**COSC 5340 - 04 Data Science and Big Data Analysis**

Spring 2016

Instructor - Dr. Sujing Wang

**Project Report**

**Searching and Counting of Words and/or Pair of Words**

Due on May 10, 2016

**Submitted by**

Bhumit Patel (L20382154)

Hardik Patel (L20366761)

# Introduction

The main purpose of our project is to compare execution time of simple java program and map reduce program. For that, we have used word count program which reads input file and counts the occurrence of each word from the input dataset. Program counts each word from dataset as well as count of pair of words from dataset. It also displays high frequency word as output.

# Project Overview

The main purpose of project is to searching and counting the words from the input dataset. We are going to use dataset named “The Island of doctor moreau” for our program. This dataset is available from “<http://docslide.us/download/link/herbertwellstheislandofdoctormoreautxt>” this link.

We will use mapper and reducer to implement the word count program. The word count program is used to search as well as count the specific word and/or pair of words and also find out the high frequency word from input dataset.

It basically MapReduce program demonstrates a map-side search. From the user side, the argument passes from the command line for a word and/or pair of words search to the mapper. The mapper will use conditional statement for compare the search string to key value. When the word found, it will send to the reducer for aggregation. This results show instances of that specific word or pair of word being counted with all other list of words along with count number.

# Implementation

We are using MapReduce technique to perform our searching, sorting and counting of words and/or pair of words.

We are planning to use Hadoop for implementation. We installed cluster machine in oracle virtual box and implement the project in it.

We developed code in eclipse using java with the map() and reduce() function and ran this code in cluster for desired output.

User need to pass the argument as a word or a pair of words to search in given dataset. After searching, the result will be sorted with occurrence. At last the high frequency word will be displayed with the number of count.

The general use of map() function is to process a key/value pair to generate intermediate key/value pairs. First of all, the mapper will tokenize line into word and then store into key value pair. It will search for occurrence of pair of words and also store into key value pair.

The general use of reduce() function is to merge all intermediate values associated with the same key. The reducer class will merge all intermediate values associated with same key and it will write count for word and pair of word into output file. On startup of reducer class it will initialize word and frequency of that word. During reduce method, it will check for highest frequency word. On cleanup of reducer class, it will write highest frequency word into output file.

Our project implementation tools are following:

* Microsoft Windows 10
* Oracle Virtual Box
* Cloudera 5.7.0
* Eclipse IDE
* Microsoft Excel 2013

To execute and perform the project, we need the two programs which are following:

# WordCount.java

*import java.io.IOException;*

*import java.nio.ByteBuffer;*

*import java.util.\*;*

*import java.util.regex.Matcher;*

*import java.util.regex.Pattern;*

*import org.apache.hadoop.fs.Path;*

*import org.apache.hadoop.conf.\*;*

*import org.apache.hadoop.io.\*;*

*import org.apache.hadoop.mapreduce.\*;*

*import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;*

*import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;*

*import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;*

*import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;*

*import org.apache.hadoop.util.Tool;*

*import org.apache.hadoop.util.ToolRunner;*

*public class WordCount extends Configured implements Tool{*

*public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {*

*private final static IntWritable one = new IntWritable(1);*

*private Text word = new Text();*

*public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {*

*String line = value.toString();*

*StringTokenizer tokenizer = new StringTokenizer(line);*

*while (tokenizer.hasMoreTokens()) {*

*word.set(tokenizer.nextToken());*

*context.write(word, one);*

*}*

*Configuration conf = context.getConfiguration();*

*String StringToFind=conf.get("StringToFind");*

*Pattern p = Pattern.compile(StringToFind);*

*Matcher m = p.matcher(line);*

*while (m.find()) {*

*word.set(StringToFind);*

*context.write(word,one);*

*}*

*}*

*}*

*public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {*

*private Text tmpWord = new Text();*

*private int tmpFrequency = 0;*

*public void reduce(Text key, Iterable<IntWritable> values, Context context)*

*throws IOException, InterruptedException {*

*int sum = 0;*

*for (IntWritable val : values) {*

*sum += val.get();*

*}*

*context.write(key, new IntWritable(sum));*

*//Check for high frequency word*

*if(tmpFrequency < sum) {*

*tmpFrequency = sum;*

*tmpWord.set(key);*

*}*

*}*

*@Override*

*public void cleanup(Context context) throws IOException, InterruptedException {*

