CIS 4930 Introduction to Hadoop and Big Data
Assignment 1
Julie Reyes U76631122

1 Introduction

In this report we investigate two projects that delve into to the basics of how to implement MapReduce jobs on a hadoop file system.The first project consists of a MapReduce job that reads input text and determines the average length of words within the text that have the same starting character. The second project in this report also performs a MapReduce job that reads in a weblog file and extracts the IP address, then counts the total number of hits on the same IP addresses.

2 Average Word Length

2.1 Driver description

The driver for the Average Word Length project is named *AvgWordLength.java* and is responsible for configuring and implementing the MapReduce job. This driver specifies which jar file contains the driver, mapper, and reducer and is named *AvgWordLength.class*. Then, the driver sets the job name to "Average word length" , this name is used in reports and logs when the job is run. Next, the driver defines the input and output paths formats based on the command line arguments. The input is set from the command line args[0] and the output is set from command line at args[1]. In the driver code, the mapper is set to use *LetterMapper.class* and the reducer is set to use *AverageReducer.class.* Lastly, the driver sets the output key and value pair configuration, particularly in this project the key is of the *Text.class* and the value is of the *IntWritable.class.*

2.2 Mapper description

The mapper for Average Word length class requires four class parameters that are defined as *LongWritable*, *Text*, *Text*, and *IntWritable*. The map function is defined by a subclass mapper function *LetterMapper,* which is overridden by the map method. The map function takes the parameters *LongWritable* as the key, *Text* as the value, and *Context* to emit the key and value pair from the map method. The map method operates by converting a line of the input file in the form of a *Text* object to a *String* object. This operation is done by the *toString()* function. Next, a for loop splits the string into separate words. Inside the loop the *Context* object method *write()* is used to emit the first letter of each word using the *substring()* method. This *Context* object is the intermediate key. The *write()* method also emits the value as the length of the word using the *length()* method. The output of the map function is of the type *Text* object for the key and of *IntWritable* type for the word length.

2.3 Reducer description

The reducer is defined by an *AverageReducer* subclass that is overridden by the reduce method. The *AverageReducer* subclass takes in the parameters *Text*, *IntWritable*, *Text*, *IntWritable*. These parameters denote the intermediate input key and value pairs from the Mapper and the output values from the Reducer method. The reduce method that overrides the *AverageReducer* takes in the three parameters including the *Text* for the key, *Iterable<IntWritable>* for the values, and a *Context* object. In the reduce method two variables, *wordCount* and *count*, are initialized to zero. Next, a for loop uses the *get()* function so the values with the same key passed by the mapper are added to the *wordCount* variable. This in effect sums the lengths of words with the same key. Inside the loop the *count* variable maintains a total count of the amount of values of a key type so that once the for loop has ended the total addition of the lengths is divided by the number of words with the same key. Therefore, the average length of words with the same starting character is obtained. Next, the *Context* object method *write()* is called to emit the final key value pair. The output key is of type *Text* and the output value is of type *DoubleWriteable*.

2.4 Data flow

Initially, the Average Word Length job requests to perform a MapReduce job on data found in the Hadoop cluster. The diver code is run on the client side, then submits a job to the cluster. Once the job is submitted to the cluster, the *shakespeare* HDFS file, which is located on blocks within the DataNodes, is read by the record reader to be used by the mapper. Next, the data is sent to the mapper to be manipulated based on the java code. In the mapper code the first letter of each word is made into a key and the length into the value. The data emitted from the map functions are shuffled and sorted , then sent to the reducer tasks. The mapper tasks and reducer tasks are both performed on the dataNodes and are maintained by either the job tracker in MapReduce v1 or by the resource manager in v2. The final output is saved as a file by the name specified in the command line of the job and is saved on the hadoop cluster.

2.5 Compile Procedure

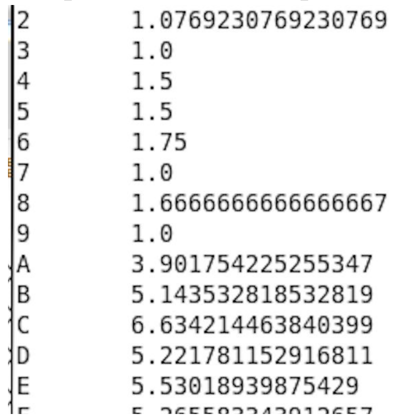
In the command line the following was input:

```
$ hadoop jar avgwordlength.jar stubs.AvgWordLength shakespeare wordlengths
```

This command line is translated as the *avgwordlength.jar* being the jar file, *stubs.AvgWordLength* is the directory of the driver code, *shakespeare* is the file the job is acting on, and *wordlengths* is the name of the output file on the Hadoop cluster.

