COEN 285 Big Data

Programming Assignment 2

Project Report on

Top K Most Popular Words using Map Reduce

Project Group:
1. Sanskriti Patole
2. Vaibhav Sachdeva

Introduction

In recent years, the volume of data generated by individuals and organizations has grown exponentially, making it increasingly challenging to extract valuable insights from these vast datasets. Apache Hadoop is an open-source tool that offers an efficient and scalable solution for processing Big Data. In this project, we have utilized Hadoop MapReduce to analyze large sets of data and calculate the top frequency words present in them. The goal of this project was to demonstrate the capabilities of Hadoop MapReduce for analyzing unstructured data and to showcase its ability to process and aggregate massive amounts of data in parallel across multiple nodes. By distributing the data across multiple nodes and processing it in parallel, Hadoop significantly improves efficiency. This report provides an overview of the project, outlines the dataset used, the tools employed to achieve the objectives of the given problem statement, and the algorithms implemented to calculate the frequency of each word and identify the top 100 most frequent. Furthermore, the report also provides graphical results and analysis, followed by a conclusion. Through this project, we aim to demonstrate the effectiveness of Hadoop MapReduce in processing large datasets and extracting valuable insights from unstructured data.

Experimental Setup

In this project, three input datasets of size 300 MB, 2.5GB, and 16GB containing English text with stopwords(that needed to be excluded) were utilized. The project involved conducting two experiments. The first experiment aimed to identify the top 100 most frequent words in the input file, while the second experiment focused on identifying the top 100 most frequent words of length greater than 6 characters. The experiments were carried out using Hadoop MapReduce, with various tuning parameters tested to optimize performance. All the experiments were run on a Hadoop cluster with 23 nodes. The programming language used to write the mapper and reducer was Python.

Techniques Used

The following techniques were used for all three datasets and for both the experiments (top 100 words and top 100 words having length greater than 6) -

1. Mapper + Reducer
2. Mapper + Reducer + Combiner
3. Mapper + Reducer + Combiner with Partitioning
4. Mapper + Reducer + Combiner with Compression

Technique 1 (Mapper + Reducer)

A basic MapReduce program comprises two phases - mapper and reducer.

Mapper - The Mapper component in MapReduce is in charge of converting input data into key-value pairs through a user-defined function. It operates on multiple parts of the input data simultaneously to process it in parallel and generates intermediate key-value pairs.

Reducer - The Reducer component in MapReduce is responsible for grouping intermediate key-value pairs created by the Mapper based on their keys. It then applies a user-defined function to each group to merge them and produce the final output. The reducer is responsible for outputting the top 100 most frequent words.

In this technique we pass all three input datasets into the mapper first. Then, the output of the mapper is fed to the reducer, which then outputs the top 100 most frequent words.

Technique 2 (Mapper + Reducer + Combiner)

In this technique we use MapReduce with a Combiner. A Combiner in MapReduce is a function that aggregates intermediate key-value pairs produced by the Mapper. It is executed after the Mapper and before the Reducer, and its main objective is to reduce the volume of data transferred between the two. By performing local aggregation at the map task nodes, the Combiner can reduce network bandwidth usage and improve the performance of the MapReduce job.

In our case we have used the Reducer as a Combiner. The input data is passed to the mapper, its output is passed to the combiner (which is the same as the reducer) and then the output of the combiner is passed to the reducer. The reason behind using the reducer as the combiner is that it reduces code complexity, simplifies debugging, improves performance, and ease of maintenance.

Technique 3 (Mapper + Reducer + Combiner with Partitioning)

A Partitioner is a component of the MapReduce process that assigns each key-value pair produced by the Mapper to a specific Reducer for processing. The goal of the Partitioner is to distribute the data uniformly across the Reducers and ensure that all pairs with the same key are sent to the same Reducer. This component is executed between the Mapper and Reducer stages in the MapReduce workflow, and it determines the Reducer task to which each pair will

