Cloud Computing - Project Report

Totoro Group

Zahra Omrani
z.omrani@studenti.unipi.it

Paolo Palumbo
p.palumbo3@studenti.unipi.it

Ettore Ricci
e.ricci32@studenti.unipi.it

June 14, 2024

https://github.com/Etto48/CloudComputingProject

Abstract

This report presents the development and performance evaluation of a Java-based application designed to count letter frequencies in large datasets using the Hadoop framework. To provide a comprehensive analysis, we also implemented and tested non-distributed applications in Python and Rust for comparison. The results show that for the sizes of the datasets considered, Java and Python execution times are comparable, while Rust is significantly faster. Both Python and Rust applications used significantly less memory than the Java application.

1 Introduction

Our project for the Cloud Computing course consists of developing a Java application to count the frequency of letters in a large dataset using the Hadoop framework. The main objective of the project is to analyze and compare the performance of the application with different configurations and input sizes and also to compare it with non-distributed implementations in Python and Rust.

2 Mapreduce

MapReduce is a programming model for processing large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. For our application we needed two subsequent MapReduce jobs: the first one to count the occurrences of each letter in the input text and the second one to calculate the frequency of each letter.

2.1 Job 1: Counting letters

2.1.1 Mapper

For testing purposes, we created two different mappers: one with an in-mapper combiner and one without. For the combiner (when enabled), we used the same code as the reducer.

- **Mapper with in-mapper combiner**: This mapper stores the letter counts in a vector and emits them in the cleanup method. At the end of the cleanup method, the mapper increments a job counter with the number of emitted letters. (Listing 1)
- **Mapper without in-mapper combiner**: This mapper emits the letter counts (with value 1) for each letter in the map method. At the end of the map method, the mapper increments a job counter with the number of emitted letters. (Listing 2)

2.1.2 Reducer

The reducer receives the letter counts from the mappers and sums them up. (Listing 3)

2.2 Job 2: Calculating frequencies

The results of the first job are saved in a temporary file and used as input for the second job. The counter obtained from the first job is stored in a configuration variable that is passed to the reducer (only one is needed because the number of different letters is small). For this job we do not need a combiner so it's always disabled.

2.2.1 Mapper

We used an identity mapper that emits the letter counts as they are.

2.2.2 Reducer

The reducer receives the letter counts from the mappers and calculates the frequency of each letter using the total count that was previously stored in a configuration variable. (Listing 4)

3 Dataset

We used three different datasets:

- **english.txt**: A 1.2GB text file containing text from the books on Gutenberg project.
- **italian.txt**: A 1.3GB text file from the PAISÀ corpus[1].
- **spanish.txt**: A 130MB text file from the Leipzig Corpora Collection[2].

We also created 11 files of increasing size (from 100MB to 1.1GB with 100MB steps) to test the performance of the application with different input sizes. These files are called **part_*X*MB.txt** and contain the first *X*MBs of the **english.txt** file.

4 Experiments

We wrote our program in a way that allows us to configure it for the different tests with command line arguments:

- $-i|--input$: Input file path inside the HDFS.
- $-r|--reducers$: Number of reducers to use in the first job.
- $-n|--no-combiner$: Disable the combiner in the first job.
- $-m|--no-in-mapper-combiner$: Disable the in-mapper combiner in the first job.
- $-o|--output$: Output directory path inside the HDFS, by default it's set to **output**.
- $-t|--tmp$: Temporary directory path inside the HDFS, by default it's set to **tmp**.

We carried out 21 tests with the configurations shown in Table 1 and Table 2. Originally we ran tests 7, 8 and 9 with 4 reducers, but test 9 could never complete because of a Shuffle Error caused by a Java Heap Space error (Out Of Memory Error). Because of this, we decided to run these tests again with 3 reducers. Only the 3 reducers configuration will be shown in the results. We also ran the Python and Rust applications with the same input file (english.txt).

