LOG8415
Advanced Concepts of Cloud Computing
TP2

Thomas Caron - 1944066
Kevin Lam - 2024921
Ghazi Ben Achour - 1926009
Slimane Boussafeur - 2017001

Our repo can be found at: https://github.com/tomtom103/LOG8415E/tree/main/TP2

1 Experiments with WordCount program

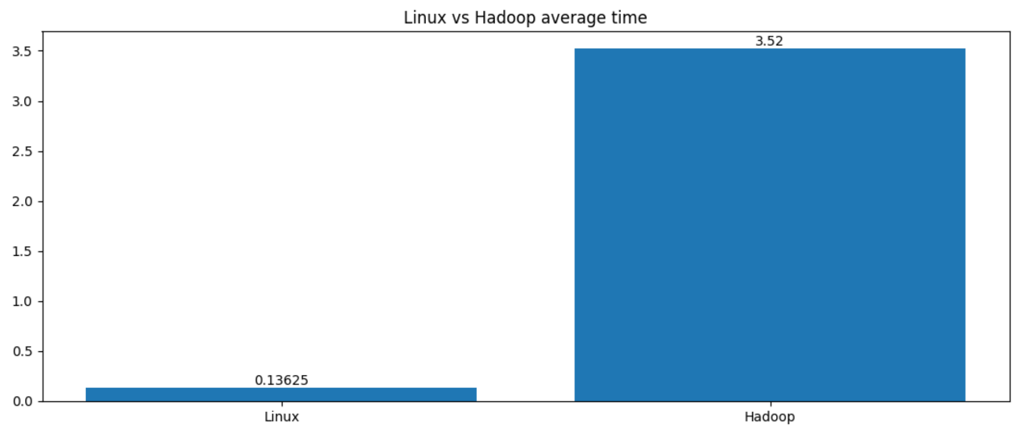
When we first got started, we tried installing Hadoop on our personal computers using the link that was provided in the assignment instructions. We soon found out however that getting Hadoop running on all of our computers was a tedious task and did not always work as we were hoping it would. As a first step, we decided to build our own Docker image containing Hadoop and PySpark (which can be found in the repository). This allowed us to run the WordCount program directly inside the Docker image, which works regardless on the computer it runs on.

We then decided to create the WordCount program for Linux using bash which lead us to use the AWK programming language instead of the provided command since we found it was easier to insert in multiple scripts.

Creating the wordcount program using PySpark was just as straight-forward, since we've had to use the python library in other courses before.

2 Performance comparison of Hadoop vs. Linux

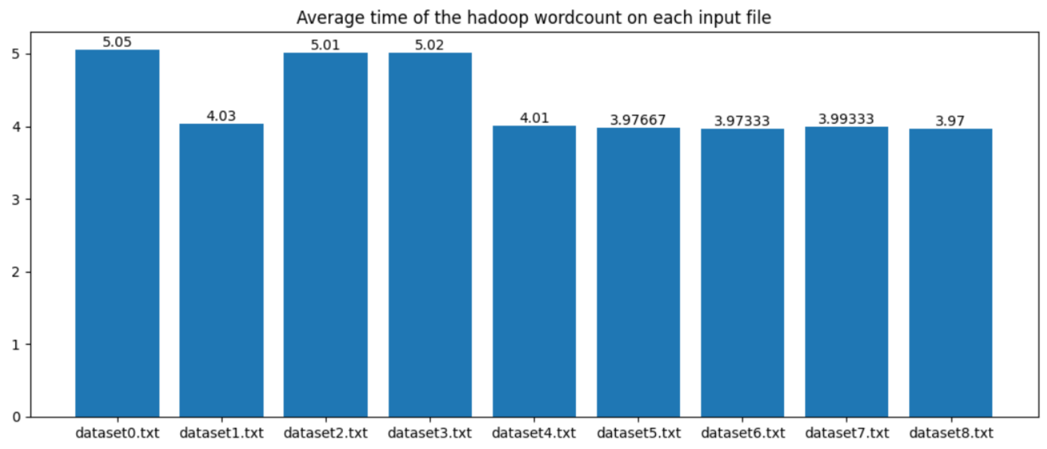
We compared the performance of running the wordcount natively with linux (using only available commands on bash), versus Hadoop. We ran both algorithms 8 times to get an approximate average run-time for each. These metrics were ran on our personal computers, results might differ from one computer to another. Here are the results we found:

