COEN 285 Big Data

Programming Assignment 1

Project Report on

Top K Most Popular Words

Project Group:

1. Sanskriti Patole

2. Vaibhav Sachdeva

1.Introduction:

The purpose of this project is to evaluate how different factors, including data structures, algorithms, memory availability, and input size, affect program execution. To achieve this objective, the study measures the execution time of each algorithm for each input size using three datasets of varying sizes: 50MB, 300MB, 2.5GB, and 16GB. The goal of the project is to determine the top K most frequent words in a given input file by calculating the frequency of each word in the text file and then printing the top K most frequent ones. The report is structured into five sections: an introduction to the project (section 1), a description of the dataset and computer specifications (section 2), an explanation of the algorithms and data structures used (section 3), graphical results and analysis (section 4), and a conclusion for the study (section 5).

In summary, this study aims to investigate how different factors affect program execution and to identify the top K most frequent words in an input file. The study employs three text files of different sizes and presents its results in a structured manner through the various sections of the report. The report outlines the project's objective, methodology, and findings.

2.Experimental Setup:

There were four datasets provided of four different sizes: 50MB, 300MB, 2.5GB, 16GB text files. Each text file had English text along with some symbols. We were also provided with a few stopwords which were not to be considered in the set of popular words. In this programming assignment, we used my personal laptop, specifications of which are given below. This laptop was used for all experiments and the available memory remained the same throughout.

Windows 10, 16GB RAM, i7 core

3.Method

The method used for the project involved writing a Python program to find the top K words, where K is a user-specified value. The data structure used for the experiments was a dictionary, also known as an associative array. This data structure is an associative array that maps keys to values, which was useful for mapping words to their frequency. The dictionary was chosen for its fast lookup time and ease of use, making it ideal for handling large amounts of data. By implementing this method, the program was able to accurately and efficiently identify the top K most frequent words in a given text.

We used 3 different approaches to solve this assignment, all of which are explained in brief below:

3.1 Method 1: Read Entire File into RAM + COUNTER

The code reads in a large file into memory and initializes an empty dictionary to store the word frequencies. Then, it reads each line of the file, cleans the text by removing stopwords, and populates the dictionary with word frequencies. After the dictionary is populated, the Counter module is used to sort the words by frequency. Finally, the top K most frequent words are printed out. The code also times the execution of each step and prints out the duration of each step as well as the total execution time.

3.2 Method 2: Read line by line so only single line is stored in RAM + COUNTER

This code reads a text file, cleans the text, and counts the frequency of each word using a dictionary and a stopword list. It sorts the words by frequency using a Counter object and prints the top K words. The code measures the time it takes to read the file, count the words, and sort them. In summary, the code analyzes text data and provides insights into the most common words.