Test ID	Arguments
0	-i english.txt -r 1
1	-i english.txt -r 2
2	-i english.txt -r 4
3	-i english.txt -r 8
4	-i english.txt -r 1 –no-combiner
5	-i english.txt -r 1 –no-in-mapper-combiner
6	-i english.txt -r 1 –no-in-mapper-combiner –no-combiner
7	-i english.txt -r 3 –no-combiner
8	-i english.txt -r 3 –no-in-mapper-combiner
9	-i english.txt -r 3 –no-in-mapper-combiner –no-combiner

Table 1: General tests

Test ID	Arguments
10	-i part_100MB.txt -r 1
11	-i part_200MB.txt -r 1
12	-i part_300MB.txt -r 1
13	-i part_400MB.txt -r 1
14	-i part_500MB.txt -r 1
15	-i part_600MB.txt -r 1
16	-i part_700MB.txt -r 1
17	-i part_800MB.txt -r 1
18	-i part_900MB.txt -r 1
19	-i part_1000MB.txt -r 1
20	-i part_1100MB.txt -r 1

Table 2: Input split tests

5 Results

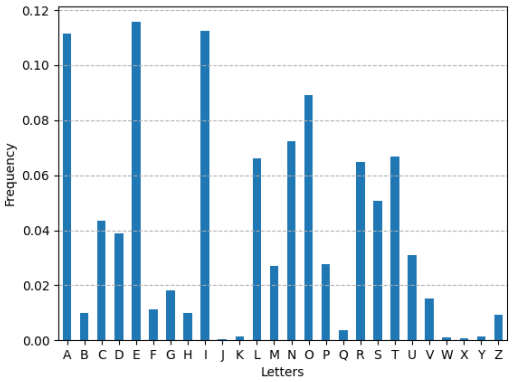

Figure 2: Italian letter frequency

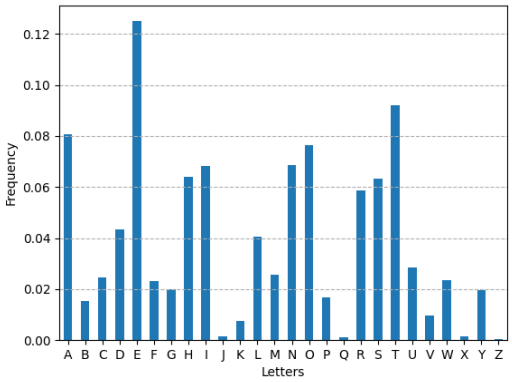

Figure 1: English letter frequency

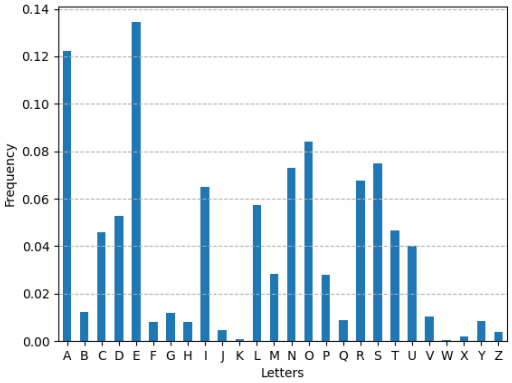

Figure 3: Spanish letter frequency

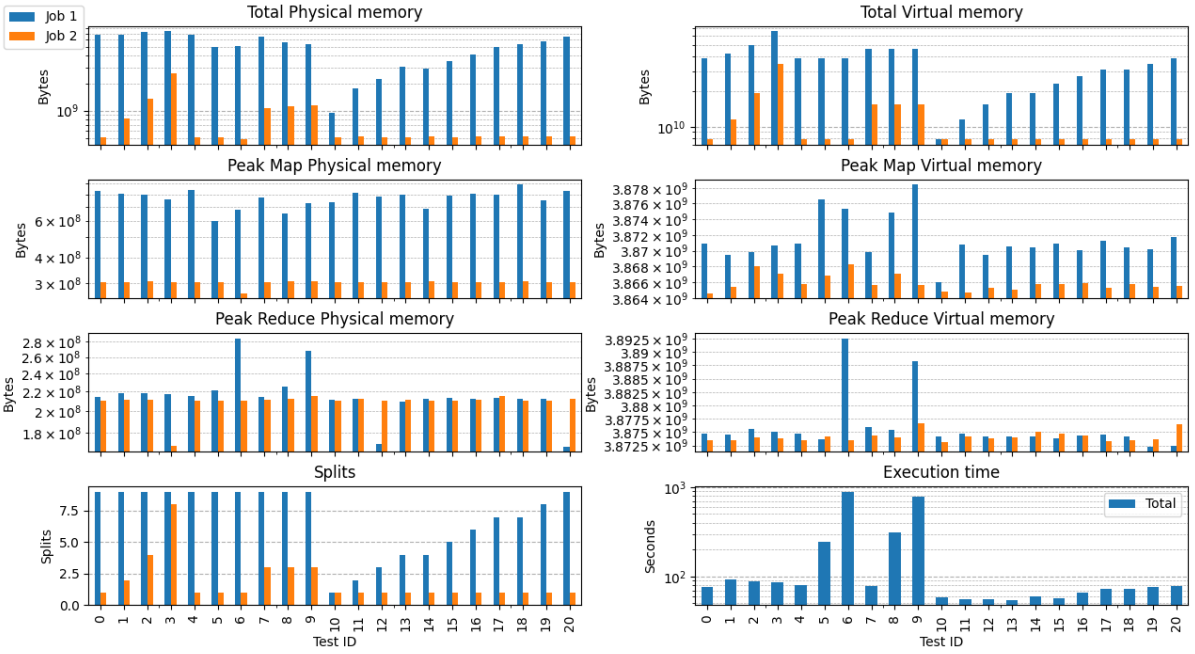

Figure 4: Tests results, the x-axis represents the test ID.

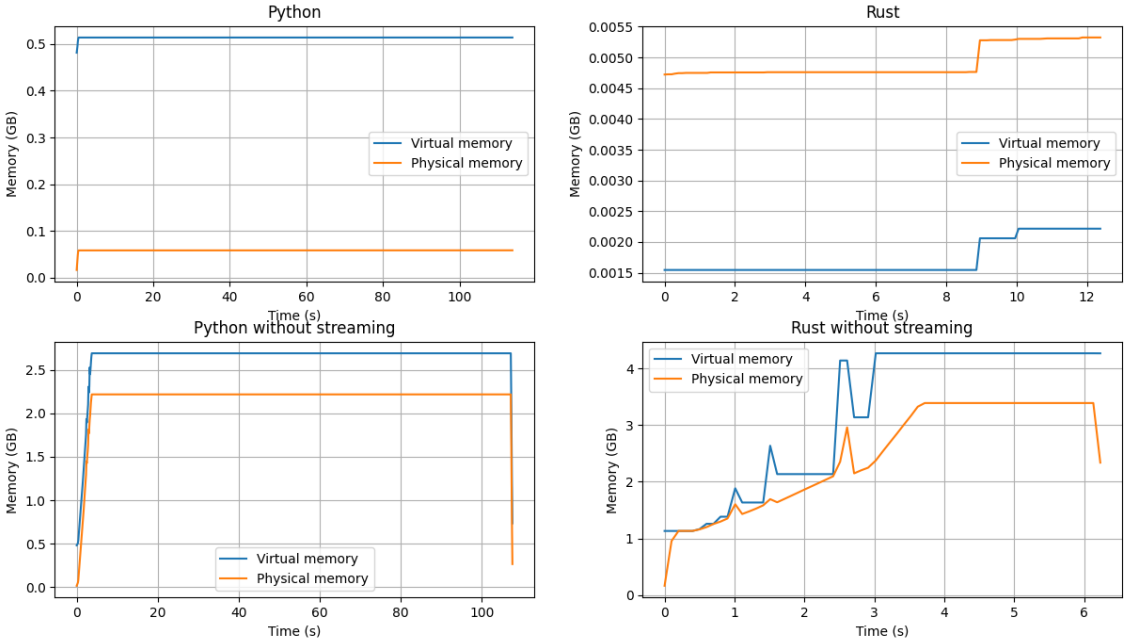

Figure 5: Python and Rust memory usage over time, with and without file streaming.

6 Conclusions

7 Listings

Listing 1: Mapper with in-mapper combiner

```
1 package it.unipi;
2
```

```java
import java.io.IOException;
import java.util.Vector;

import org.apache.commons.lang3.StringUtils;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

public class Mapper extends org.apache.hadoop.mapreduce.Mapper<Object, Text, Char, LongWritable> {
    private final static int LETTER_COUNT = 26;
    private final Vector<Integer> combinerCounters = new Vector<Integer>();
    private long sum = 0;
    private final Char key = new Char();
    private final LongWritable value = new LongWritable();

    @Override
    public void setup(Context context) {
        sum = 0;
        combinerCounters.clear();
        for (int i = 0; i < LETTER_COUNT; i++) {
            combinerCounters.add(0);
        }
    }

    @Override
    public void cleanup(Context context) throws IOException, InterruptedException {
        for (int i = 0; i < LETTER_COUNT; i++) {
            key.set((char)('a' + i));
            value.set(combinerCounters.get(i));
            context.write(key, value);
        }
        context.getCounter("LetterFreq", "total").increment(sum);
    }

    @Override
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        StringUtils.stripAccents(value.toString()).chars().forEach(c -> {
            int lowerC = Character.toLowerCase(c);
            if (lowerC >= 'a' && lowerC <= 'z') {
                int index = lowerC - 'a';
                combinerCounters.set(index, combinerCounters.get(index) + 1);
                sum++;
            }
        });
    }
}
```