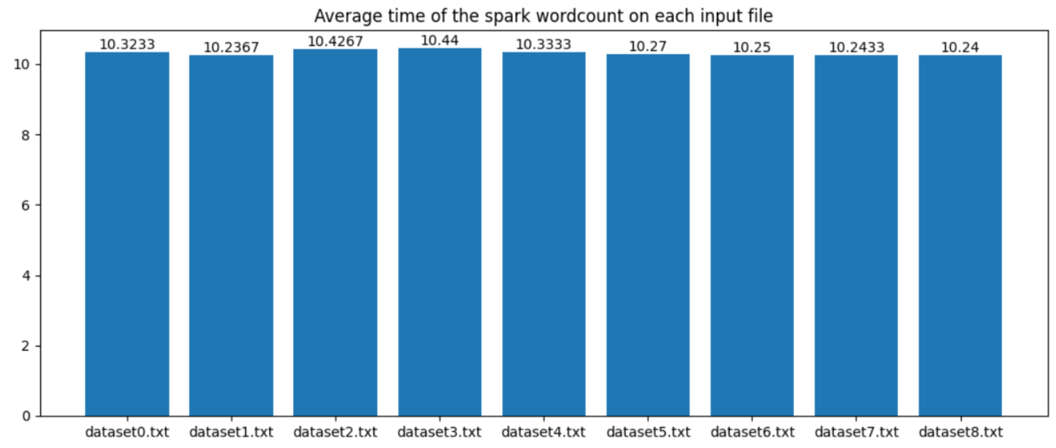


We can see that Linux is the clear winner here, running on average 26 times faster than Hadoop.

3 Performance comparison of Hadoop vs. Spark on AWS

For this part of the assignment, we were task of comparing the performances of Hadoop and Spark using a simple map reduce word count algorithm on multiple data sets (9 in total). We ran each algorithm 27 times (3 times on each data sets) on an EC2 M4.large virtual machine. We collected the real time using the **time** Linux command. Here are the results of this experiment :





Surprisingly, Hadoop is twice as fast on this occasion even though Spark is supposed to be the quickest. This can be explained by the fact that for each time the algorithm is ran, we have to do the set up for Spark. Each time we run the word count, a Spark Session needs be created and the data needs to be loaded into Spark which is the most time consuming operation of the whole algorithm because of how Spark structures the data internally. On the other hand, Hadoop loads the file and reads it line by line, which is much faster. If we only timed the map and reduce operation, we would see that Spark if far ahead in termes of performance.

4 How we used MapReduce jobs to solve the social network problem

We used MapReduce jobs to solve the social network problem because MapReduce is used to accelerate a computation by dividing it up into smaller problems which are then solved in parallel using multiple machines. In our case, we have a large number of users, so we divide them into smaller groups and solve the "People you might know" problem, then we sum the results of those small groups to obtain the solution for the entire data.

5 How our algorithm tackles the social network problem

Our algorithm divides the social network problem into two parts: the map and reduce functions.

We divide the text file line into two parts in the map function: the user id for whom we want to make friend recommendations and the list of his current friends. We send the user's id, along with his friend list and the indicator "/DEG1." We also verify the scenario when the user has no friends. Following that, we emit all of the potential mutual friends from the list, including the indicator "/DEG2" since they share a friend in common. The map function essentially maps the user relationships and adds an indicator to help the reduce function.

We group all of the recommended friends by users in the reduce function. We take the value from the key-value emitted by the map function and split it in two, the values before the "/" and the indicator after the "/." If we see the indicator "DEG1," we add them to the list of current friends so we can remove them from the list of recommendations later, and if we see the indicator "/DEG2," we add them to the list of recommendations. Finally, we emit a key-value pair in which the value is from a list of recommendations sorted in decreasing order in order to obtain the top ten recommendations.

6 Recommendations of connections of users

Our recommendations of connection for the users with the IDs: 924, 8941, 8942, 9019, 9020, 9021, 9022, 9990, 9992, 9993 are as follow:

- 924: 11860, 15416, 2409, 43748, 439, 45881, 6995
- 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

The list of the filtered IDs and the list of all IDs can be found in the *out/* directory

7 Summary of results and how to run our code

In the end, we found out that for a simple problem such as word counting, Spark and Hadoop seem to be vastly overkill compared to the Linux solution which ran much faster. However, we can also see that for the social network problem, these technologies are more appropriate as it would be much harder to write a solution to that problem with a shell script.

Everything needed to run our code, as well as explanations can be found in the multiple README.md files located in our github repository linked above.