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 13, 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.

2.1 Mapper

The Mapper class is responsible for processing the input data and generating a set of key/value pairs. In our case the input data is a piece of text, the key is a letter and the value is the number of occurrencies of that letter. To count the occurrencies of each letter we first transformed the input text to ascii (normalizing accented letters) and then we counted the occurrencies of each letter. For this project we tested two different mappers:

- **In-mapper combiner**: (Listing 1) For the in-mapper combiner, we used a vector of size 26 to store the frequency of each letter and another variable to store the total number of letters. During the map phase, we update the vector and the total count for each letter. In the cleanup phase, we emith for each letter its occurrencies and the total count.
- **No in-mapper combiner**: (Listing 2) For the mapper without in-mapper combiner, we emit a key-value pair for each letter in the input text containing the letter and the number 1.

The key is a custom class called Char (Listing 5) and the value is a custom class called CountTotalPairWritable (Listing **??**). We used custom classes to optimize the memory usage and the serialization/deserialization process.

2.2 Reducer

(Listing 3) The Reducer class is responsible for processing the intermediate key/value pairs generated by the Mapper and generating the final output. In our case, the reducer receives a letter and a list of pairs of its occurrencies and total letters count. The reducer sums the occurrencies of each letter and then calculates the frequency of each letter.

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.

- **spanish.txt**: A 130MB text file from the Leipzig Corpora Collection.

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_XMB.txt** and contain the first XMBs of the **english.txt** file.

4 Experiments

5 Results

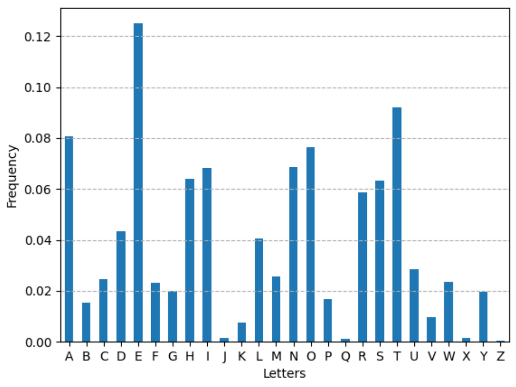

Figure 1: English letter frequency

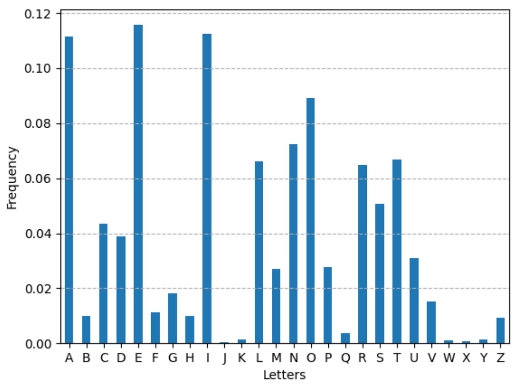

Figure 2: Italian letter frequency

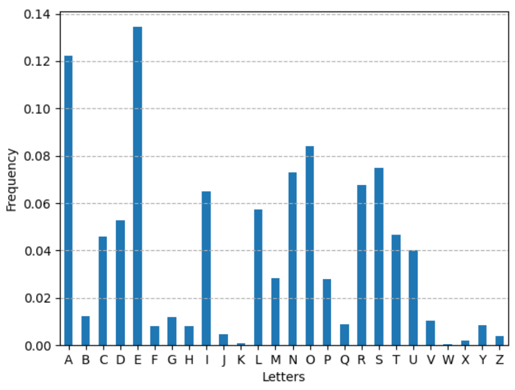

Figure 3: Spanish letter frequency

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 4 –no-combiner
8	-i english.txt -r 4 –no-in-mapper-combiner
9	-i english.txt -r 4 –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

