UNIVERSITY OF PISA

Artificial Intelligence and Data Engineering

Cloud Computing

Hadoop Letter Frequency

Authors: Gemelli M. Namaki D. Nocella F.

Academic Year 2023/2024

Contents

Introduction

The objective of this project is to implement a data processing pipeline that can handle substantial data sets, ensuring efficient computation and meaningful data insights. By exploiting the MapReduce paradigm, data processing tasks is split into two main functions: the **Mapper** and the **Reducer**.
The Mapper function processes the input data, emitting **key-value pairs**, while the Reducer aggregates these pairs to produce the final output.

The project is developed using Hadoop framework, which provides an open-source implementation of the MapReduce paradigm. Hadoop allows for the distributed processing of large data sets across clusters of computers using simple programming models. The Hadoop ecosystem also includes other tools, such as HDFS (Hadoop Distributed File System) for distributed storage, and YARN (Yet Another Resource Negotiator) for cluster resource management.

Algorithm Design

The objective of this project is to analyze letter frequency in text documents utilizing Hadoop's MapReduce framework. Specifically, two distinct approaches are adopted to optimize the MapReduce task: the use of a Combiner and the implementation of an In-Mapper Combiner. These methods aim to enhance the efficiency of the MapReduce process by reducing the amount of data transferred between the Mapper and Reducer stages.

MapReduce with Combiner

The Combiner is a mini-reducer that processes the output of the Mapper tasks before passing it to the Reducer. By aggregating the intermediate data locally on the mapper nodes, the Combiner reduces the volume of data shuffled across the network, thus improving the performance of the MapReduce job. It performs its operations on the same node where the mapper is running.

Pseudocode

Algorithm 1 Letter Count with Combiner

Require: Txt file
Ensure: Total count of each letter in the input file

Mapper

```
1: procedure SETUP(context)
2:     normalize ← context.getConfiguration().get("normalize")
3:     one ← new LongWritable(1)
4:     letterCountKey ← new Text("total_letter_count")
5: end procedure

6: procedure MAP(Object key, Text value)
7:     line ← Normalize(value.toString(), normalize)      ▷ Remove accents and set lowercase
8:     for each character c in line do
9:         Emit(letterCountKey, one)
10:    end for
11: end procedure
```

Combiner & Reducer

```
12: procedure REDUCE(Text key, Iterable<LongWritable> values)
13:    sum ← 0
14:    for each LongWritable val in values do
15:        sum ← sum + val.get()
16:    end for
17:    Emit(key, new LongWritable(sum))
18: end procedure
```

Algorithm 2 Letter Frequency with Combiner

Require: Txt file, Total number of characters in the txt file
Ensure: Frequency of each letter in the input file

Mapper

```
1: procedure SETUP(context)
2:     normalize ← context.getConfiguration().get("normalize")
3:     one ← new LongWritable(1)
4: end procedure

5: procedure MAP(Object key, Text value)
6:     line ← Normalize(value.toString(), normalize)        ▷ Remove accents and set lowercase
7:     for each character c in line do
8:         Emit(new Text(c), one)
9:     end for
10: end procedure
```

Combiner

```
11: procedure REDUCE(Text key, Iterable<LongWritable> values)
12:     sum ← 0
13:     for each LongWritable val in values do
14:         sum ← sum + val.get()
15:     end for
16:     Emit(key, new LongWritable(sum))
17: end procedure
```

Reducer

```
18: procedure SETUP(context)
19:     letterCount ← context.getConfiguration().getLong("letterCountValue", 1)
20: end procedure

21: procedure REDUCE(Text key, Iterable<LongWritable> values)
22:     sum ← 0
23:     for each LongWritable val in values do
24:         sum ← sum + val.get()
25:     end for
26:     freq ← (double) sum / (double) letterCount
27:     Emit(key, new DoubleWritable(freq))
28: end procedure
```

MapReduce with In-Mapper Combiner

The In-Mapper Combiner aggregates the mapping and combining steps within the Mapper itself. This method involves accumulating the results in a data structure within the Mapper, which is then emitted at the end of the mapping phase. This approach minimizes the overhead of multiple data passes by efficiently combining intermediate results within the Mapper, reducing the need for external Combiner steps and further optimizing network usage and processing time.

