Università di Pisa

Artificial Intelligence and Data Engineering

Cloud Computing

A.A. 2022/23

Word Co-Occurrence in Hadoop

Author: Giulio Bello

Chapter 1: Introduction

The volume of data being generated and processed in various domains has increased significantly in recent years. Extracting meaningful insights and patterns from this vast amount of data is a challenging task. In this context, Hadoop MapReduce has emerged as a powerful framework for distributed processing of large-scale datasets.

1.1 Project Overview

This project focuses on utilizing Hadoop MapReduce to perform word co-occurrence analysis. Word co-occurrence analysis involves identifying the frequency and patterns of word pairs occurring together within a given text corpus. By analyzing word co-occurrences, we can gain insights into the relationships and associations between different words, which can be valuable in various applications such as natural language processing, text mining, and information retrieval.

1.2 Objectives

The primary objective of this project is to develop a MapReduce-based solution for word co-occurrence analysis. The project aims to implement efficient algorithms and strategies to identify co-occurring word pairs and generate meaningful output based on the input text data. Additionally, the project aims to evaluate the performance and effectiveness of different implementation choices and optimization techniques.

1.3 Hadoop MapReduce

Hadoop MapReduce is a distributed data processing framework designed to handle large-scale datasets across a cluster of computers. It provides a scalable and fault-tolerant infrastructure for processing and analyzing data in parallel. The core idea behind MapReduce is to divide a computation task into smaller sub-tasks that can be executed independently on different machines in the cluster. The results from these sub-tasks are then combined to produce the final output.

1.4 Problem Statement

The specific problem addressed in this project is the analysis of word co-occurrences within a text corpus using Hadoop MapReduce. Given a collection of text documents as input, the objective is to identify word pairs that occur together within a specified window of words. The project aims to count the occurrences of these word pairs and generate a meaningful output that can provide insights into the relationships between words in the corpus.

Chapter 2: Project Overview

2.1 Project Description

The project aims to develop a word co-occurrence analysis solution using Hadoop MapReduce. Given a large text corpus as input, the system will process the data in a distributed manner and generate co-occurrence information for word pairs. The output will provide insights into the relationships between words and their frequency of co-occurrence within a specified window.

2.2 Technology Stack

The project leverages several technologies and frameworks to implement the word co-occurrence analysis solution. The key components of the technology stack include:

- Hadoop: An open-source distributed computing framework that provides a scalable and fault-tolerant platform for processing large-scale datasets.

- MapReduce: A programming model and computational framework for distributed data processing in Hadoop, allowing parallel execution of tasks across a cluster.

- Java: The programming language used to develop the MapReduce application, as it is the primary language for Hadoop development.

- Hadoop File System (HDFS): The distributed file system used by Hadoop for storing and retrieving data across multiple machines.

2.3 Project Workflow

The project follows a specific workflow to perform word co-occurrence analysis using Hadoop MapReduce. The workflow consists of the following steps:

1. Input Data: The project takes a collection of text documents as input. These documents could be stored in a directory or multiple directories accessible by the Hadoop cluster.

2. Map Phase: The mapper class is responsible for processing each input document. It tokenizes the text, extracts individual words, and creates co-occurrence "stripes" for each word. These stripes contain information about the co-occurring words and their counts within a specified window.

3. Reduce Phase: The reducer class receives the output from the mappers and combines the co-occurrence stripes for each word. It aggregates the counts of co-occurring word pairs, producing the final output for each word.

4. Output Generation: The output consists of the co-occurrence information, typically represented as word pairs and their respective counts. Optionally, the output can be filtered or formatted based on specific requirements.

5. Execution: The MapReduce job is submitted to the Hadoop cluster, which assigns tasks to the available nodes for parallel processing. The results are then collected and combined to generate the final output.

The project workflow demonstrates the distributed nature of Hadoop MapReduce, where data is divided into chunks and processed concurrently by multiple mappers and reducers, providing scalability and efficiency in large-scale data analysis.

