CIS 4930 Introduction to Hadoop and Big Data
Lab Homework 7

Julie Reyes

1 InvertedIndexInput

1.1 Project Introduction

The purpose of this project is to perform an inverted index job. Specifically, the job for this project is performed on the invertIndexInput.tgz file, which is decompressed and submitted to the Hadoop distributed file system. This project will output an index list for all the words in the text. For example, for an instance of a word found in the text, the index will contain a list of all the locations the word is found.

1.2 Driver Description

The driver for this project is named InvertedIndex, and is configured to implement Tool. The driver defines the input and output paths to be defined by the command line arguments. The driver map class is set to be the IndexMapper.class and the Reducer is set to IndexReducer.class. The driver sets the map output key and value class to be of type Text. The input format class for this project is set to KeyValueTextInputFormat.class. The driver sets the job name as “Inverted Index”. The driver sets the driver class to be InvertedIndex.class.

1.3 Description on Mapper

The Mapper for this project is named IndexMapper.java. This Mapper extends the Mapper class. This Mapper takes in four parameters, all of which are of type Text. The Map function overrides the Mapper and takes in three parameters including the Text key, Text value, and Context context. The map function utilizes FileSpit object which is initialized by the context.getInputsplit() that is cast to a FileSplit object. Next, the file path is extracted by the function getPath(). The path of the file is saved in a variable named fileName, which is modified to include an “@” and the key as the whole string. Therefore, the final fileName string is represented as filename@key. Before appending the key to the fileName string it is converted to type string with the toString() function. Next, the variable line is initialized as the value converted to type string with toString() method and made lowercase with the toLowerCase() method. A for loop is used to iterate through the input line and split it into words, which are then emitted by the context.write() method. The key emitted from the Mapper is the word extracted from the input line and the value emitted is the filename and the line number the word is found.

1.4 Description on Reducer

The Reducer class for this project is called IndexReducer, which extends the Reducer class that takes in four parameters of which are all of type Text. The reduce method overrides the Reducer, and takes in three parameters. The parameters the reduce method accepts includes, the Text key, Iterable<Text> values, and Context context. The reduce method then uses the StringBuilder class to create a mutable string named valueList. The valueList string contains an index of all the locations of the key word. A for loop takes in the iterable values and appends the values to the valueList string with a comma separating the filenames. The final output that is emitted from the Reducer is the word received from the Mapper as the key and the value is the index list that includes all the locations of the word by file name and line number it is found at separated by commas.

1.5 Data flow description

Data flow description. The file that the job for this project is performed on is named invetedIndexInput. This file is decompressed and placed on the Hadoop distributed file system. The driver configures the job then submits it to the Hadoop Cluster. Once the job is submitted to the Cluster, the tasks are submitted to the data nodes. The driver code determines the format that the data is sent to the Mappers. In this project the input format type was of the type KeyValueTextInputFormat.class, which extends the TextInputFormat. Therefore, the record reader splits the data into input splits of type FileSplit. Next, the data goes to the Mapper from the record reader. In the Mapper the data is broken down by input splits into a keyword value pair that is emitted to the Reducer. In the Reducer the data is modified to emit the keyword from the file and all the index values associated with the word in the form of a list. The final output of the Reducer is saved to the output file specified in the command line.

1.6 Test data and results

Test data and results: The job was tested in the eclipse environment with the Run Configuration option. The test run was saved locally and verified to ensure proper output. Once the job output was correct on eclipse test run, the job was submitted to the Hadoop Cluster. The following is an example of the final output:

```
accessible     cymbeline@2332,
accidence      merrywivesofwindsor@3016,
```

2 Word Co-Concurrency

2.1 Project Introduction

The purpose of this project is to calculate the number of times words appear next to each other. This may be of importance when determining connections, especially in the field of data mining. For this project the shakespeare file on the Hadoop Cluster from previous labs was used to verify the efficacy of the MapReduce job for this project.

2.2 Description on driver

The driver for the job in this project is named WordCo.java. In this driver the class WordCo extends Configured and implements Tool. First, in the driver program, the command line is arguments are checked to determine if the proper command has been input. Next, the job is initialized by the Job(getConfig()) method. The jar for this project is set by the .setJarByClass() method as WordCo.class. The job name is specified as “Word Co-Occrence”. The file path for the input is specified by the command line arguement at args[0], and the file output path is specified by the command line argument at args[1]. The Mapper class is set to WordCoMapper.class and the Reducer is set to SumReducer.class. The output key for this project is of type Text and the output value class type is set to IntWritable.class. Lastly, the driver set the job to wait until complete.

2.3 Description on Mapper

The Mapper for this project is defined as WordCoMapper. The Mapper class is extended by WordCoMapper. The Mapper takes in four parameters of type LongWritable, Text, Text, Text, and IntWritable. The subclass method map() overrides the Mapper function. The map() method takes in three parameters that include the LongWritable key, Text value, and Context context. The input value of type Text is converted to a lower case string with the methods toString() and toLowerCase(). Next, a temporary String variable temp is initialized to null. Since, Mapper is looking for pairs of words that are next to each other the boolean variable firstWords is initialized to true. This variable is need to ensure that the first word is selected before finding the word that is next to it. Next, a for loop iterates through the line input and breaks it up by pairs words. If the boolean firstWord is false then the String variable temp is initialized as the first word in the pair. The for loop iterates through values to find all the words that are next to each other. The pairs of words are emitted by the the context variable with the write() method as the key output in the format (word one, word two) of type Text. The value emitted from the Mapper is of type IntWritable and is just the number one as each unique pair will be counted in the Reducer.

2.4 Description on Reducer

The Reducer for this project is named SumReducer. This Reducer is the same Reducer used in the previous Labs. The SumReducer extends the Reducer class and accepts four parameters including Text, IntWritable, Text, and IntWritable. The reduce() method overrides and takes in the parameters Text key, Iterable<IntWritable> values, and Context context. Next, a variable named wordCount is initialized to zero. This variable is used to keep count of the pairs that were the same. To count all similar pairs a for loop iterates through the values and uses the get() method to access them. The Reducer then emits the final key value pairs, which includes the unique pair of consecutive words and the value is the number of times that pair was found on the file.

2.5 Data flow description

Initially the data for this project is found on the Hadoop Cluster. When the job is submitted the job configured on the local machine. Next, the job is submitted to the Cluster. Once the job is submitted, the data input splits are sent to the Mapper. The Mapper takes the data provided by the record reader and forms key value pairs. For this project the pairs are the keys and the value is just one. The intermediate data is shuffled and sorted. Once, last Mapper is complete and the shuffle and sort phase is complete the Reducers begin the reduce tasks on the intermediate data. Lastly, once the all similar pairs of co-current words are counted, the reduce method outputs the pair as the key and the total number of times it was encountered as the value. The final data output is maintained on the Hadoop Cluster.

2.6 Test data and results

The final job for this project was tested within eclipse and on the Hadoop Cluster. The following is an expert of the final output from the job submitted.

```
youth,hath      2
youth,have      1
youth,he 5
youth,henry     1
youth,here      1
youth,how       1
```