Pseudocode

Algorithm 3 Letter Count with In-Mapper Combiner

Require: Txt file
Ensure: Total count of each letter in the input file

Mapper

procedure SETUP(Context context)
    normalize ← context.getConfiguration().get("normalize")
    map ← {}
    letterCountKey ← new Text("total_letter_count")
end procedure

procedure MAP(Object key, Text value)
    line ← Normalize(value.toString(), normalize)
    for each character c in line **do**
        map{$letterCountKey$} ← map{$letterCountKey$} + 1
    end for
end procedure

procedure CLEANUP(Context context)
    for each entry $< k, v >$ in map **do**
        Emit(k, v) ▷ Emint key and count map{key}
    end for
end procedure

Reducer

procedure REDUCE(Text key, Iterable<LongWritable> values)
    sum ← 0
    for each value in values **do**
        sum ← sum + value
    end for
    Emit(key, new LongWritable(sum))
end procedure

Algorithm 4 Letter Frequency with In-Mapper Combiner

Require: Txt file, Total number of characters in the txt file
Ensure: Frequency of each letter in the input file

Mapper

procedure SETUP(Context context)
    normalize ← context.getConfiguration().get("normalize")
    map ← {}
end procedure

procedure MAP(Object key, Text value)
    line ← Normalize(value.toString(), normalize)
    for each character c in line **do**
        map{c} ← map{c} + 1
    end for
end procedure

procedure CLEANUP(Context context)
    for each entry $< k, v >$ in *map* **do** ▷ Emit key and count map{key}
        Emit(k, v)
    end for
end procedure

Reducer

procedure SETUP(Context context)
    letterCount ← context.getConfiguration().getLong("letterCountValue", 1)
end procedure

procedure REDUCE(Text key, Iterable<LongWritable> values)
    sum ← 0
    for each value in values **do**
        sum ← sum + value
    end for
    freq ← (double) sum / (double) **letterCount**
    Emit(key, new DoubleWritable(freq))
end procedure

Results

Experimental Setup

- **Input size**: the size of the input file is varied to evaluate the performance of the MapReduce workflow.
  - **Paradise Lost** $\sim$ 310 kB
  - **Moby Dick** $\sim$ 421kB
  - **Frankenstein** $\sim$ 440 kB
  - **Divina Commedia** $\sim$ 600 kB
  - **Gerusalemme Liberata** $\sim$ 691 kB
  - **Promessi Sposi** $\sim$ 1.440 kB
  - **Test file** (random generated sequence of char) of $\sim$ 800 MB.
- **Number of mappers**: Hadoop handles this step
- **Number of reducers**: From one up to three reducers are used for letter frequency