Chapter 3: Implementation Details

3.1 Mapper Implementation

The mapper plays a crucial role in the word co-occurrence analysis project. The WordCoOccurrenceMapper class is responsible for processing each input document and generating co-occurrence "stripes" for each word. Here's an overview of the mapper implementation:

- Initialization: The mapper initializes a data structure, such as a HashMap, to store the co-occurrence stripes for each word. It also retrieves the window size parameter from the job configuration.

- Mapping: For each input document, the mapper tokenizes the text by splitting it into individual words. It then iterates through the words, creating or updating the corresponding co-occurrence stripe for each word.

- Co-occurrence Counting: Within a specified window around each word, the mapper identifies the neighboring words. It increments the count of co-occurring words in the stripe, excluding the word itself.

- Output: After processing all the words in a document, the mapper emits key-value pairs, where the key is the word, and the value is the corresponding co-occurrence stripe.

```
Mapper:
    Initialize a HashMap for storing the co-occurrence stripes
    Retrieve the window size parameter from the configuration

    for each input document:
        tokenize the text into individual words

        for each word in the document:
            create or update the co-occurrence stripe for the word

            for each neighboring word within the window:
                increment the count of the neighboring word in the stripe

        emit the word as the key and the co-occurrence stripe as the value
```

3.2 Reducer Implementation

The reducer class, WordCoOccurrenceReducer, receives the output from the mappers and combines the co-occurrence stripes for each word. The implementation of the reducer is as follows:

- Input: The reducer receives a key-value pair, where the key is a word, and the values are the co-occurrence stripes associated with that word.

- Combining Stripes: The reducer iterates through the co-occurrence stripes and combines them for each word. It aggregates the counts of co-occurring word pairs, creating a consolidated stripe for each word.

- Output: For each consolidated stripe, the reducer emits key-value pairs where the key is a word pair (e.g., "word1 - word2"), and the value is the count of their co-occurrence.

```
Reducer:
    for each word, stripes in the input:
        create a consolidated stripe

        for each stripe in stripes:
            combine the stripe with the consolidated stripe

        for each word pair, count in the consolidated stripe:
            emit the word pair as the key and the count as the value
```

3.3 Configuration Parameters

The project utilizes several configuration parameters to control the behavior of the MapReduce job. These parameters can be set through command-line arguments or programmatically. Here are the main configuration parameters used:

- Window Size: The size of the window around each word within which co-occurrences are considered. It determines the range of neighboring words to analyze for each word.

- Filter Combinations: A boolean parameter indicating whether the output should be filtered to include only unique word pairs. If set to true, the reducer filters out duplicate word pairs.