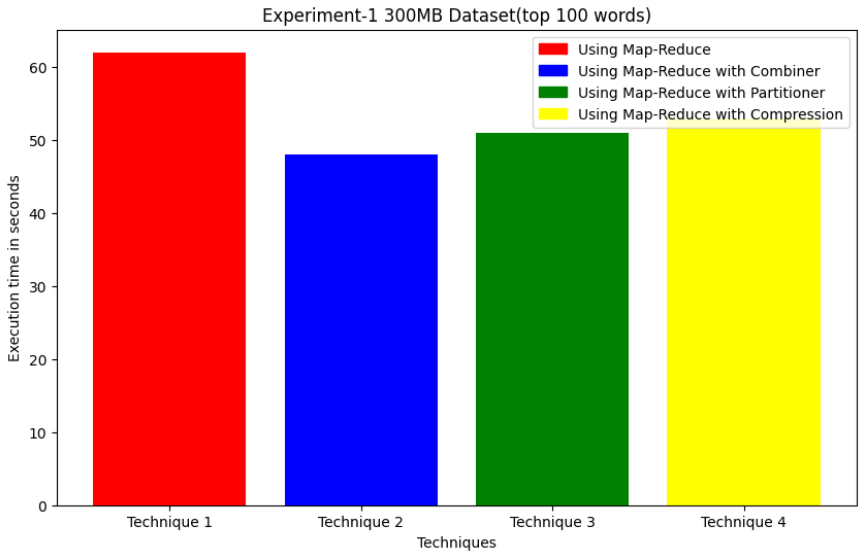
be assigned based on its key. Using a Partitioner in MapReduce is highly advantageous as it helps distribute data uniformly across the Reducer tasks, preventing a situation where some tasks are overloaded with data while others are idle. This equal distribution of data among the Reducers avoids longer processing times and inefficient utilization of resources, thereby improving the overall performance of the MapReduce job. In our case we used KeyFieldBasedPartitioner, which is specified using the -partitioner option. The -D mapreduce.partition.keypartitioner.options=-k1,2 option was also used which specifies the partitioning key to be the first two fields (-k1,2) of the input data.

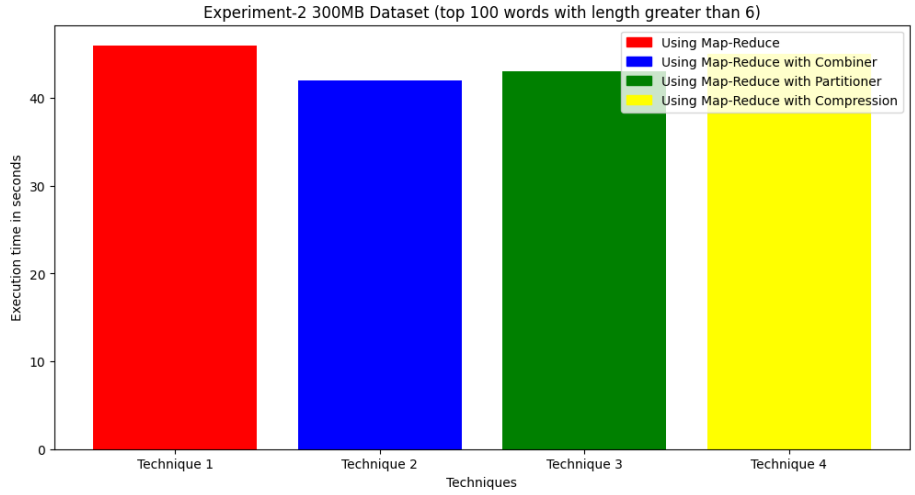
Technique 4 (Mapper + Reducer + Combiner with Compression)

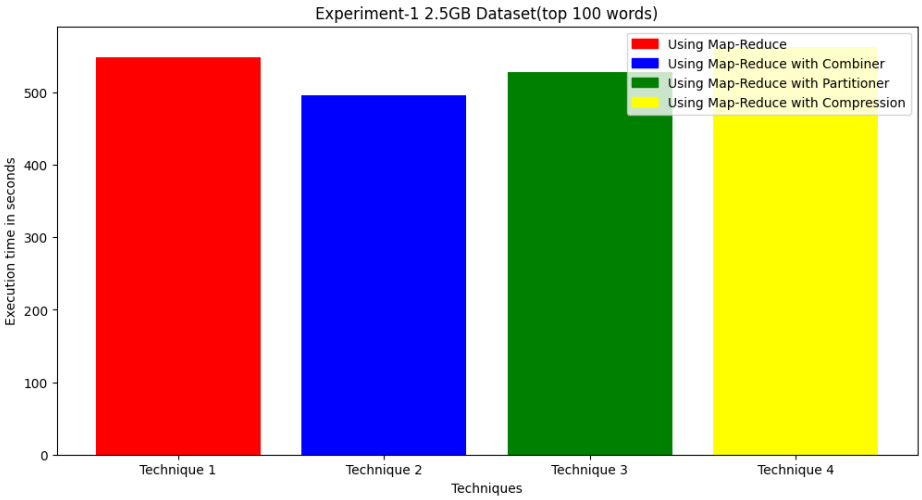
In MapReduce, compression is utilized to lessen the data that has to be moved between the Mapper and the Reducer. This technique compresses the intermediate data, which reduces the amount of data that must be written to disk and transmitted over the network. As a result, MapReduce jobs can run faster. There are multiple compression algorithms available for use in MapReduce, such as Gzip, Snappy, and LZO. In our case we have chosen LZO as the compression algorithm.

