TP2 - LOG8415E: MapReduce sur AWS

Authors : Stéphane Michaud 1904016 - Stephen Cohen 2412336 - Zabiullah Shair Zaie 2087651 - Asma Boukhdhir 2412257

WordCount Program Experiments

Setup

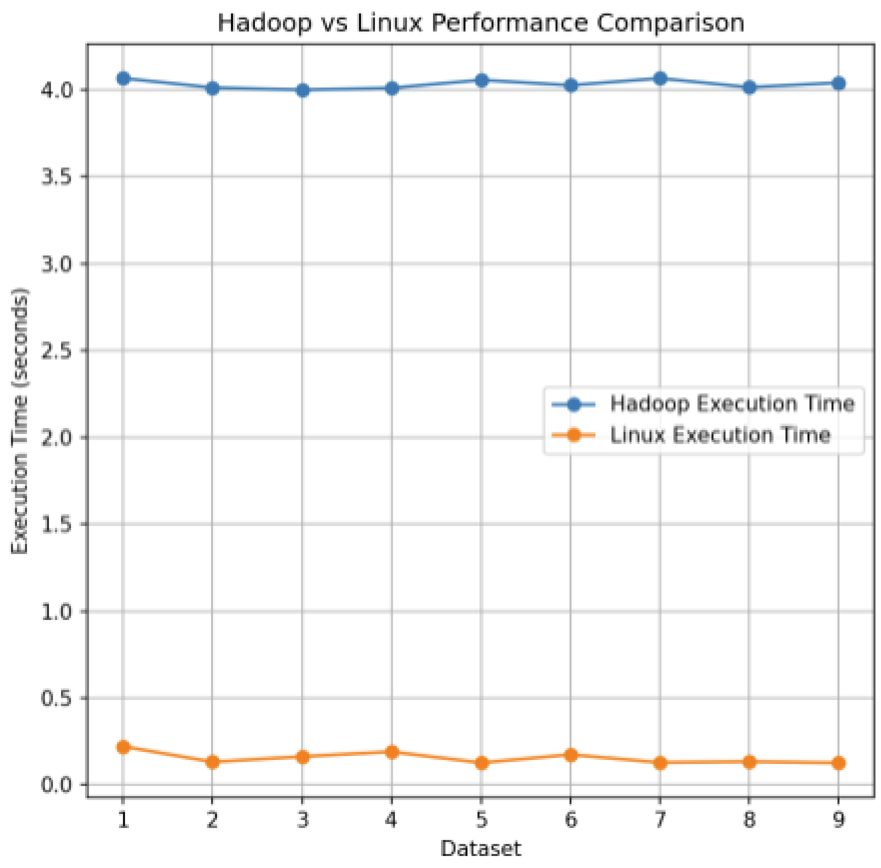
We used the classic WordCount program to count occurrences of words in large text files. The input datasets were loaded from various sources, and we ran the WordCount algorithm using:

- Hadoop (with HDFS)
- Spark
- A simple Linux command pipeline (`cat`, `tr`, `sort`, `uniq`).

Each method was executed on different datasets to measure their performance on tasks of increasing size.

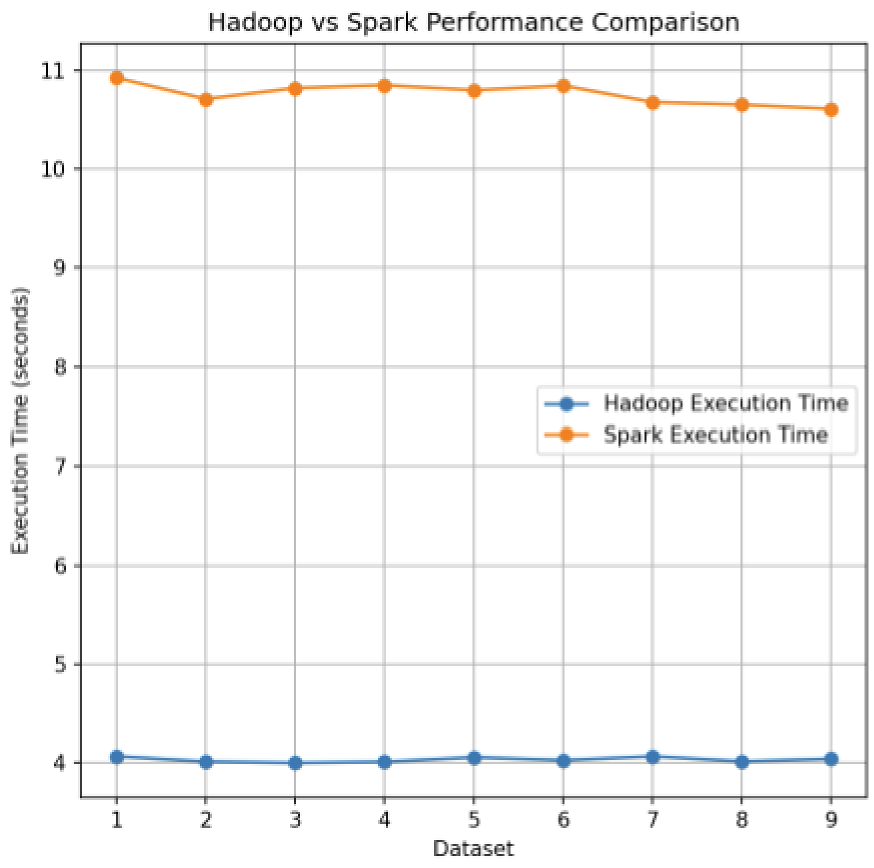
Results

Hadoop VS Linux



As we can observe, Linux is more efficient and `13 times` faster than Hadoop to realize those tasks.

Hadoop VS Spark



Here, Hadoop is `3 times` faster to complete this task than Spark on every dataset.

Remark :

- We expected Spark to be faster than Hadoop however the results show the opposite. It is perhaps due to the fact that memory is not requested and Spark is better for completing a lot of small tasks like in machine Learning. Moreover, it is possible that if the tasks were longer, Spark would be faster to complete.

- On the other hand, we observe that there is an huge variance when we run the wordcount_spark_profile.py This could explain why Spark can be disadvantaged.

Social Network Problem Using MapReduce

The social network problem involves recommending friends to users based on mutual connections. We used MapReduce to solve this problem by calculating second-degree connections and ranking users by the number of mutual friends.

Approach

1. **Map Phase:** For each user, create a list of all second-degree connections (friends of friends).
2. **Reduce Phase:** For each pair of users, count the number of mutual friends and sort them by this count to generate friend recommendations.

Algorithm

Our algorithm takes the following approach:

- **Input:** A tab-separated file where each line lists a user and their direct friends.

- **Map Phase:** Emit pairs of users who share common friends.

- **Reduce Phase:** Count how many mutual friends exist between each pair and generate the top 10 friend recommendations.

The code is available here : friend_recommendation.py

Challenges

One challenge was ensuring that no direct friends were recommended. We handled this by filtering out direct connections in the Reduce phase.

```
friends_filtered = friends_connection.filter(lambda line: -1 not in line[1])
```

Results

Below are the friend recommendations for the specified users based on our MapReduce algorithm:

User ID	Recommendations

User ID	Recommendations
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

Appendix

- To run code, you have to :
    1. Have you AWS credentials in your environnement thanks to `aws configure` or by copying it in `~/.aws/credentials`
    2. Run `automation.py`

The full code is available here