ELEN4020: Data Intensive Computing Lab 3

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Elias Sepuru 1427726 Boikanyo Radiokana 1386807 Lloyd Patsika 1041888

I. INTRODUCTION

In this report, the MapReduce framework is used to count the occurrence of each word in a text file. On top of finding the occurrence of each word, the top K, where K=10,20, occurring words are distinguished. The framework is also used to find the indices of words in a text file. The time taken to count the occurrences of each word in a text file and the indices the words appear on are recorded. Mrs-MapReduce, a lightweight implementation of MapReduce is used.

II. DESIGN & IMPLEMENTATION

This lab is divided into three sections, the basic WordCount hello world, topK query and the inverted index of text.

A. Hello World: WordCount

The basic WordCount implementation involves obtaining words from a text file which are then assigned to a value, which is the number of times it occurs in the text file. The output text file simply sorts the words in alphabetical order and also shows the value. The role of the map function is to assign a word, the key, to an initial value of 1. In order for the mapping to occur, the program first checks whether the word is not one of the stop words. Stop words are located in a separate file and are not included in any mapping. The reduce function then aims to find identical keys from the mappings and add their values in order to obtain the final count for each word. The implementation of the WordCount algorithm is presented in algorithm 1.

B. TopK Query

The topK query implementation involves finding the top 10 and 20 words that appear the most in a text file. It is achieved through the use of two functions, `SortInDesc` and `TopKQuery`. The `SortInDesc` uses an array to sort the words and frequencies in descending order by continuously sorting every time a word is read from a file. After sorting the words in descending order only the top 10 and 20 entries of the array are written to two different files respectively. The sorting function is called in the `reduce` function so that it continuously sorts the array as a word and its total frequency is generated by the `reduce` function. Algorithm2 illustrates how the two functions were implemented:

input : A text file and the directory of where the output file is going to be saved, all from the terminal
output: Text file containing words with the number of times they occur

```
map(line_num, line_text):
for word in Apostrophe.findall(line_text) : do
 | word←word.strip(string.punctuation).lower()
end
if word.lower() not in StopWords: then
 | if word: then
 |  | yield (word,1)
 | end
end

reduce(word,counts)
yield sum(counts)
```

Algorithm 1: `BasicWordCount(mrs.MapReduce)`: Hello World and Stop Words

C. Inverted Index

The inverted index text involves returning the lines that each word in a text file resides on. The number of lines returned are restricted to 50. The inverted index program is created using Mrs-MapReduce. The following algorithm shown in algorithm 3 is used to perform the inverted index task.

The `map` function works the same way as described in the previous sub-section, the only difference is that it returns the word and its line number instead of the number 1. In the `reduce` function the line numbers corresponding to each word are appended to a single array, the if statements guard to ensure that no line numbers are repeated and to ensure that no more than 50 line numbers are returned.

III. RESULTS

The three programs Simple WordCount, TopK Query and Inverted index are run serially and then later with Mrs-Mapreduce dividing the job into tasks and giving it to slaves. Table I presents the run times for each program for running serially and using Map-Reduce.

Unfortunately the Hadoop setup always gave errors when running the master and slaves, for some odd reason the

```
input : A text file and the directory of where the output
        file is going to be saved, all from the terminal
output: Two text files containing the top 10 and top 20
        words respectively

SortInDesc(word, counts):
words←open("SortedWords.txt","append")
write word and counts to words
words.close()

open SortedWords.txt as textFile:
lines←lines.split() for line in textFile
sort lines in descending order
TopKQuery(lines)
words.close()
num←0

TopKQuery(lines):
top10←open("top10.txt","w")
top20←open("top20.txt","w")
for i in lines do
    if num < 20 then
        if num < 10 then
            top10.write(i[0] +" "+ i[1] + "")
        end
        top20.write(i[0] +" "+ i[1] + "")
        num = num+1
    end
end
top10.close()
top20.close()
```

Algorithm 2: `SortInDesc` and `TopKQuery`

```
input : A text file and the directory of where the output
        file is going to be saved, all from the terminal
output: Text file containing words with indices

map(line_num, line_text):
for word in line_text.split(): do
    word ← word.strip(string.punctuation).lower()
end
if word not in StopWords and not word[0].isdigit()
 then
    yield (word, line_num + 1)
end

reduce(word,line_num)
lineNumbers ← []
for i in line_num do
    if len(lineNumbers) >= 50 then
        break
    end
    if i not in lineNumbers then
        lineNumbers.append(i)
    end
end
yield lineNumbers
```

Algorithm 3: `WordIndex(mrs.MapReduce)`: Inverted text code

program was complaining that the slaves and master are not running the same programm. It gave out the following error : *WARNING: Slave tried to sign in with non matching code*. So the the three programs were never ran using Hadoop, only serially using the basic Mrs-MapReduce framework

IV. CONCLUSION

In this lab report, a simple WordCount, a TopKQuery and inverted index programs are created in python using Mrs-MapReduce. Mrs-MapReduce uses the concept of master and slave to divide the job into tasks that run parallely by the slaves to be collected by the master at the end of the program.

TABLE I
TIME TAKEN TO RUN PROGRAM

Program	File1ForLab3.txt time(s)	File2ForLab3.txt time(s)
Simple WordCount	0.54324	7.10453
TopK Query	13.65718	0.00
Inverted Index	1.104700	5.16014