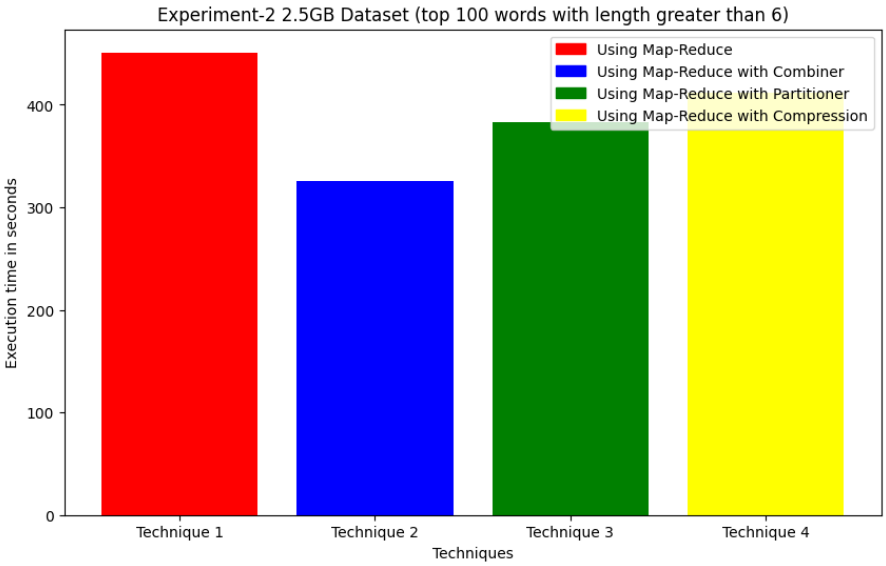
We have used LZO as a compression algorithm due to its popularity and several advantages over other compression algorithms. It is fast and efficient, making it suitable for distributed computing environments like MapReduce. Additionally, LZO is highly optimized for use with Hadoop and MapReduce, enabling it to exploit the unique features of these frameworks and deliver better performance.

