MapReduce

Sanele Ndlovu-716411
Knowledge Dzumba - 813137
School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg
ELEN4020A:Data Intensive Computing in Data Science

1 Introduction

This report presents results of a programming laboratory work on MapReduce, using MRJob python framework. The purpose of the lab is to explore the use cases of MapReduce as applied to common problems on big data problems.

2 Counting Occurrences of Words

Algorithm 1 computes the occurrences of words within a text file. This is done by mapping the single words individually and counting them, a combiner is used to optimize the algorithm and sum the the word frequencies that have been encountered already. The reducer sums up the total word frequencies after the mapper is done with the text file, because of the combiner the reducer's job steps are reduced. The time taken to execute the program using a mid size file is **0.12503480911254883s**, and the time taken to execute the program using the File2ForLab3.txt is **2.828373432159424s**.

Algorithm 1: Word Counting

Input: text file with words
Output: key = words, value = frequency

```
open textfile and remove all punctuation
for word in textfile to end of file do
    word to lowercase
    return word,1
end
for key,value in word do
    return key,sum(value)
end
return key,value
```

3 Top-K

Algorithm 2 outputs the top-K words of a textfile. The top appearances of the words in a text file are stored in descending order the first being the most appearing word.

Algorithm 2: Naive Top-K algorithm

Input: text file with words
Output: key = words, value = frequency

```
for word in textfile to end of file do
    remove all punctuation
    Counter[word.lower()] += 1
    sort Counter
    return Counter
end
for key,value in Counter.most_common(k) do
    return key,value
end
return key,value
```

Testing the algorithm using my_File.txt for K=10 gives the time **0.13726162910461426s** and for K=20 **0.15708613395690918s** respectively. Testing the algorithm using File2ForLab3.txt for K=10 gives the time **8.13726162910461426s** and for K = 20 the program did no finish running. This is because of the inefficient algorithm used in the implementation. An alternative more efficient algorithm to be used is Fagin's Algorithm. This would be more efficient because not all the text would be searched and summed to give the Top-K.

4 Inverted Index Algorithm

Algorithm 3 computes the inverted index of the texts, not implemented in code

Algorithm 3: Inverted Index

Input: (filename,texts)records
Output: words,filenames

```
for words in text.split() do
    return word,filename
end
for words,filenames in record do
    return word, sort(filenames)
end
```