Performance Analysis

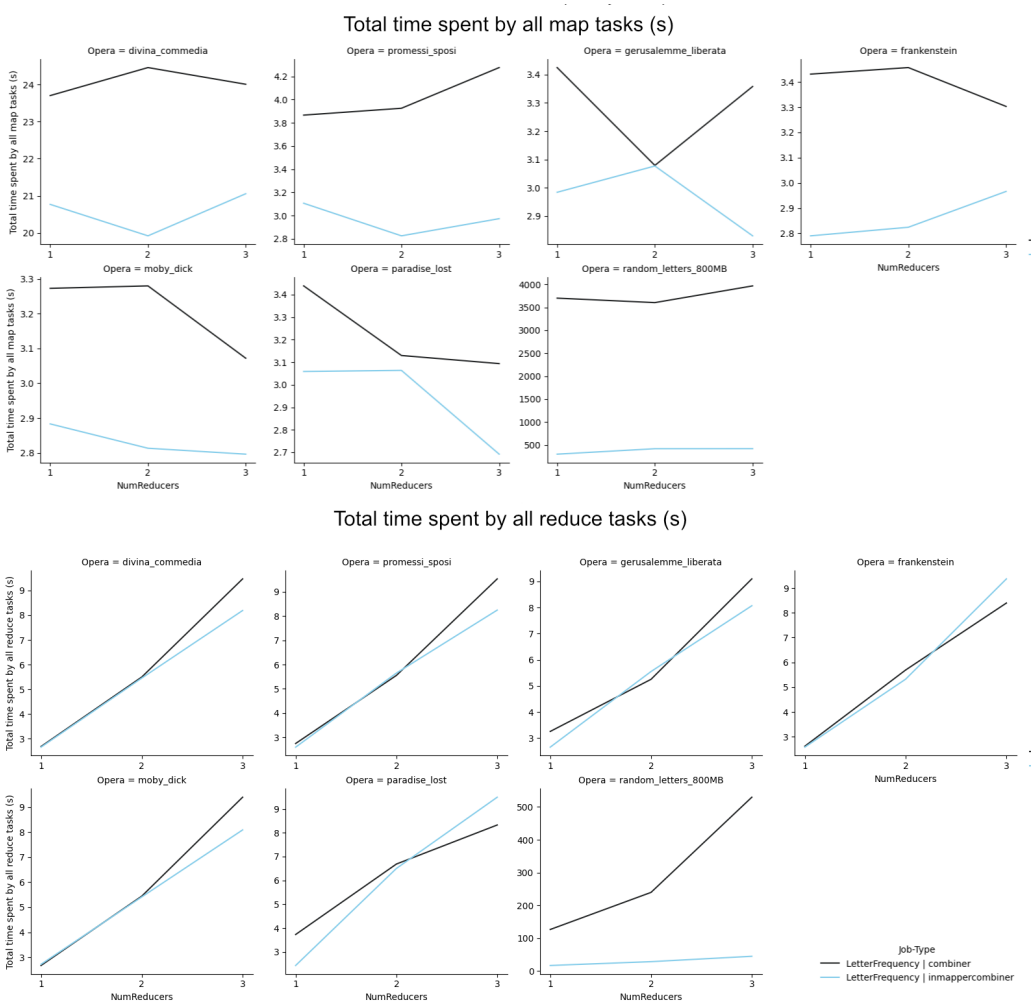

Figure 1: Total Time MapReduce

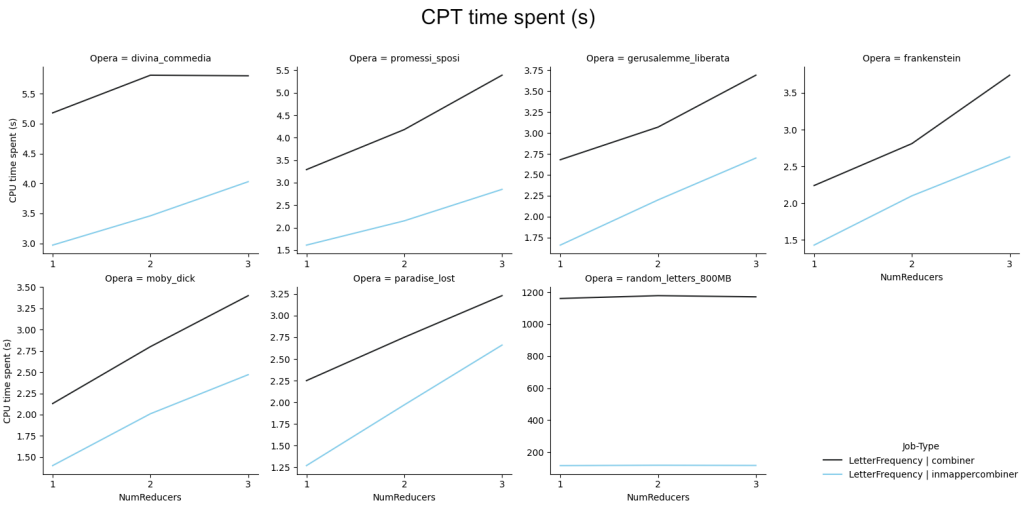

Figure 2: Cpu Time

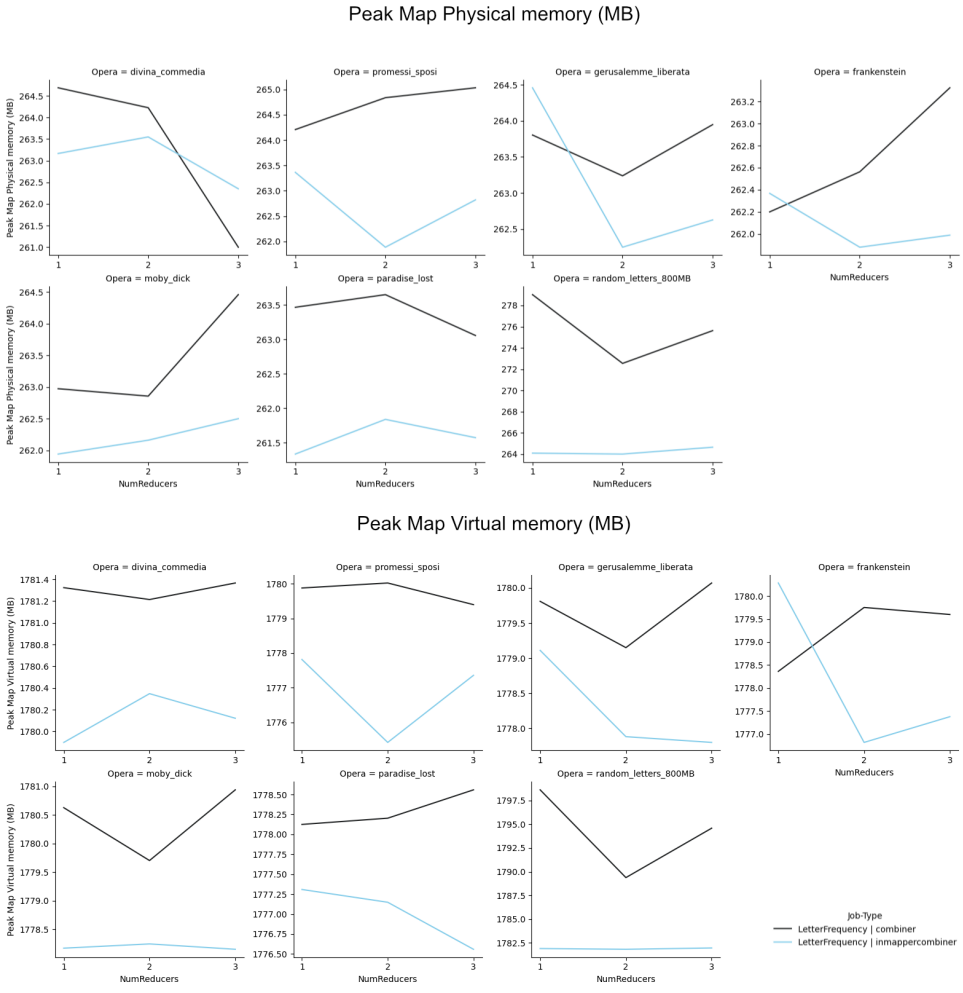

Figure 3: Peak Map Memory

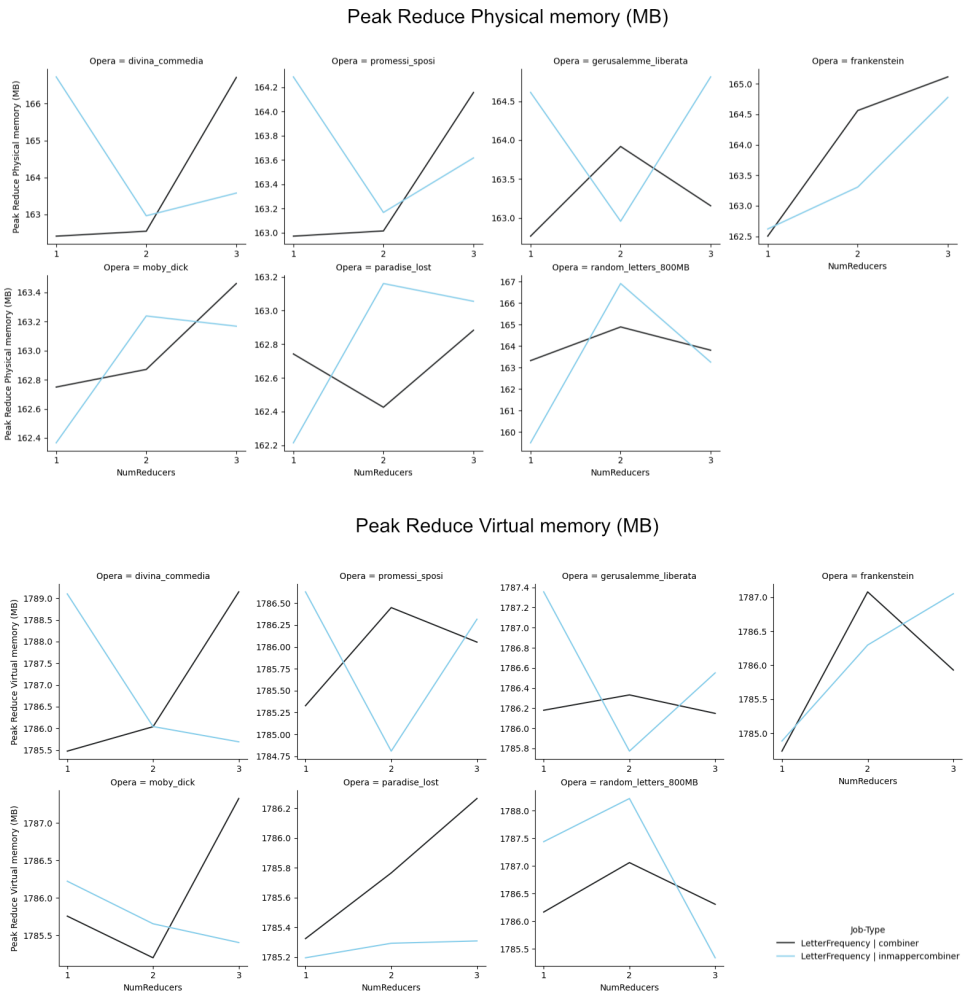

Figure 4: Peak Reduce Memory

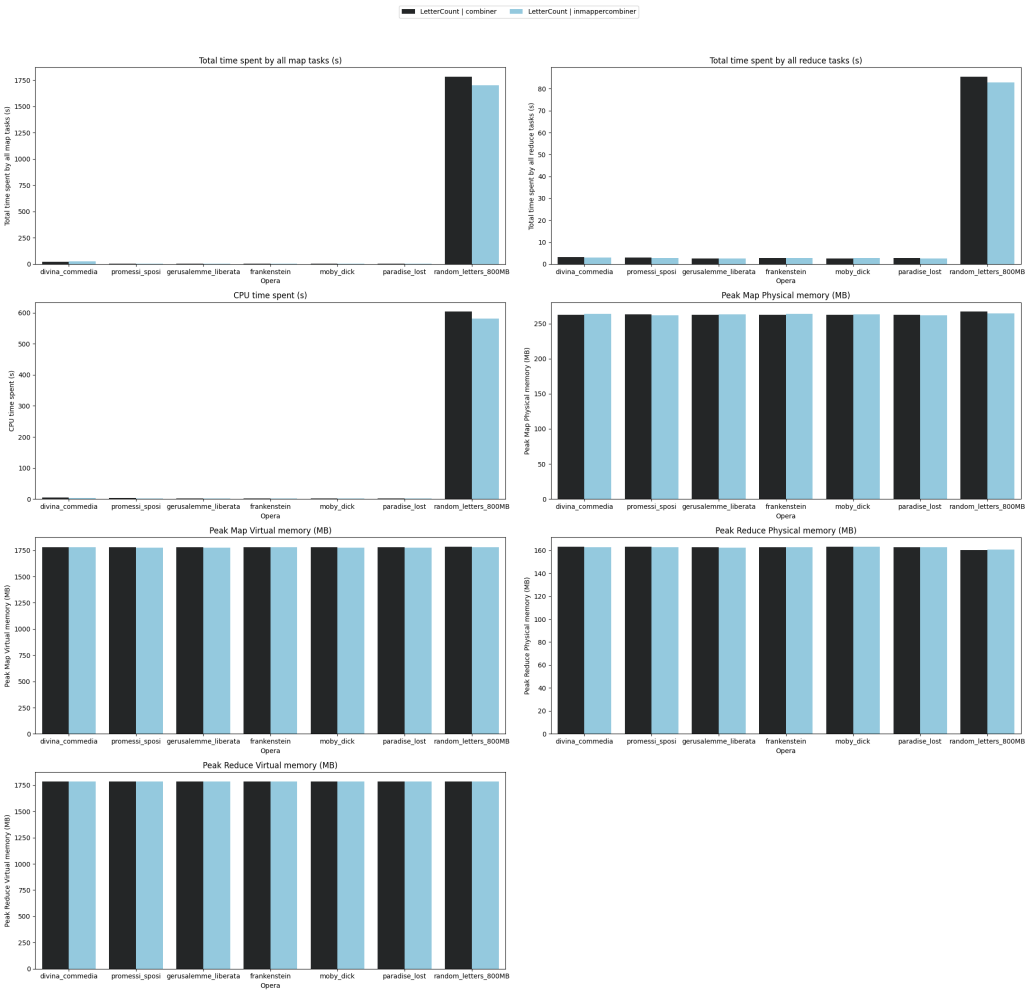

Figure 5: Performance Letter Count

Letter Frequency Analysis

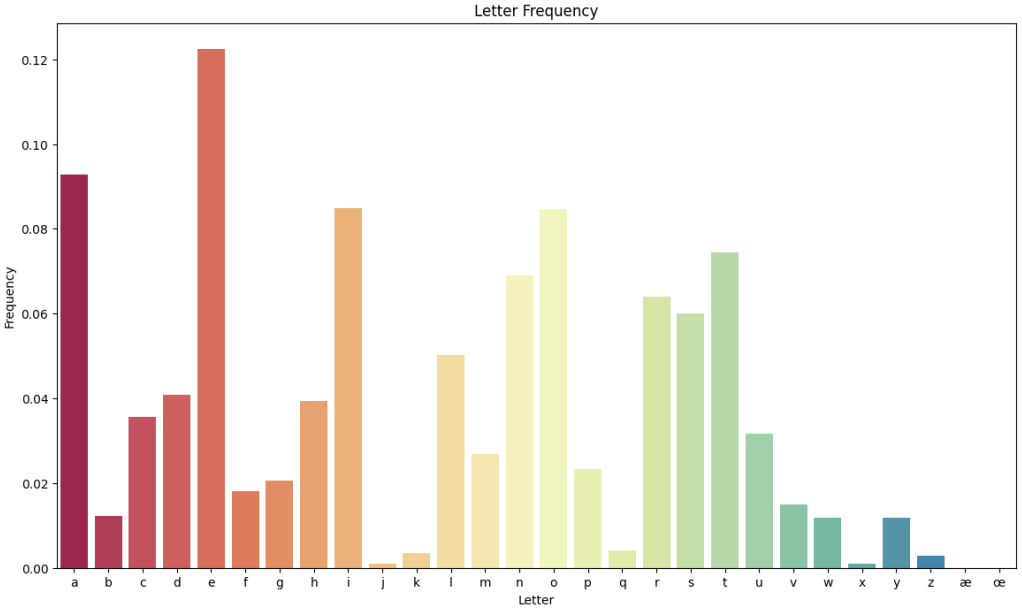

Figure 6: Letter Frequency Distribution

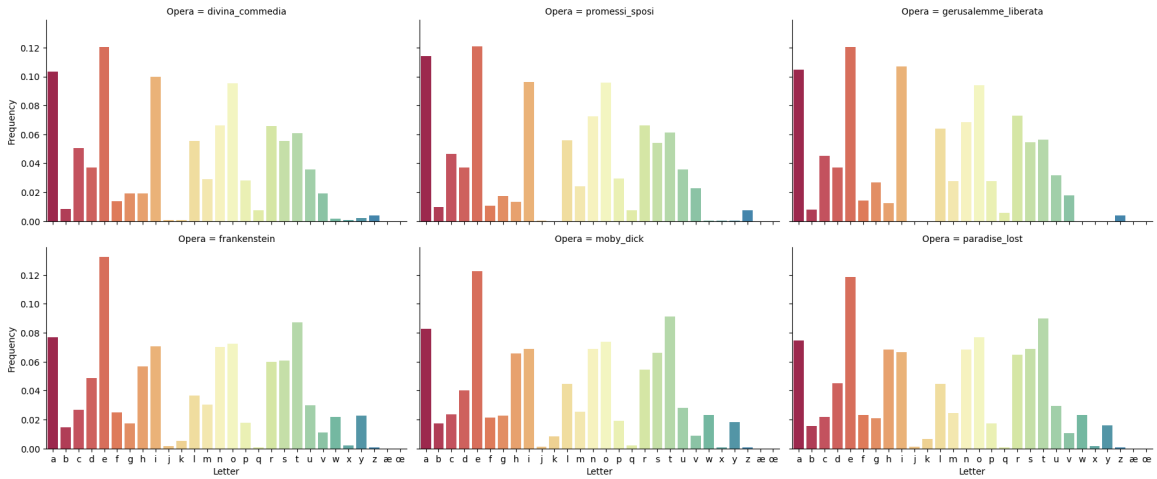

Figure 7: Letter Frequency Distribution for different Operas

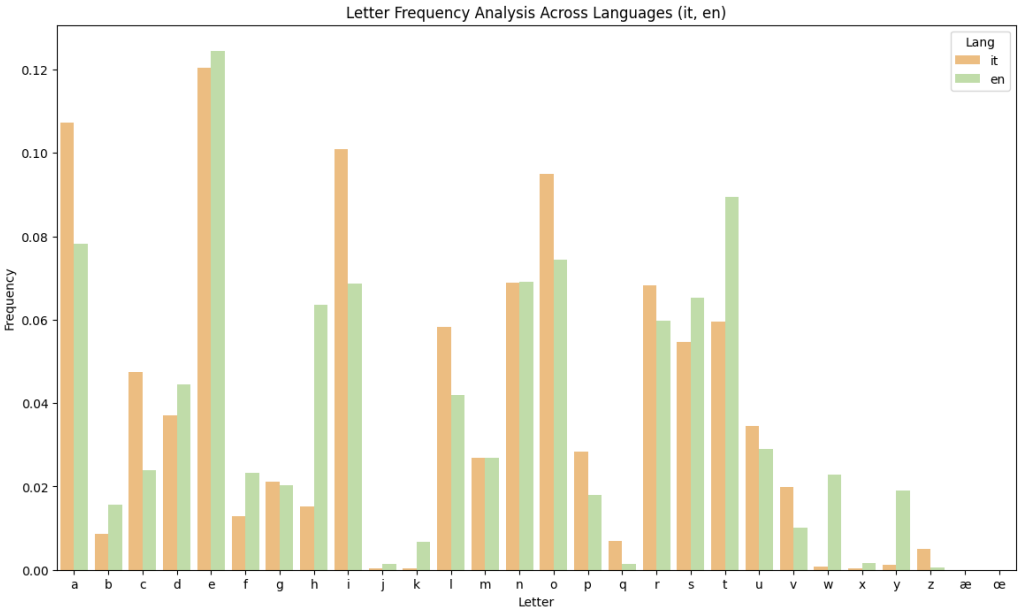

Figure 8: Letter Frequency Distribution for italian and english

