IN SUPREMÆ DIGNITATIS
1343

UNIVERSITY OF PISA

MSc in Artificial Intelligence and Data Engineering

Cloud Computing

Letter Frequency Analysis through Hadoop MapReduce

Project Report

Emanuele Respino

Christian Petruzzella

Alessandro Ciarniello

ACADEMIC YEAR 2023/2024

Contents

Chapter 1

Algorithm Design

The goal of this study is to provide the computing of letter frequency, given an input test: a **workflow** is required, allowing firstly the *letter counting* in the text, and then the *frequency computing* of each letter, based on results of the first job. This project provides the development and comparison of **two different workflow implementations**, which pseudo-codes are provided as following.

In *Letter counting*, a **Partitioner** has been implemented to equally distribute keys among each reducer: otherwise, due to the fact that *reducer key* has always the same value, this would result in having all the pairs redirected to the same reducer, nullifying the benefits of exploiting more reducers.

During the project, some *scripts* (bash files and Jupyter notebooks) have been implemented, leading to an easier plotting of results and a faster execution of the workflow with several configurations and different inputs.

MapReduce with Combiner

This is the classic implementation, in which a Combiner function is built to be optionally executed for local aggregation.

Algorithm 1 LetterCount with Combiner

```
Require: Txt file
Ensure: Total number of characters belonging to English alphabet

Class MAPPER
charPattern ← REGEX("[a-z]", caseInsensitive)
procedure MAP(docid key, doc value)
    line ← NORMALIZE(value)                     ▷ Remove accents and set lowercase
    for each character $c$ in line do
        if $c$ matches charPattern then           ▷ Focus only on English alphabet
            EMIT($\perp$, count 1)

Class PARTITIONER                                         ▷ For load balancing
procedure PARTITION(id $\perp$, count $c$, count numReducer)
    return RANDOM( ) * numReducer

Class REDUCER                                   ▷ Used also as Combiner class
procedure REDUCE(id $\perp$, counts $[c_1, c_2, ...]$)
    sum ← 0
    for each count $c$ in $\in$ counts $[c_1, c_2, ...]$ do
        sum ← sum + $c$
    EMIT(id $\perp$, count sum)
```

Algorithm 2 LetterFrequency with Combiner

Require: Txt file, Total number of characters in the txt file

Ensure: Frequency of each letter in the input file

```
Class MAPPER
$charPattern$ ← REGEX("[a-z]", caseInsensitive)
procedure MAP(docid $key$, doc $value$)
    $line$ ← NORMALIZE(value)                  ▷ Remove accents and set lowercase
    for each character $c$ in $line$ do
        if $c$ matches $charPattern$ then        ▷ Focus only on English alphabet
            EMIT(character $c$, count 1)

Class COMBINER
procedure COMBINE(character $c$, counts $[c_1, c_2, ...]$)
    $sum$ ← 0
    for each count $c$ in ∈ counts $[c_1, c_2, ...]$ do
        $sum$ ← $sum + c$
    EMIT(character $c$, count $sum$)

Class REDUCER
procedure INITIALIZE(Context $context$)
    TEXT_LENGTH ← GETTEXTLENGTH($context$)   ▷ Retrieve configuration
procedure REDUCE(character $c$, counts $[c_1, c_2, ...]$)
    sum ← 0
    for each count $c$ in ∈ counts $[c_1, c_2, ...]$ do
        $sum$ ← $sum + c$
    $freq$ ← $sum$ / TEXT_LENGTH
    EMIT(character $c$, frequency $freq$)
```

In-Mapping Combining

In-Mapper Combining design pattern provides local aggregation directly in the Mapper procedure, resulting in the complete control of the local aggregation process.