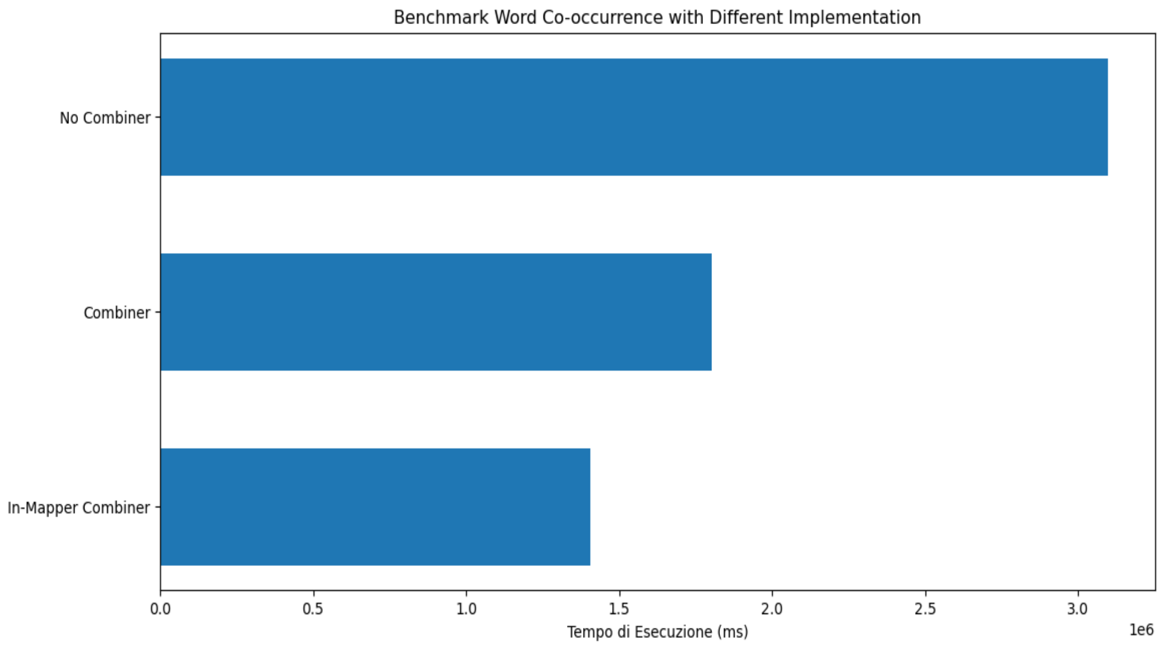
- Number of Reducers: The number of reducer tasks to be executed in parallel. It affects the level of parallelism and resource utilization in the MapReduce job.

3.4 In-Mapper Combiner: Stateful Approach

In the word co-occurrence analysis project, we have chosen to employ a stateful in-mapper combiner approach instead of a stateless in-mapper combiner, a regular combiner, or no combiner at all. This choice is motivated by the characteristics of the co-occurrence analysis process and the benefits offered by a stateful in-mapper combiner.

In a stateful in-mapper combiner, each mapper maintains a local data structure, such as a HashMap, to store partial aggregation results before emitting them. This allows the mapper to aggregate co-occurrence counts locally, reducing the amount of data that needs to be transferred to the reducers. By keeping track of co-occurrence counts within the mapper, we avoid unnecessary network traffic and improve the overall efficiency of the MapReduce job.

The advantage of using a stateful in-mapper combiner becomes evident in word co-occurrence analysis, where a large number of co-occurring word pairs can be encountered. Without any combiner, the mapper would emit a considerable volume of intermediate data, which could lead to performance bottlenecks during data transfer and processing in the reducers. By using a stateful in-mapper combiner, we effectively reduce the volume of data flowing through the MapReduce pipeline, leading to faster execution times and reduced resource consumption.



Chapter 4: Experimental Evaluation

4.1 Dataset Description

The word co-occurrence analysis project utilizes a dataset consisting of the first part of the Bible, which is divided into 781 individual text files. The dataset represents a substantial collection of textual data, providing a suitable basis for evaluating the performance and efficiency of the word co-occurrence analysis solution.