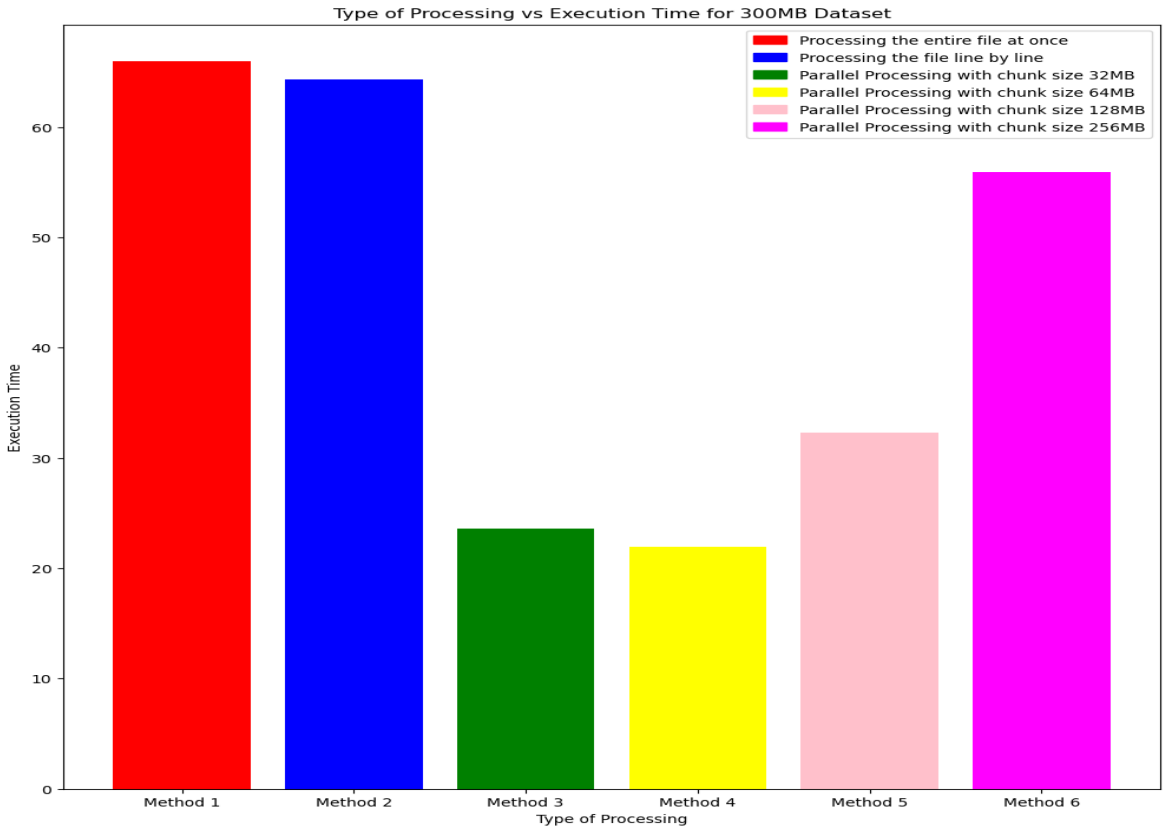
3.3 Method 3: Read file in chunks and process in parallel

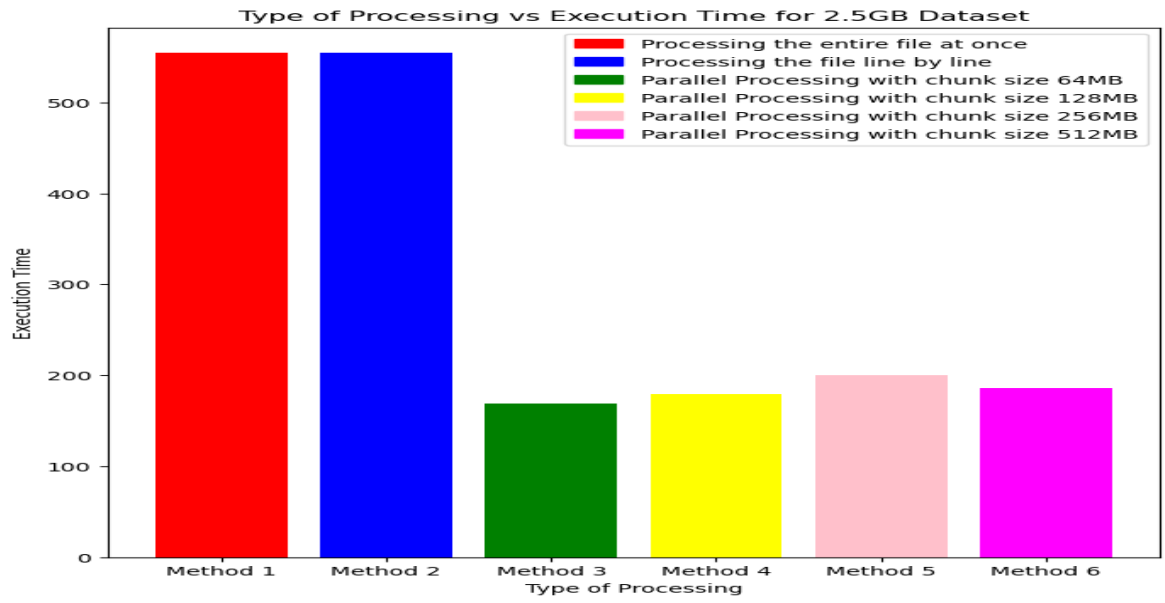
Although reading the text files line by line improved the performance slightly and reduced memory strain, it did not have a significant impact. To address this, the author employed a common technique used in Big Data where large files are broken into smaller chunks and processed in parallel. The author implemented this approach in their final experiment on three datasets. Each file was divided into equal-sized chunks(64MB, 128MB, 256MB, 512MB), and the chunks were processed in parallel using a dictionary and the Python Counter, as previously used. We selected the counter because it demonstrated faster performance than sorting the dictionary in their previous experiments. The resulting word frequencies were stored in a dictionary. This method significantly enhanced performance, with an average speedup of three times over previous cases.