Algorithm 3 LetterCount with In-Mapper Combining

```
Require: Txt file
Ensure: Total number of characters belonging to English alphabet

Class Mapper
procedure Initialize
    $sum \leftarrow 0$
procedure Map(docid key, doc value)
    line ← Normalize(value)                  ▷ Remove accents and set lowercase
    for each character $c$ in line do
        if $c$ matches charPattern then       ▷ Focus only on English alphabet
            $sum \leftarrow sum + 1$
procedure Close
    Emit(id ⊥, count sum)

Class Partitioner                                        ▷ For load balancing
procedure Partition(id ⊥, count $c$, count numReducer)
    return Random( ) * numReducer

Class Reducer
procedure Reduce(id ⊥, counts $[c_1,\ c_2,...]$)
    $sum \leftarrow 0$
    for each count $c$ in $\in$ counts $[c_1,\ c_2,...]$ do
        $sum \leftarrow sum + c$
    Emit(id ⊥, count sum)
```

Algorithm 4 LetterFrequency with In-Mapping Combining

```
Require: Txt file, Total number of characters in the txt file
Ensure: Frequency of each letter in the input file
Class Mapper
procedure SETUP
    map ← new AssociativeArray
procedure MAP(docid key, doc value)
    line ← NORMALIZE(value)                  ▷ Remove accents and set lowercase
    for each character c in line do
        if c matches charPattern then        ▷ Focus only on English alphabet
            map{c} ← map{c} + 1
procedure CLEANUP
    for each character $c \in$ map do
        EMIT(character c, count map{c})

Class REDUCER
procedure INITIALIZE(Context context)
    TEXT_LENGTH ← GETTEXTLENGTH(context)  ▷ Retrieve configuration
procedure REDUCE(character c, counts [$c_1$, $c_2$,...])
    sum ← 0
    for each count c in ∈ counts [$c_1$, $c_2$,...] do
        sum ← sum + c
    freq ← sum / TEXT_LENGTH
    EMIT(character c, frequency freq)
```

Chapter 2

Results

Experimental Settings

Two different analysis have been provided:

- **Performances**: Executing Map-Reduce workflow with different configurations leads to a better comprehension of its behavior:
  - *Type of workflow*: with Combiner, In-Mapper Combining;
  - *Size of input data*: 15.2KB, 2.93MB, 2.82GB (randomly generated);
  - *Number of Reducers*: 1, 2, 3.
- **Letter frequency**: using a fixed configuration, the letter frequency of several languages have been compared, using the same text as reference, in order to understand similarities and differences between each other:
  - *Type of workflow*: In-Mapper Combining;
  - *Number of Reducers*: 3;
  - *Input text*: The Bible (∼4.2MB), from Bible SuperSearch API;
  - *Languages*: Afrikaans, Albanian, Czech, Dutch, English, Finnish, French, German, Hungarian, Indonesian, Italian, Maori, Polish, Portuguese, Romanian, Spanish, Turkish.

Performance Evaluation

For the performance evaluation, 5 parameters have been considered among the ones Hadoop provides at the end of each Map-Reduce job:

- **Total time spent by all map tasks** (ms);
- **Total time spent by all reduce tasks** (ms);

- **CPU time spent** (ms);
- **Garbage Collection (GC) time elapsed** (ms);
- **Reduce shuffle bytes**.

The results of those parameters for each configuration are shown in the following charts.