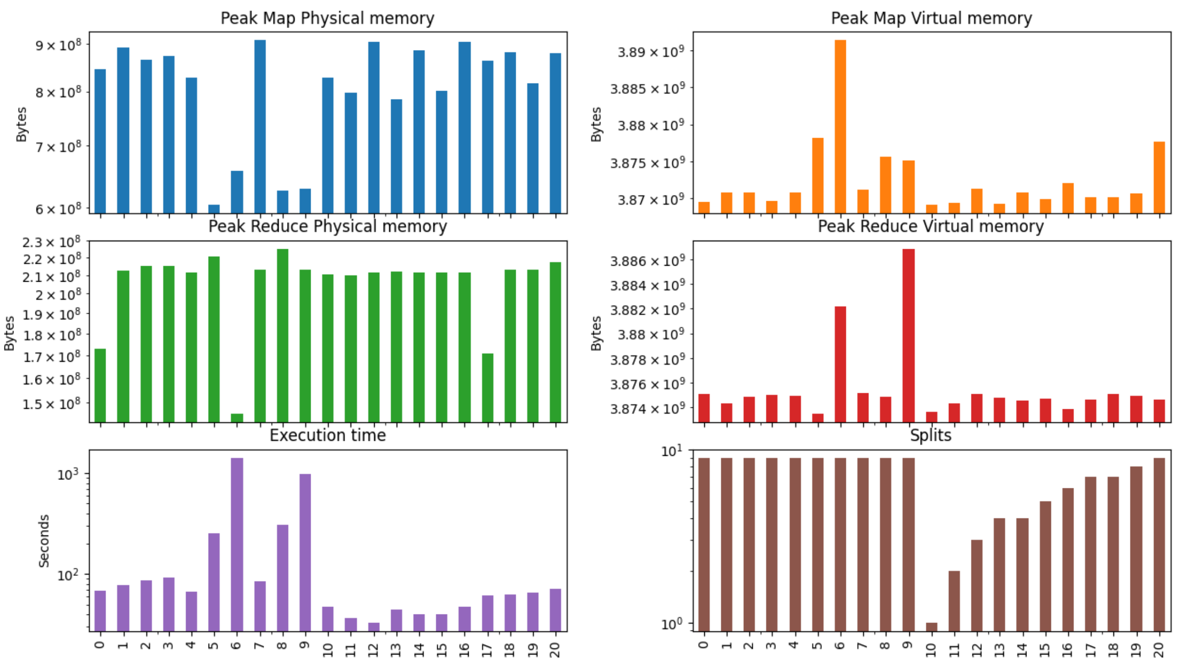

Figure 4: Tests results, the x-axis represents the test ID.

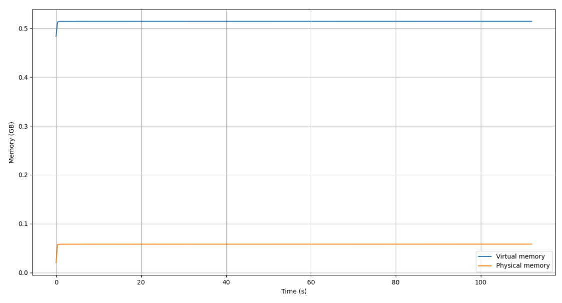

Figure 5: Python baseline memory usage (GB) over time (s)

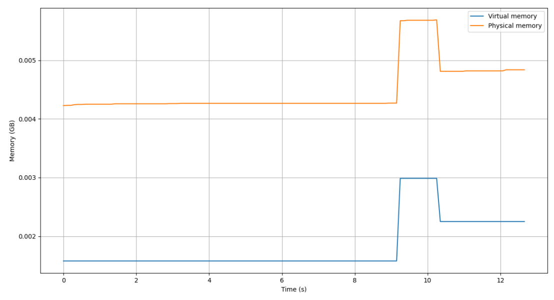

Figure 6: Rust baseline memory usage (GB) over time (s)

6 Conclusions

7 Listings

Listing 1: Mapper with in-mapper combiner

```java
package it.unipi;

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);
        }
        key.set('*');
        value.set(sum);
        context.write(key, value);
    }

    @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

```
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 value = new LongWritable(1);

    @Override
    public void map(Object key, Text value, Context context) throws IOException
        , InterruptedException {
        long sum = 0;
        for (int c : StringUtils.stripAccents(value.toString()).chars().toArray
            ()) {
            int lowerC = Character.toLowerCase(c);
            if (lowerC >= ’a’ && lowerC <= ’z’) {
                this.key.set((char)lowerC);
                context.write(this.key, this.value);
                sum++;
            }
        };
        this.key.set(’*’);
        this.value.set(sum);
        context.write(this.key, this.value);
    }
}
```

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) {
```

```java
                sum += val.get();
            }
            if (key.get() == '*') {
                context.getCounter("LetterFreq", "total").setValue(sum);
            }
            else {
                result.set(sum);
                context.write(key, result);
            }
        }
}
```

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

```java
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

```java
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 int hashCode() {
        return Character.hashCode(c);
    }

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