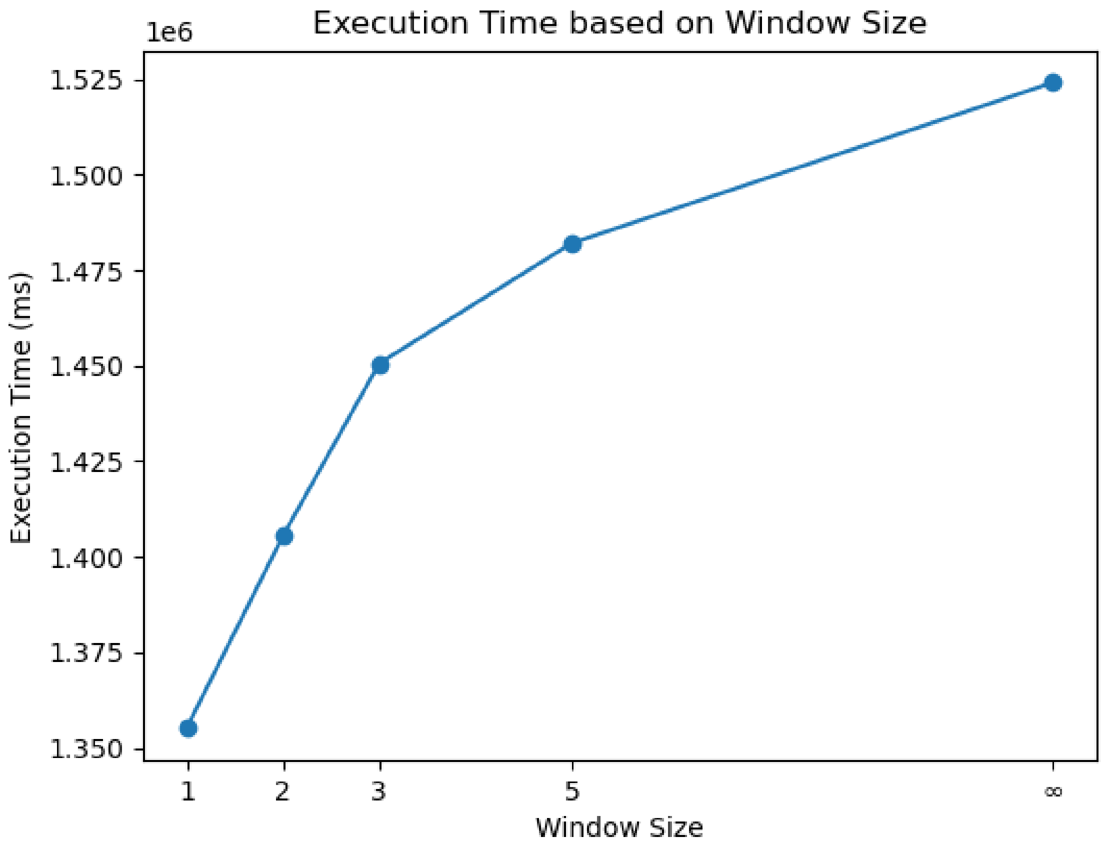
4.2 Experiment Setup

To evaluate the performance of the MapReduce-based word co-occurrence analysis, we conducted several experiments with varying configurations. The key aspects we focused on were the window size and the number of reducers. We also compared the results with a Python-based implementation to observe the impact of dataset size on performance.

4.3 Impact of Window Size

One of the critical parameters in the word co-occurrence analysis is the window size, which determines the range of neighboring words considered for co-occurrence counting. We ran the MapReduce job with different window sizes to observe its impact on execution time.