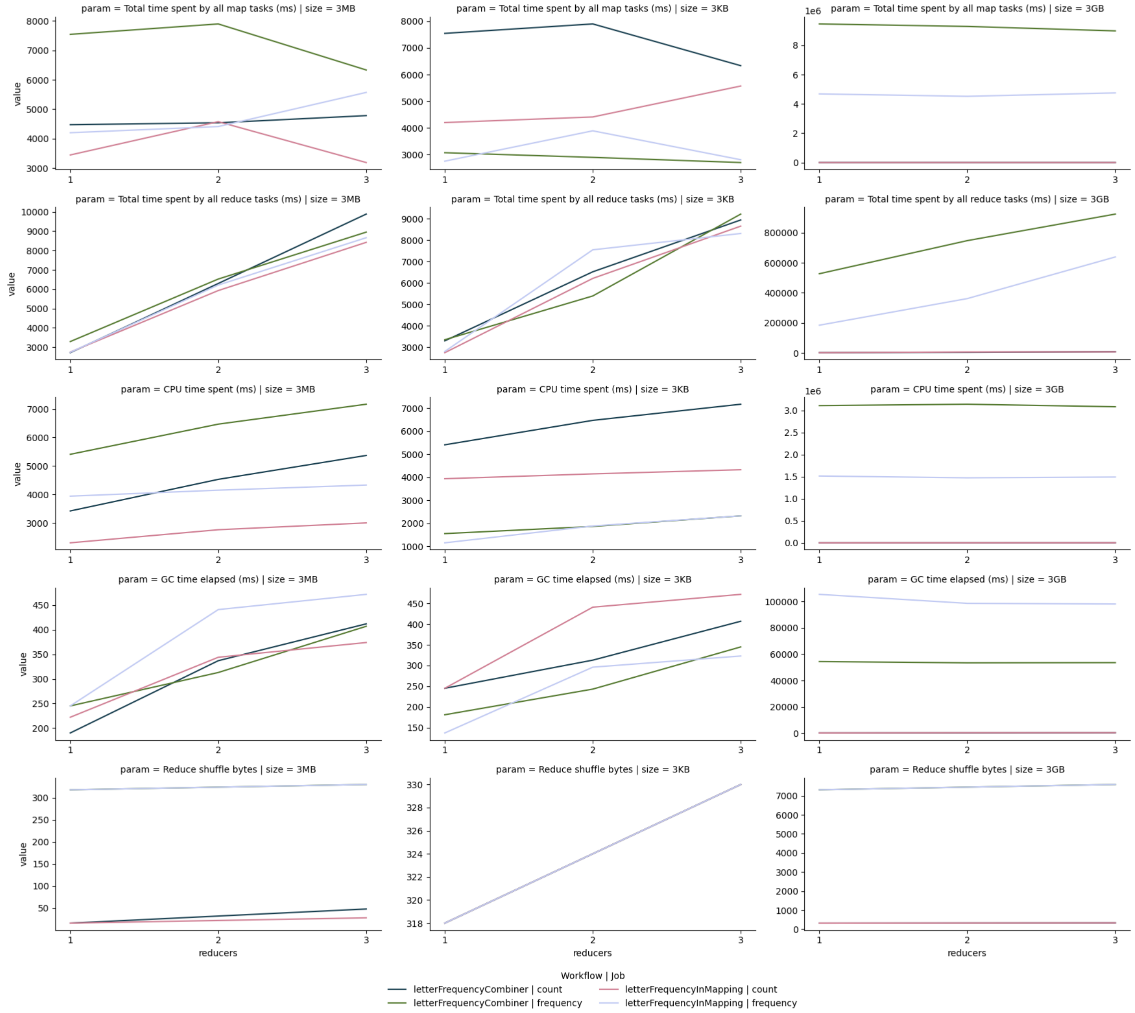


Figure 2.1: Performance comparison on some parameters for each configuration

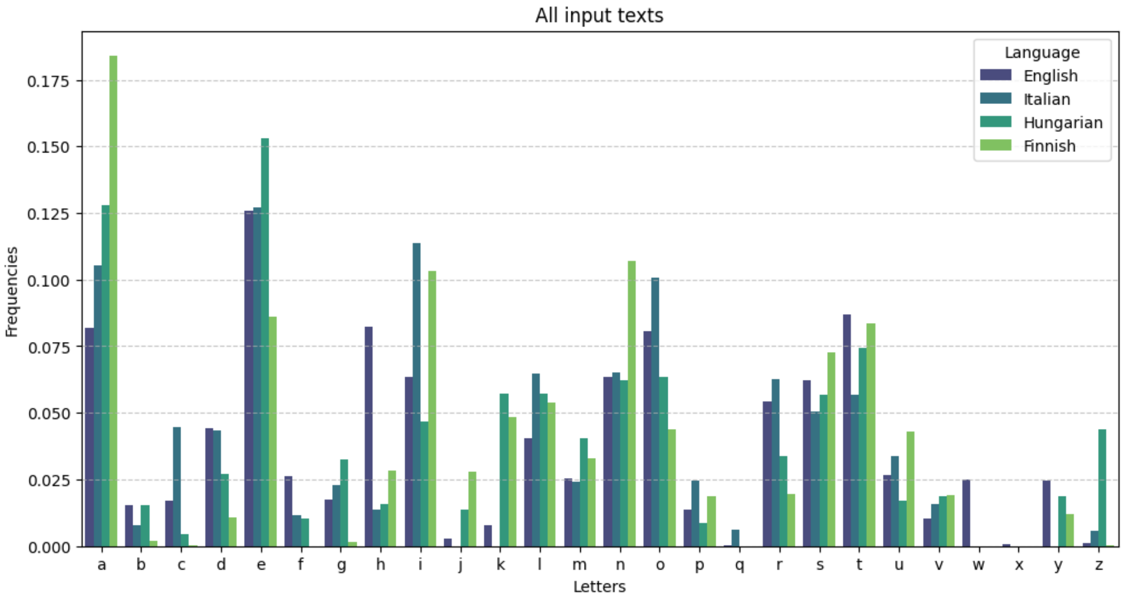
Workflow. The results obtained show that the usage of In-Mapper Combining reduces execution time, especially with larger input files. On the other hand this implementation leads to a much higher *GC time elapsed* value, showing that it is more memory consuming rather than the implementation with the Combiner option.

Reducers. Considering the specific purpose of this study, it is shown that increasing the number of reducers do not lead to an overall significant improvement. Actually, for input texts of lower dimension, using a higher number reducers is less time efficient.

Letter Frequency

The analysis has been conducted for all the languages previously mentioned.

A more detailed comparison is reported, highlighting the differences and similarities among four European languages: **English**, **Italian**, **Hungarian** and **Finnish**.



This report shows some interesting results:

- **Most common letters**: ***E*** is the most common letter in English, Italian and Hungarian, while for Finnish it is the letter ***A***: this can indicate differences it the vowel structure between this language and the others.
- **Rare letters**: ***Q***, ***W***, ***X***, ***Z*** tends to be rare or absent, except for ***Z*** in Hungarian and ***W*** in English.
- **Peaks in frequencies**: ***C*** is relatively more used in Italian rather than in the other languages, as for ***H*** in English and ***N*** in Finnish.

The following graphs focus on the most common letters for each language.