4.Results and Analysis

In this section, We analyzed the performance of different cases based on their execution time in seconds, and the results are presented in Figures 1 and 2 for 300MB and 2.5GB sizes. The findings indicate that reading the entire file into memory, as done

in Method 1, is not efficient and results in the slowest execution times. Additionally, this case demonstrates the superiority of using Counter over sorting the dictionary in all three charts. On the other hand, Method 2, which reads the file line by line, showed improved execution times. Finally, Method 3, which reads the file in chunks and processes it in parallel, significantly reduced the execution time for all three files, indicating its effectiveness for large data files.





Method 1: The entire file is read into memory at once, and the words and their frequency are stored in a dictionary. The Python Counter function is used to sort the dictionary and count the top K words. The execution time for 300MB dataset is 65.99s, for 2.5GB dataset is 554.59s.
Method 2: Read the file line by line into memory, storing only one line at a time. The words and their frequency are stored in a dictionary and the Python Counter function is used to sort and count the top K words. The execution time for 300MB dataset is 64.34s, for 2.5GB dataset is 554.51s.
Method 3: Divide each file into smaller chunks(64MB), and each chunk is processed in parallel for improved efficiency. The execution time for 300MB dataset is 23.60s, for 2.5GB dataset is 168.52s.
Method 4: Divide each file into smaller chunks(128MB), and each chunk is processed in parallel for improved efficiency. The execution time for 300MB dataset is 21.98s, for 2.5GB dataset is 178.94s.
Method 3: Divide each file into smaller chunks(256MB), and each chunk is processed in parallel for improved efficiency. The execution time for 300MB dataset is 32.28s, for 2.5GB dataset is 200.19s.
Method 3: Divide each file into smaller chunks(512MB), and each chunk is processed in parallel for improved efficiency. The execution time for 300MB dataset is 55.95s, for 2.5GB dataset is 185.79s.

In Method 3, the file is partitioned into chunks of uniform size, although the size of each chunk can be modified. To identify the optimal chunk size, I conducted three tests using chunk sizes of 64MB, 128MB, 256MB, and 512MB. The outcome of these experiments depicts that the chunk size of 64MB resulted in the best performance for 2.5GB dataset, with the shortest execution time. And the chunk size of 128MB resulted in the best performance for 300MB dataset, with the shortest execution time

5.Conclusion

Big Data is centered around the ability to effectively process and read large datasets. The primary goal of this project was to analyze the impact of various factors on program efficiency. Specifically, the focus was on creating an efficient code to identify the top K most frequent words in a text file with minimal execution time. Three distinct scenarios were tested, and the algorithms used were evaluated based on execution time, with the results presented in a graphical format.

The presented figure displays the compiled outcomes for all cases and datasets. The findings indicate that loading complete files into memory, as in Method 1, is a slow and memory-intensive process. Method 2's performance improvement over Method 1 demonstrates the benefits of reading each line of the file, doesn't consume as much memory, yet it still doesn't offer a significant advantage over Method 1.

The most efficient performance was achieved by Method 3 for all file sizes, which involved dividing each file into smaller chunks and processing them in parallel. This supports the use of parallel processing and distributed storage in Big Data applications to improve performance. The aim of Big Data is to handle large and complex datasets that are challenging for traditional data processing methods and find ways to process them efficiently. This project's findings offer an example of how large files can be managed and processed efficiently.