

The results show that the Linux command is a lot faster than Hadoop, also, most of the elapsed time is executing instructions in user mode. We can explain this result by the fact that Hadoop is not designed to work with small datasets because in this particular case it has a lot of unnecessary overhead due to Java execution on the JVM for map and reduce tasks. Since the Ulysses text we fed Hadoop is very small, we don't get interesting results because we can't benefit from the performance gain of the Hadoop framework. However, we can easily think that with a large dataset, the performance gain of Hadoop would surpass its overhead and the Linux command's performance would be less than Hadoop. The explanation of the scripts used in these experiments can be found in section 4.

Performance comparison of Hadoop vs. Spark on EC2

In our extensive examination of WordCount task execution on an AWS EC2 m4.large instance, employing both Spark and Hadoop across nine datasets, distinctive performance characteristics surfaced. Spark demonstrated an average real-time processing duration of 10.549 seconds, showcasing minimal user-mode CPU time (0.170 seconds) and efficient system-level processing (0.027 seconds). In contrast, Hadoop exhibited a faster real-time processing speed (4.190 seconds) but incurred significantly higher user-mode

CPU time (6.962 seconds) and system time (0.417 seconds). These results underscore the nuanced trade-offs between the two frameworks. Spark, with its longer processing time, showcased resource efficiency, while Hadoop excelled in rapid real-time processing at the cost of higher resource utilization. The choice between Spark and Hadoop hinges on specific project requirements, necessitating a careful balance between processing speed and resource optimization.

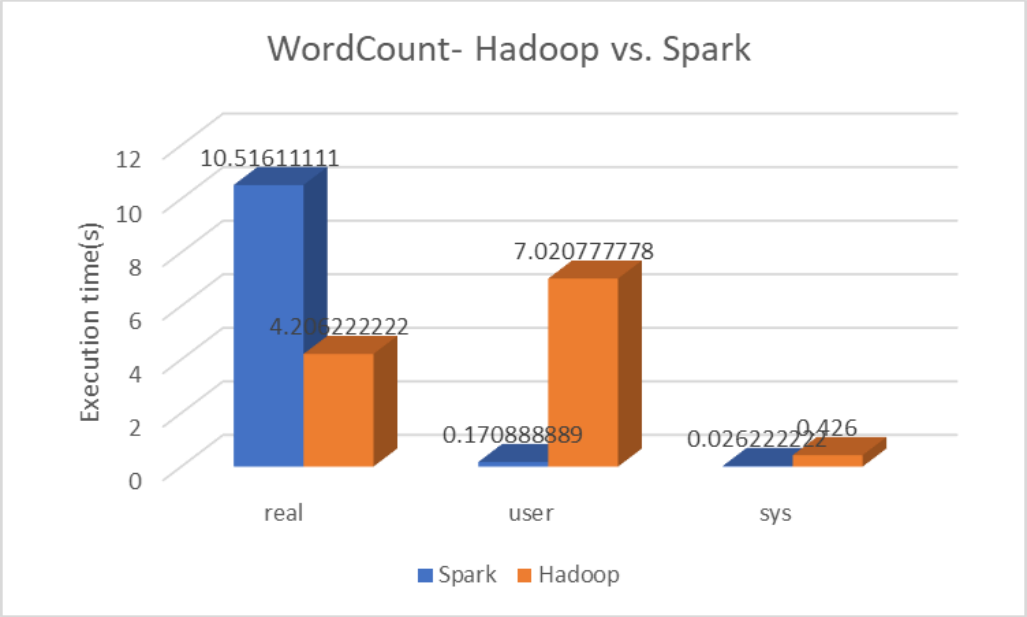

Figure 2: Hadoop vs Spark

3. Social Network Problem

The friend suggestion problem is effectively addressed through a MapReduce solution, where the dataset is divided into input splits. In our scenario, these splits consist of distinct users, serving as input records sent to the mapper. Each mapper receives a unique set of users, processes them, and forwards the results to a reducer for consolidation.

The mapper algorithm initially sends pairs for the user's existing friends, marked as "already friends." Subsequently, it generates combinations among the user's friends, recognizing that the user acts as a common friend for pairs of his friends. These combinations are then sent to a reducer for further processing.

In the reducer, intermediate key-value pairs are received, representing a user and the individuals recommended to them. Leveraging the mapper's markings, the reducer filters out recommended individuals who are already friends with the user. The reducer then maintains a record of all common friends between the user and the recommended individuals. The top 10 recommendations are selected based on the number

of common friends, and these individuals become the output, suggesting the user's top 10 potential connections with the most shared friends.

This MapReduce approach efficiently handles the friend recommendation task, leveraging the distributed processing capabilities of the framework to analyze large datasets and provide meaningful suggestions.