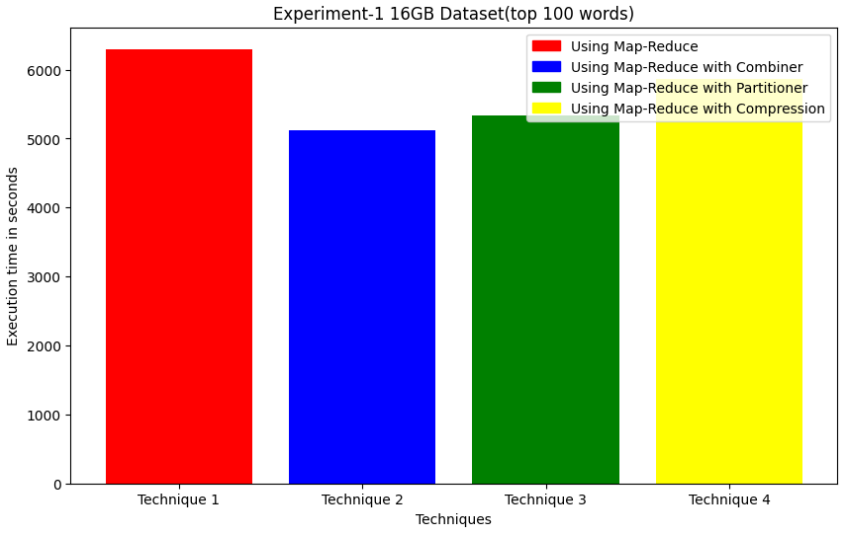
Results and Analysis

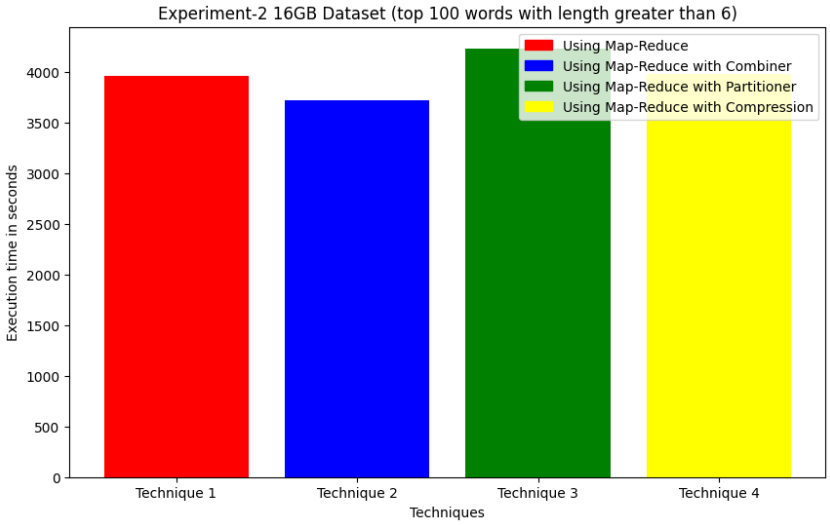



Experiment-2 300MB Dataset (top 100 words with length greater than 6)
Using Map-Reduce
Using Map-Reduce with Combiner
Using Map-Reduce with Partitioner
Using Map-Reduce with Compression
Execution time in seconds
0
10
20
30
40
Technique 1
Technique 2
Technique 3
Technique 4
Techniques


Experiment-1 2.5GB Dataset(top 100 words)
Using Map-Reduce
Using Map-Reduce with Combiner
Using Map-Reduce with Partitioner
Using Map-Reduce with Compression
Execution time in seconds
0
100
200
300
400
500
Technique 1
Technique 2
Technique 3
Technique 4
Techniques


Experiment-2 2.5GB Dataset (top 100 words with length greater than 6)
Using Map-Reduce
Using Map-Reduce with Combiner
Using Map-Reduce with Partitioner
Using Map-Reduce with Compression
Execution time in seconds
0
100
200
300
400
Technique 1
Technique 2
Technique 3
Technique 4
Techniques





Conclusion

From the above results we can conclude that -

In Experiment 1 (top 100 most frequent words) -

1. For the 300 MB dataset, Mapper + Reducer + Combiner takes the least execution time of 48s followed by Mapper + Reducer + Combiner with Partitioning taking 51s. Mapper + Reducer and Mapper + Reducer + Combiner with Compression takes 62s and 53s respectively.
2. For the 2.5GB dataset, Mapper + Reducer + Combiner takes the least execution time of 496s followed by Mapper + Reducer + Combiner with

Partitioning taking 528s. Mapper + Reducer and Mapper + Reducer + Combiner with Compression takes 594s and 563s respectively.

3. For the 16 GB dataset, Mapper + Reducer + Combiner takes the least execution time of 5124s followed by Mapper + Reducer + Combiner with Partitioning taking 5336s. Mapper + Reducer and Mapper + Reducer + Combiner with Compression takes 6289s and 5861s respectively.

In Experiment 2 (top 100 most frequent words with length greater than 6) -

4. For the 300 MB dataset, Mapper + Reducer + Combiner takes the least execution time of 42s followed by Mapper + Reducer + Combiner with Partitioning taking 43s. Mapper + Reducer and Mapper + Reducer + Combiner with Compression takes 46s and 45s respectively.
5. For the 2.5GB dataset, Mapper + Reducer + Combiner takes the least execution time of 326s followed by Mapper + Reducer + Combiner with Partitioning taking 383s. Mapper + Reducer and Mapper + Reducer + Combiner with Compression takes 451s and 412s respectively.
6. For the 16 GB dataset, Mapper + Reducer + Combiner takes the least execution time of 3725s followed by Mapper + Reducer + Combiner with Partitioning taking 4237s. Mapper + Reducer and Mapper + Reducer + Combiner with Compression takes 3964s and 3982s respectively.