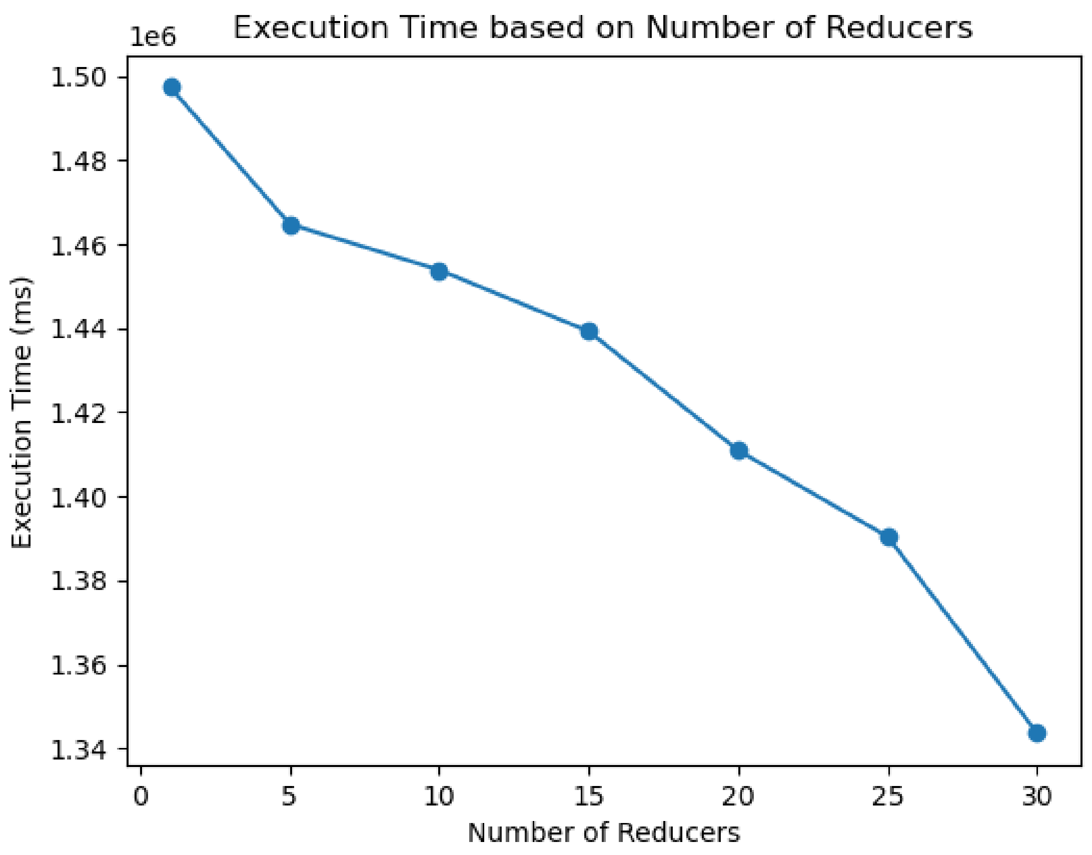
Results: As expected, a smaller window size led to faster execution times. With a smaller window, each mapper processed fewer words, resulting in reduced computation and data transfer overhead. However, a smaller window might miss some long-range co-occurrences, which could affect the analysis accuracy.



4.4 Impact of Number of Reducers

The number of reducers is another crucial parameter influencing the performance of the MapReduce job. We conducted experiments with varying numbers of reducers to evaluate its impact on execution time.