Friends suggested for user 924 are 439, 2409, 6995, 11860, 15416, 43748, and 45881. For user 8941, the suggestions include 8943, 8944, and 8940. User 8942 received suggestions for users 8939, 8940, 8943, and 8944. User 9019 was recommended to connect with 9022, 317, and 9023. Recommendations for user 9020 involve users 9021, 9016, 9017, 9022, 317, and 9023. User 9021 is suggested to connect with users 9020, 9016, 9017, 9022, 317, and 9023. User 9022 received recommendations for users 9019, 9020, 9021, 317, 9016, 9017, and 9023. For user 9990, the suggestions include 13134, 13478, 13877, 34299, 34485, 34642, and 37941. User 9992 is suggested to connect with users 9987, 9989, 35667, and 9991. Lastly, for user 9993, the recommendations include users 9991, 13134, 13478, 13877, 34299, 34485, 34642, and 37941.

If a user has more than 10 recommendations, the program will only keep the 10 with the highest count of common friends. If the user has less than 10 friends all the recommendations will appear.

4. Instructions to run the code.

1. To seamlessly navigate through the AWS setup and execute the log8415-tp2 scripts, follow these steps: Begin by updating the **~/.aws/credentials** file with your AWS credentials.
2. Next, clone the project repository from **https://github.com/decorJim/log8415-tp2.git** onto your local machine.
3. Move into the **log8415-tp2/infra/** directory and initialize Terraform with **terraform init, terraform plan** followed by applying it using **terraform apply**.
4. Once the instance is created in your AWS account, connect to it as a **root** user.
5. Inside the instance, clone the same project repository or use scp to secure copy the project from your local machine to the instance.
6. The following sections explain how to run the code for the 3 experiments which are Hadoop vs Linux, Hadoop vs Spark and Friend suggestions

Hadoop vs. Linux

1. To execute the Hadoop vs Linux experiment, navigate to the **log8415-tp2/hadoop_vs_linux/** directory and ensure executables permissions by running **sudo chmod +x configuration.sh** and **sudo chmod +x hadoop_vs_linux.sh**.

2. After, run the **hadoop_vs_linux.sh** script.
3. The previous command should output a file named **hadoop_vs_linux_comparison.txt** which contain the chrono time for each execution. Also, you can find in this same directory the output of the two algorithms which are the lists of words with their number of occurrences.

Hadoop vs. Spark

1. To execute the Hadoop vs Linux experiment, navigate to the **log8415-tp2/hadoop_vs_spark/** directory and ensure executables permissions by running **sudo chmod +x configuration.sh** and **sudo chmod +x hadoop_vs_spark.sh**.
2. After, run the **hadoop_vs_spark.sh script** and follow the prompts displayed on the screen, checking all options and pressing 'enter'.
3. The previous commands should output a file named **hadoop_vs_spark_comparison.txt** which contain the chrono time for each execution. Also, you can find in this same directory the output of the two algorithms which are the lists of words with their number of occurrences.

Friend suggestions

1. To execute the Friend suggestions experiment, navigate to the **log8415-tp2/friend_suggestions/** directory and ensure executables permissions by running **sudo chmod +x configuration.sh** and **sudo chmod +x friend_suggestions.sh**.
2. After, run the **friend_suggetions.sh** script.
3. The previous commands should output a directory called ouput which contain the results of the algorithm which are the friend suggestions.

5. Conclusion

The study presented in this report provides a comprehensive evaluation of word count operations using Hadoop and Linux command line tools on an AWS EC2 instance. The results clearly demonstrate the efficiency of native Linux commands for processing small datasets, outperforming the Hadoop implementation in terms of speed due to reduced overhead. Hadoop's strength, however, lies in its ability to handle larger datasets where its distributed computing capabilities can be fully leveraged.

Further, the comparison between Hadoop and Spark for word count tasks revealed Spark's proficiency in resource management, although it incurred a longer processing time. Hadoop, conversely, achieved faster processing times at the expense of greater resource usage. This highlights the importance of choosing the

right framework based on specific project needs, particularly when balancing processing speed against resource consumption.

The exploration of the Social Network problem via a MapReduce algorithm showcased Hadoop's robustness in distributed processing. The successful implementation of this algorithm underscores Hadoop's potential in analyzing extensive datasets to yield meaningful social connection recommendations.

Overall, the experiments conducted provide valuable insights into the selection of appropriate technologies for different scales of data processing. They emphasize the importance of aligning the choice of technology with the dataset size and the specific requirements of the computational task at hand. For small data, the simplicity and speed of Linux commands are unparalleled, while for large-scale data processing, the distributed capabilities of Hadoop and Spark become indispensable. This study serves as a guide for making informed decisions in the deployment of data processing technologies in cloud-based environments.

References

1. https://stackoverflow.com/questions/15035778/hadoop-m-r-to-implement-people-you-might-know-friendship-recommendation
2. https://dzone.com/articles/getting-hadoop-and-runninghttps://dzone.com/articles/getting-hadoop-and-running
3. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
4. https://www.digitalocean.com/community/tutorials/how-to-install-java-with-apt-on-ubuntu-22-04
5. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html