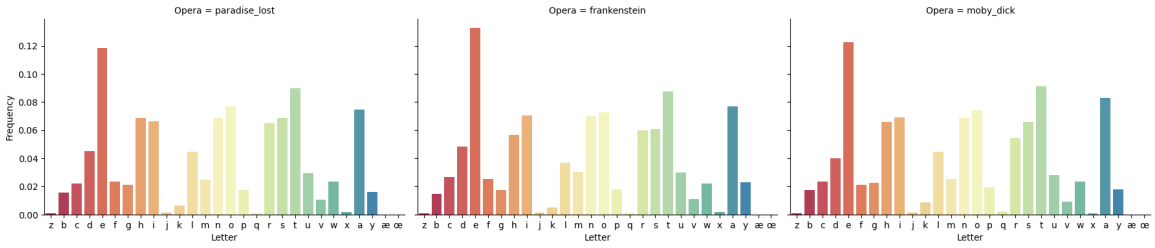

Figure 9: Letter Frequency Distribution for english

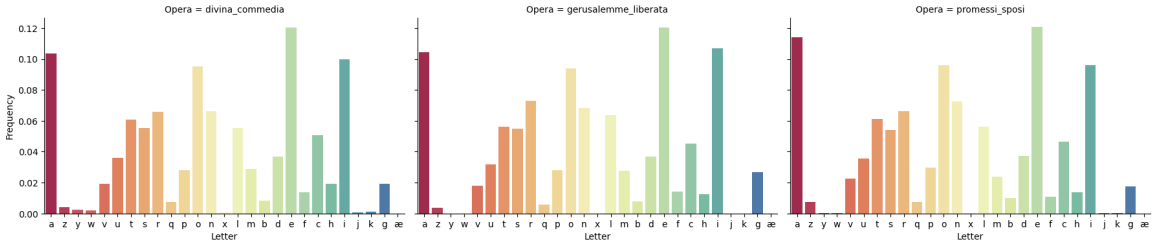

Figure 10: Letter Frequency Distribution for italian

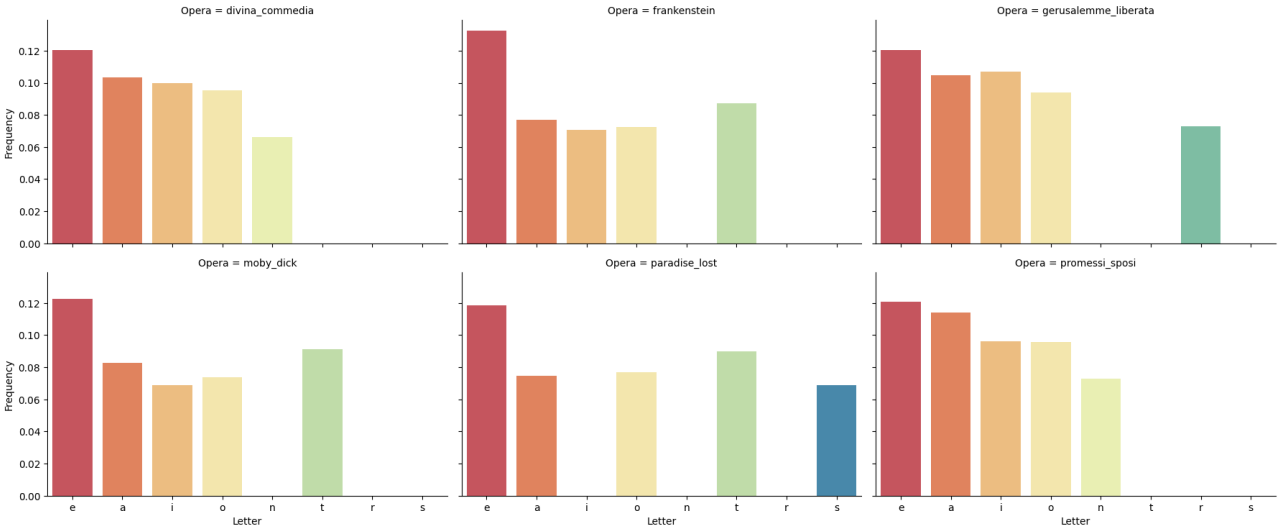

Figure 11: Top 5 letters

Conclusions

Performance

The performances of the letter frequency job and letter count one show the impact of input size and number of reducers. Moreover, the comparison between the Combiner and In-Mapper Combiner implementations highlights the better performances obtained by using the *In-Mapper Combining* design pattern.

- **Impact of Input Size**: larger input sizes, such as the 800 MB test file, greater is the total processing time. This effect is evident across both the Combiner and In-Mapper Combiner implementations.