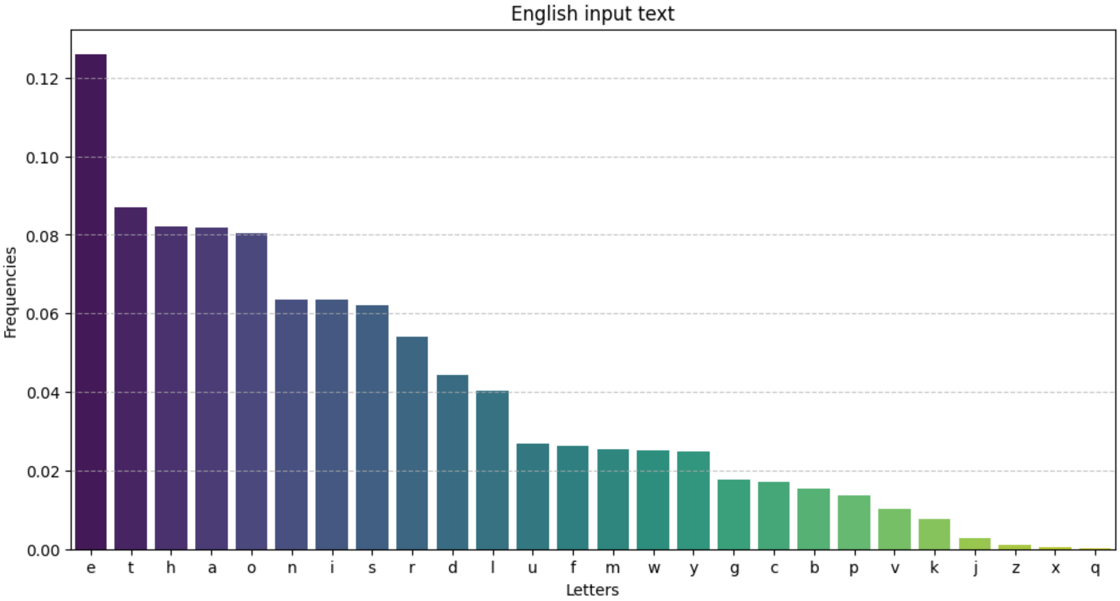


Figure 2.2: English most-common letters

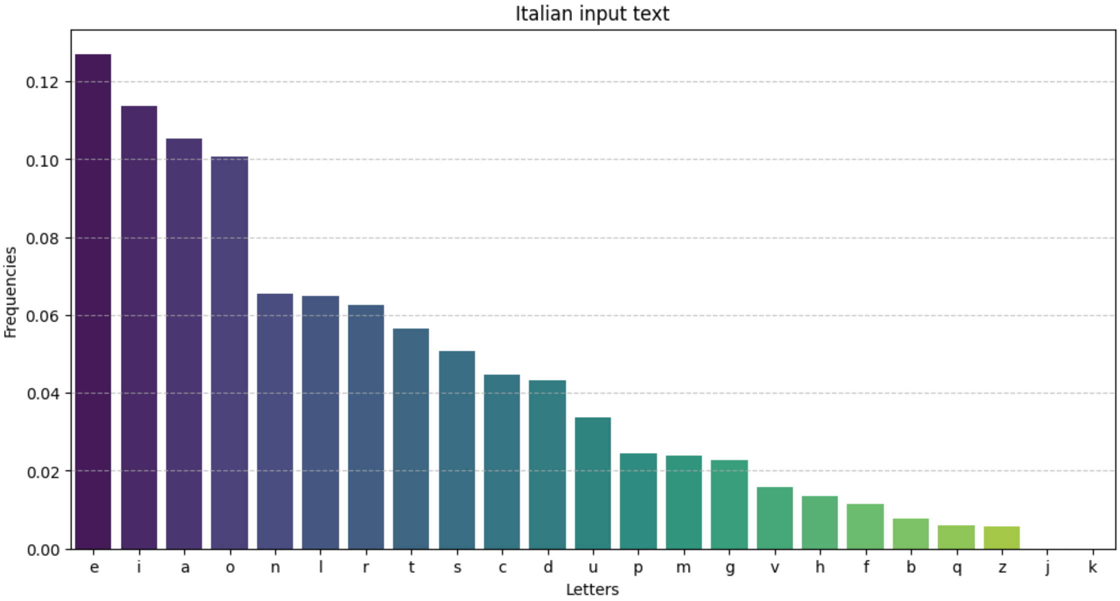


Figure 2.3: Italian most-common letters