*// write the word with the highest frequency*

*String maxWord = tmpWord.toString();*

*Text tempText = new Text(*

*"------- High Frequency Word-----------\n"+ maxWord + ":");*

*context.write(tempText, new IntWritable(tmpFrequency));*

*}*

*}*

*@Override*

*public int run(String[] args) throws Exception {*

*Configuration conf = this.getConf();*

*if(args.length != 3){*

*System.out.println("usage: InputFilePath OutputFilePath StringToFind");*

*System.exit(0);*

*}*

*conf.set("StringToFind",args[2]);*

*@SuppressWarnings("deprecation")*

*Job job = new Job(conf, "wordcount");*

*job.setJarByClass(WordCount.class);*

*job.setOutputKeyClass(Text.class);*

*job.setOutputValueClass(IntWritable.class);*

*job.setMapperClass(Map.class);*

*job.setReducerClass(Reduce.class);*

*job.setInputFormatClass(TextInputFormat.class);*

*job.setOutputFormatClass(TextOutputFormat.class);*

*// job.setSortComparatorClass(IntComparator.class);*

*job.setNumReduceTasks(1);*

*FileInputFormat.addInputPath(job, new Path(args[0]));*

*FileOutputFormat.setOutputPath(job, new Path(args[1]));*

*return job.waitForCompletion(true) ? 0 : 1;*

*}*

*public static void main(String[] args) throws Exception {*

*int res = ToolRunner.run(new WordCount(), args);*

*System.exit(res);*

*}*

*}*

# SimpleWordCount.java

*import java.io.\*;*

*import java.util.Map;*

*import java.util.Set;*

*import java.util.StringTokenizer;*

*import java.util.TreeMap;*

*import java.util.regex.Matcher;*

*import java.util.regex.Pattern;*

*public class WordCountSimple {*

*// The name of the file to open.*

*String inputFile = "Datafile2\_128MB.txt";*

*String outputFile = "output.txt";*

*BufferedReader bufferedReader;*

*BufferedWriter bufferedWriter;*

*Map<String, Integer> wordCounts = new TreeMap<String, Integer>();*

*public void countWord(String stringToFind) {*

*// This will reference one line at a time*

*String line = null;*

*try {*

*bufferedReader = new BufferedReader(new FileReader(inputFile));*

*bufferedWriter = new BufferedWriter(new FileWriter(outputFile));*

*while ((line = bufferedReader.readLine()) != null) {*

*// count occurrences*

*StringTokenizer tokenizer = new StringTokenizer(line);*

*while (tokenizer.hasMoreTokens()) {*

*String word = tokenizer.nextToken();*

*if (!wordCounts.containsKey(word)) {*

*wordCounts.put(word, 1);*

*} else {*

*wordCounts.put(word,*

*wordCounts.get(word) + 1);*

*}*

*}*

*Pattern p = Pattern.compile(stringToFind);*

*Matcher m = p.matcher(line);*

*while (m.find()) {*

*if (!wordCounts.containsKey(stringToFind)) {*

*wordCounts.put(stringToFind, 1);*

*} else {*

*wordCounts.put(stringToFind,*

*wordCounts.get(stringToFind) + 1);*

*}*

*}*

*}*

*Set<Map.Entry<String, Integer>> entrySet = wordCounts.entrySet();*

*String maxWord = "";*

*int maxCount = 0;*

*bufferedWriter.write("Words" + "\t\t" + "# of Occurances\n");*