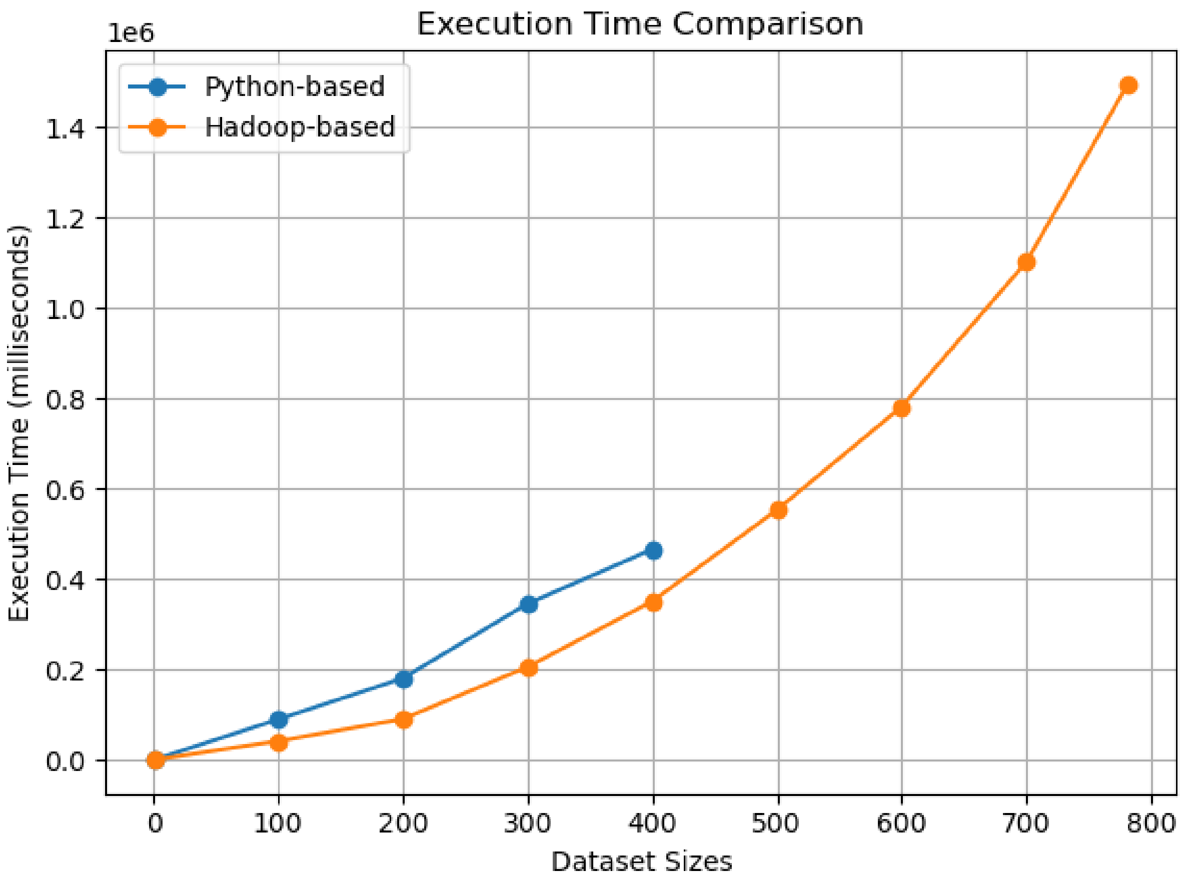
Results: Increasing the number of reducers improved the parallel processing capability, leading to faster execution times. However, setting an excessively high number of reducers could lead to excessive overhead, as the amount of data exchanged between reducers increases.



4.5 Comparison with Python Implementation

To assess the scalability of the MapReduce solution, we compared its performance with a Python-based word co-occurrence analysis implementation. For small datasets, Python was able to execute the analysis relatively quickly due to its simplicity and low overhead.

Results: As the dataset size grew, Python's execution time increased significantly due to its inability to handle large-scale distributed processing. Python's memory limitations became evident, and the execution time skyrocketed compared to the MapReduce solution. The Hadoop MapReduce framework demonstrated its superiority in handling large-scale datasets efficiently and distributing the processing load across the cluster of machines.



4.6 Memory Efficiency of MapReduce

One of the significant advantages of the MapReduce approach is its memory efficiency in handling large datasets. While Python struggled with memory limitations for large-scale datasets, Hadoop MapReduce demonstrated robust memory management and fault tolerance.

Results: The MapReduce-based word co-occurrence analysis solution efficiently distributed the data across the cluster, enabling parallel processing with minimal memory consumption on individual nodes. This efficient memory management allowed the MapReduce job to handle large datasets without exhausting system resources.

4.7 Conclusion

The experimental evaluation of the word co-occurrence analysis project using Hadoop MapReduce demonstrated its effectiveness and efficiency in processing large-scale datasets. The use of a stateful in-mapper combiner contributed to reducing data transfer overhead and enhancing overall performance.

The experiments revealed the impact of window size and the number of reducers on execution time, providing insights into parameter tuning for optimal results. Moreover, the comparison with a Python-based implementation highlighted the limitations of non-distributed solutions for handling large datasets.

In conclusion, the Hadoop MapReduce-based word co-occurrence analysis solution proved to be a powerful and scalable approach for analyzing vast amounts of textual data efficiently. The framework's ability to distribute computation across a cluster and its memory efficiency make it an ideal choice for big data processing tasks. As the volume of data continues to grow, the scalability and performance of MapReduce will become increasingly valuable in various domains, including natural language processing and information retrieval.