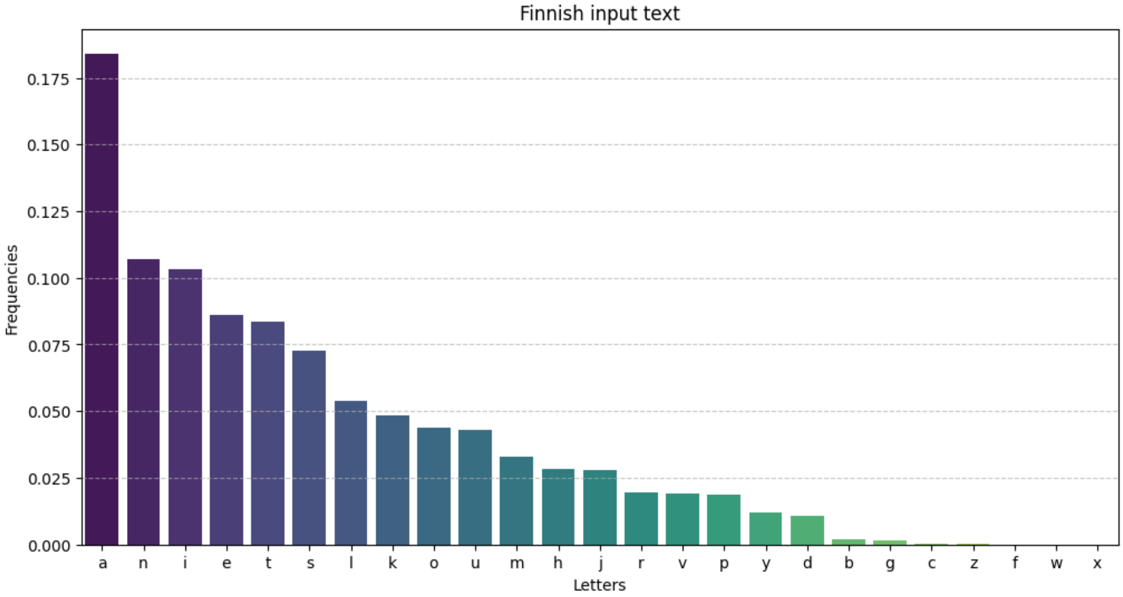


Figure 2.4: Finnish most-common letters

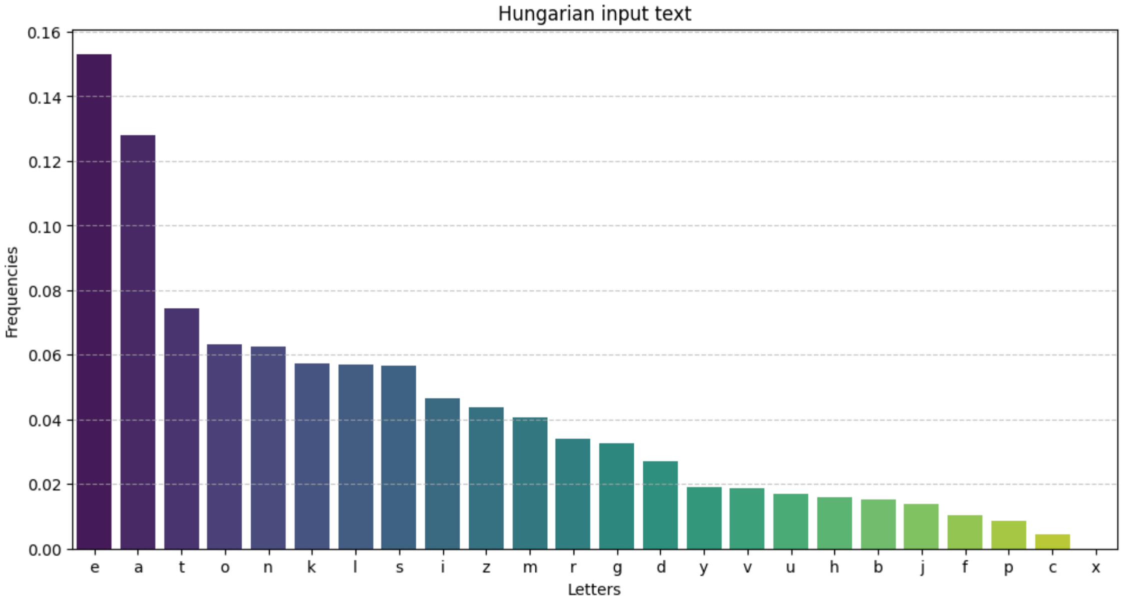


Figure 2.5: Hungarian most-common letters