*for (Map.Entry<String, Integer> entry : entrySet) {*

*bufferedWriter.write(entry.getKey() + "\t\t" + entry.getValue()*

*+ "\n");*

*if (entry.getValue() > maxCount) {*

*maxWord = entry.getKey();*

*maxCount = entry.getValue();*

*}*

*}*

*bufferedWriter.write("------- High Frequency Word-----------\n ");*

*bufferedWriter.write(maxWord + ":" + maxCount + "\n");*

*// Always close files.*

*bufferedWriter.close();*

*bufferedReader.close();*

*} catch (FileNotFoundException ex) {*

*System.out.println("Unable to open file");*

*} catch (IOException ex) {*

*System.out.println("Error reading file");*

*}*

*}*

*public static void main(String[] args) {*

*long startTime = System.currentTimeMillis();*

*/\*if (args.length < 1) {*

*System.out.println("usage: WordCountSimple StringToFind");*

*System.exit(0);*

*}\*/*

*WordCountSimple wc = new WordCountSimple();*

*wc.countWord("The People");*

*long stopTime = System.currentTimeMillis();*

*long elapsedTime = stopTime - startTime;*

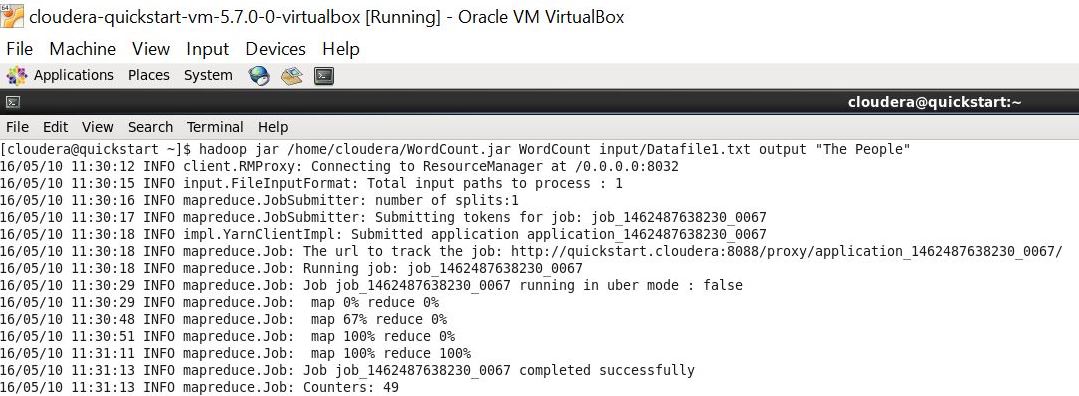
*System.out.println(elapsedTime);*

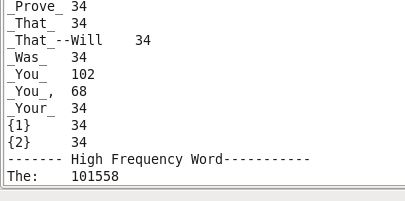
*}*

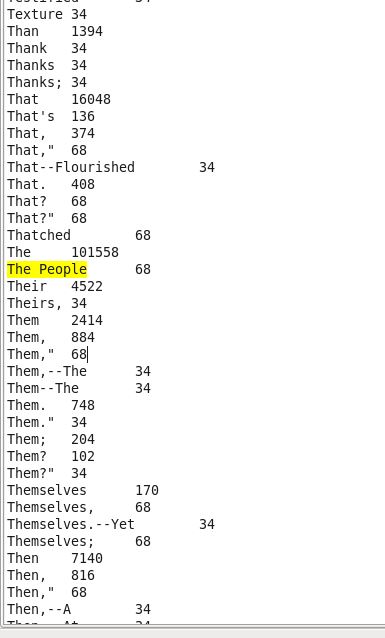
*}*

# Outputs

The desired output for this project is to show the word or a pair of word which was inserted from command line by the user. Also show the highest frequent word in output from the input dataset. In here, user passed the argument “The People” as an input pair of word to be search. So, the desired output of our project is shown in following screenshots.







# Analysis and Conclusion

We developed two programs for word count. One is a simple java program and another is used with mapper and reducer.

First, we used 0.242MB (242KB) size dataset. 0.711 second time taken by the simple word count program. The time consumed for word count program with using mapper and reducer is 86 seconds.

Secondly we used approximately 8MB size of dataset. It takes 2.18 second to run simple word count program. The time consumed for word count program with using mapper and reducer is 66 seconds.

So, we can say that, the time consumption for simple java program is increased when the file size increases, and time reduce for word count program with mapper and reducer.

# Future Works

We concluded that while we are increasing file size, it is giving good result for Hadoop Map reduce. For the future work, we can add input file of a list of pair of words and search for that pair of word into dataset.

# References

* <http://www.worldcat.org/title/island-of-doctor-moreau/oclc/1811077>
* <http://www.gutenberg.org/ebooks/19726>
* <https://en.wikipedia.org/wiki/MapReduce>
* <http://www.dummies.com/how-to/content/hadoop-mapreduce-for-big-data.html>
* <http://www.informit.com/articles/article.aspx?p=2008905>
* https://github.com/himank/K-Means
* <http://www.ijircce.com/upload/2015/october/30_MapReduce.pdf>