- **Number of Reducers**: it is shown that increasing the number of reducers do not lead to an overall significant improvement. This is due to the overhead of managing multiple reducers and the limited amount of data to process. In most cases, the performance is optimal with just two reducers.
- **Combiner vs In-Mapper Combiner**: the In-Mapper Combiner approach consistently outperforms the traditional Combiner method. The In-Mapper Combiner design pattern reduces the volume of intermediate data and minimizes data transfer overhead, leading to a more efficient MapReduce workflow.

Qualitative Analysis

The Letter Frequency Analysis provides significant insights into the variations and trends in letter usage across different texts and languages.

- **Letter Frequency Distribution (Figure 6)**: the general letter frequency distribution graph highlights the overall trends in letter usage across the analyzed documents. Common letters like /e/, /a/, /t/, /i/, and /o/ show high frequencies, consistent with typical linguistic patterns in English and other Latin-based languages.
- **Comparison Across Different Operas (Figure 7)**: when comparing letter frequency distributions across different operas, noticeable variations can be seen. These differences may reflect the unique stylistic and linguistic choices of the authors, as well as the historical and cultural contexts in which these operas were written. For instance, certain operas may exhibit a higher frequency of specific letters due to their thematic or lexical peculiarities.
- **Italian vs. English (Figures 8-10)**: the comparison between Italian and English texts reveals distinct frequency patterns. Italian texts, for instance, tend to have higher frequencies of vowels such as /a/ and /o/, which are more prevalent in Italian phonology. Conversely, English texts show a higher occurrence of consonants like /t/ and /h/. These differences underscore the phonetic and orthographic characteristics inherent to each language.
- **Top 5 Letters (Figure 11)**: the analysis of the 5 most frequent letters across the documents provides further insights into common linguistic elements. In English, letters like /e/, /t/, and /a/ dominate due to their essential roles in constructing common words and grammatical structures. In Italian, vowels play a more prominent role, reflecting the language's phonetic structure.

In conclusion, the qualitative analysis of letter frequency data across different texts and languages highlights both universal and language-specific patterns. These findings not only reveal underlying linguistic trends but also enhance our understanding of how language and stylistic elements vary across different cultural and historical contexts.