University of the Witwatersrand
School of Electrical & Information Engineering

ELEN4020A
Data Intensive Computing for Data Science
Laboratory 3 Report

Milliscent Mufunda - 1473979
Keanu Naidoo - 1422973
Matthew Woohead - 1385565

Due Date: 16 April 2019

***Abstract*—This report discusses the laboratory work done on a MapReduce framework, Mrs-MapReduce, to create and test three algorithms involving the analysis of three different sized sample texts. These include a Word Count Algorithm, a Top-K query Algorithm and an Inverted Index of Text Algorithm. These algorithms were tested with samples texts of different sizes, it was found that these algorithms were successfully implemented in the MapReduce framework.**

I. Introduction

MapReduce is a technique used for processing and generating big data sets. It is a framework that is used to split problems and solve them using a parallel distributed algorithm on a cluster. Only certain problems, that can be decomposed, can be solved using MapReduce techniques. A MapReduce program is composed of a map procedure, which is responsible for filtering and sorting, and a reduce method, which is responsible for performing a summary operation. This report discusses the development of three algorithms for MapReduce Frameworks. Included are the lab requirements and the pseudo code for the algorithms. These can be found in sections II and III respectively.

II. Laboratory Requirements

The laboratory has the following requirements; it should be conducted using the Linux programming environment and with the use of one or more of specified editors. All laboratory documentation is to be done using Latex. The use of GIT and GitHub is also required. The problem to be addressed is the development of three separate procedures to test and demonstrate the features of the MapReduce framework Mrs-MapReduce. These procedures require input text sizes, which vary from small, medium and large texts segments, these can be seen as attached documents. These procedures are the:

- Basic Word Count Algorithm
- Top-K Query Algorithm
- Inverted Index of Text Algorithm

To meet the requirements, all the coding is done using Ubuntu 16.04 and the python coding language with the gedit editor. The laboratory documentation is done using Overleaf. A GIT repository is also used to track the history of the laboratory work done.

III. Explanation of Algorithms

All of the algorithms build on each other, the first algorithm makes use of the map reduction framework to create a list of words and the frequency of these words in a sample text. The second algorithm will make use of the data from the first algorithm and indicate the top K recurring words, where K is equal to 10 and then 20. The third algorithm lists the lines in which these words occur; however, only the first 50 lines of the sample text are analyzed.

A. Word Count Algorithm

This is a basic "Hello World" algorithm for MapReduce frameworks, and this achieved by making use of Mrs MapReduces built in libraries and the map and reduce functions. The word count is done by first splitting the text up and then removing all punctuation marks and converting all letters in the text to lower case. The rest of the algorithm simply runs through the text and counts the occurrences of each word for all the parts of the texts that were split, this is followed by a reduce function which merges the accumulated words and their frequencies. If the same words appeared in separate split text samples, these will be simply added together and only occur once in the entire word listing. The psuedo code for the basic algorithm can be seen in Algorithm 1.

```
Input: line number, line text
Output: word, word count
initialization: map;
for first word to last word do
    word ← $word.removePunctuation$
    word ← $word.lowerCase$
    return word ;
end
initialization: reduce;
word count ← $wordcount + wordcount$
```
Algorithm 1: Word Count Algorithm

B. Top-K Query Algorithm

This algorithm is an extension of the first algorithm. The Top-K Query Algorithm takes the generated map from the first algorithm and finds the most frequent words. This is achieved by evaluating the frequency of word entries in the map to find the top 10 and 20 most reoccurring words. This can be seen below in Algorithm 2.

C. Inverted Index of Text Algorithm

This algorithm is very similar to the Word Count Algorithm, the only addition is recording the line which a word occurred. This simple adjustment is made in the map function of the Word Count Algorithm, which now also yields the line number which the word appeared in. This can be seen in below in Algorithm 3.

```
Input: word, word count
Output: word, word count
initialization: Array A (10);
initialization: Array B (20);
sort.descending(word count)
for $i$ = 0 to A.size do
    A$_i \leftarrow word_i$;
end
for $i$ = 0 to B.size do
    B$_i \leftarrow word_i$;
end
```

Algorithm 2: Top-K Query Algorithm

```
Input: line number, line text
Output: word, word count, line number
initialization: map;
for first word to last word do
    word $\leftarrow word.removePunctuation$
    word $\leftarrow word.lowerCase$
    return word, line number;
end
initialization: reduce;
word count $\leftarrow wordcount + wordcount$
```

Algorithm 3: Inverted Index of Text Algorithm

IV. Analysis of the Algorithms

The three aforementioned algorithms were tested using three sample texts of different sizes, this was provided by File2ForLab3.txt. The sample text was used for all three cases; however, for the small text sample size only line one to one thousand was used. For the medium text size, line one to twelve thousand was used and the large sample text composed of the entire file, which had 24 693 lines. The efficiency of the algorithms is determined by analyzing the running time. As can be seen, an exponential runtime can be seen with a larger sample text size for each algorithm. The running times are shown in table 1.

TABLE I
Running Time for Algorithms

Algorithm	Short Sample Text	Medium Sample Text	Large Sample Text
Word Count	0.11 s	0.87 s	1.98 s
Top-K Query	0.17 s	1.09 s	2.28 s
Inverted Index of Text	0.64 s	2.41 s	3.68 s

V. Recommendations for Future Work

The analysis of the performance of the Mrs-MapReduce framework could be improved by comparing the Word Count, Top-K Query and Inverted Index of Text algorithms with corresponding algorithms that do not make use of the Mrs-MapReduce framework. The analysis done could also be improved by testing with larger sample texts.

VI. Conclusion

Therefore, through the use of the Mrs-MapReduce map reduction framework, the three algorithms were successfully implemented and tested. The algorithms were used to test small, medium and large text sources; which all proved to be successful. Pseudo code has been included for all the algorithms created. Further discussion were made on the effectiveness of these algorithms.