2.6 Testing and Results

The *Average Word Length* job was initially tested on a small text file that would produce easy to confirm outputs. Once the MapReduce job produced the desired results, the job was submitted on the *shakespeare* file on the hadoop cluster. The picture below shows a sample of the final output.



```
2       1.0769230769230769
3       1.0
4       1.5
5       1.5
6       1.75
7       1.0
8       1.6666666666666667
9       1.0
A       3.901754225255347
B       5.143532818532819
C       6.634214463840399
D       5.221781152916811
E       5.53018939875429
F       5.265583243912657
```

3 Log File Analysis

3.1 Driver Description

The driver for this project is named *ProcessLogs.java*. This driver is responsible for configuring and implementing the MapReduce job named *Process Logs*. This driver specifies the jar file that contains the driver, mapper, and reducer as *ProcessLogs.class.* The driver then specifies the input directory from which to read and an output directory to write to based on the command line inputs. The driver gives the job object the information for the mapper named *LogFileMapper.class* and the reducer as *SumReducer.class.* The reducer for this project is reused from the *WordCount* example given in Lab 2 of this course. Next, the driver defines the intermediate key and values from the Mapper, these are of type *Text* and *IntWritable* for this project.

2.2 Mapper description

The mapper class *LogFileMapper* extends the Mapper base class and accepts four parameters that include *LongWritable, Text, Text,* and *IntWritable.* The map method overrides the mapper and takes in three parameters *LongWritable*, *Text*, and *Context.* In the map method, the input line is converted from type *Text* to a string via the *toString()* method. Next, the function *indexOf()* finds the index of the first instance of a space character and saves that index of the string in an integer variable. Hence, this integer variable holds the length of the IP addresses based on the format of the *weblogs* file. Lastly, the *write()* method from the *Context* object emits the substring holding the IP address key as type *Text* and emits the value *1* of type *IntWritable*.

2.3 Reducer description

The reducer class for this project is the same as the reducer used in the Word Count. Example, *SumReducer*. This reducer has four parameter that include two intermediate types *Text* and *IntWritable*, and two output types *Text* and *IntWritable.* Next the reduce method overrides the reducer and accepts four parameters. The reduce method in this example takes in the parameters that include a *Text* key, an *Iterable<IntWritable>* set of objects that are iterable, and a *Context* object. Inside the reduce method a for-each loop uses the *get()* method to add the set of values together producing the sum. Lastly, the *Context* object emits the final key value pairs, which include the IP address and the number of times it exits in the input file.

2.4 Data flow Description

The *weblog* file was used to implement this job and was located on the Hadoop cluster. Initially, the driver code executes on the local computer and then submits the job to the cluster. The input file is located on many different blocks inside dataNodes on the cluster. The data in the blocks is acted upon by the record reader first. Next, the data is manipulated by the map tasks. The map tasks are performed in the dataNodes, and ideally are tasked to the same dataNode where the data is located to increase efficiency. Once the map tasks are completed, the shuffle and sort merges the intermediate data from all the mappers. The data then is sent to the reduce task, which is the final step the data goes through before output. Once, the job is completed it sends the data to the output directory provided by the location specified in the driver code.

2.5 Compile Procedure

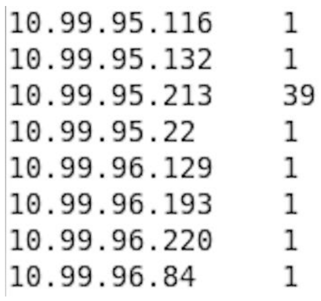
In the command line the following was input:

$ hadoop jar log_file_analysis.jar stub.ProcessLogs weblog webIP

In the example above, the *log_file_analysis.jar* is the jar file, *stub.ProcessLogs* is the directory of the driver code, *weblog* is the file the job is acting on, and *webIP* is the name chosen for the output file.

2.6 Program test procedure

The program is initially tested on a smaller file *testlog* that was placed on the cluster for testing purposes. Once the job produced the intended results, the final job was used to extract the IP addresses from the *weblog* file on the Hadoop cluster. Below is a picture of the results produced by the job.



```
10.99.95.116    1
10.99.95.132    1
10.99.95.213    39
10.99.95.22     1
10.99.96.129    1
10.99.96.193    1
10.99.96.220    1
10.99.96.84     1
```