Listing 2: Mapper without in-mapper combiner

```java
package it.unipi;

import java.io.IOException;

import org.apache.commons.lang3.StringUtils;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

public class MapperNoCombiner extends org.apache.hadoop.mapreduce.Mapper<Object, Text, Char, LongWritable> {
    private final Char key = new Char();
    private final LongWritable one = new LongWritable(1);

```

```
    @Override
    public void map(Object key, Text value, Context context) throws IOException
        , InterruptedException {
        long sum = 0;
        String normalizedInput = StringUtils.stripAccents(value.toString());
        for (int i = 0; i < normalizedInput.length(); i++) {
            int lowerC = Character.toLowerCase(normalizedInput.charAt(i));
            if (lowerC >= 'a' && lowerC <= 'z') {
                this.key.set((char)lowerC);
                context.write(this.key, this.one);
                sum++;
            }
        };
        context.getCounter("LetterFreq", "total").increment(sum);
    }
}
```

Listing 3: Reducer

```
package it.unipi;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

public class Reducer extends org.apache.hadoop.mapreduce.Reducer<
    Char,
    LongWritable,
    Char,
    LongWritable>
{
    private final LongWritable result = new LongWritable();
    public void reduce(Char key, Iterable<LongWritable> values, Context context
        )
        throws IOException, InterruptedException {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

Listing 4: ReducerFrequency class, used to calculate the frequency of each letter

```
package it.unipi;

import java.io.IOException;

import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;

public class ReducerFrequency extends org.apache.hadoop.mapreduce.Reducer<
    Char,
    LongWritable,
    Char,
    DoubleWritable>
{
    private long total;

    @Override
    public void setup(Context context) {
        total = context.getConfiguration().getLong("total", 0);
```

```
        if (total == 0) {
            throw new RuntimeException("Total count is zero");
        }
    }

    private final DoubleWritable result = new DoubleWritable();
    public void reduce(Char key, Iterable<LongWritable> values, Context context
        )
        throws IOException, InterruptedException
    {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        result.set((double)sum / total);
        context.write(key, result);
    }
}
```

Listing 5: Char class, used as a custom key

```
package it.unipi;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;

public class Char implements WritableComparable<Char> {
    private char c;

    public Char() {
        c = 0;
    }

    public Char(char c) {
        this.c = c;
    }

    public void set(char c) {
        this.c = c;
    }

    public char get() {
        return c;
    }

    @Override
    public void readFields(DataInput arg0) throws IOException {
        c = arg0.readChar();
    }

    @Override
    public void write(DataOutput arg0) throws IOException {
        arg0.writeChar(c);
    }

    @Override
    public int compareTo(Char o) {
        return Character.compare(c, o.c);
    }

    @Override
```

```
    public boolean equals(Object o) {
        if (o instanceof Char) {
            return c == ((Char)o).c;
        }
        return false;
    }

    @Override
    public int hashCode() {
        return Character.hashCode(c);
    }

    @Override
    public String toString() {
        return Character.toString(c);
    }
}
```

References

[1] V. Lyding, E. Stemle, C. Borghetti, M. Brunello, S. Castagnoli, F. Dell'Orletta, H. Dittmann, A. Lenci, and V. Pirrelli, "The PAISÀ corpus of Italian web texts," in *Proceedings of the 9th Web as Corpus Workshop (WaC-9)*, F. Bildhauer and R. Schäfer, Eds. Gothenburg, Sweden: Association for Computational Linguistics, Apr. 2014, pp. 36–43. [Online]. Available: https://aclanthology.org/W14-0406

[2] T. Eckart and U. Quasthoff, *Statistical Corpus and Language Comparison on Comparable Corpora*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 151–165. [Online]. Available: https://doi.org/10.1007/978-3-642-20128-8_8