Report on Map-Reduce program implementing Inverted Index

• **Project Introduction**: Inverted-index map reduce program involves indexing each and every word detailing the locations where the word appears. The input file for this java project is a directory of files of Shakespeare's play with line number and a value, with a separator, which in this case is a tab character. Given a body of text in the form already defined, the indexer should produce an index of all the words in the text. For each word, the index should have a list of all the locations where the word appears.

• **Driver Description**: The driver class InvertedIndex comprises of the main method where job configuration has been defined. The main method in this driver class instantiates a job object which is configured by calling the *setJarByClass* method and specifying the jar file containing driver, mapper and reducer and reducer. The *setJobName* method is called to set the name for the job which is to appear in reports and logs. The *setInputPaths* and *setOutputPaths* is called on job object to set the input and output paths using command-line arguments. The *setMapperClass* and *setReducerClass* is called to specify the mapper and reducer classes. The input and output format of the key-value pair of is required to be specified well before running the Map Reduce program. In this case output format for the key of both mapper and reducer is same i.e. Text format. And the input format for value in mapper and reducer is same as well i.e., Text Format. This format can be obtained directly using the *KeyValueInputFormat* class provided in the Hadoop API.

• **Mapper Description:** The mapper class i.e., *IndexMapper*, involves the use of *getInputSplit* method using the a context object, and thus obtaining the pathname of the each file. The mapper class has a map method which when called takes the line field as a value parameter from the (k,v) pair and splits in phrases or fields separated by spaces. For each of the word in the line, the context object is used to call the write method to write the mapper output i.e., key as word itself and value as the filename@index. The input key value pair is availed using *KeyValueInputFormat* class provided in the Hadoop API.

• **Reducer Description**: Upon the intermediate shuffling and sorting, all the all the values corresponding to the same key is consolidated and forwarded to reducer function. In this example, the reducer class i.e., *IndexReducer* has input (k,v) pair with key as word in Text Format and value as Iterable elements of type Text representing filename and the line number the word is located at. The output (k,v) pair contains key as words in Text format and value as all the comma-separated indexes in *Text* format, and is emitted by calling write method on context object.

• **DataFlow Description**: First off, the data to be processed is stored in Hadoop File System on the data node. The map task is used to process the data locally on each block where it is stored. The HDFS input file is retrieved in the form of k,v pair with key as first field before separator in a line and values as field after the separator, because of the use of *KeyValueInputFormat* class provide in the Hadoop API. This (k,v) pair is mapped using the defined map function, which in this case is emitting mapper output key-value pair. Now further, the reducer plays its role by processing the intermediate consolidated data using the reduce function i.e. to sum up all the indexes separated by commas related to a particular word emitting the output key-value pair with key as the word and value as comma-separated indexes.

This output (k,v) pair is stored in already defined in defined HDFS directory which can further be accessed, displayed and modified using command-line script.

• **Test Data and Results**: The program is run and tested using Linux command-line scripts. It includes copying the data of local disk to hadoop file system using -cat while defining the directories where it is copying from and storing to. Further the driver, mapper and reducer classes are required to be compiled and exported as a .jar file. The next command runs the map reduce program assigning the tasks and processing the data locally at the blocks. Further the output file invertedIndexOutput/part-r-0000 of the MapReduce program can be accessed or displayed using script commands.

Report on Map-Reduce Program Calculating a Co-Occurrence of a word

• **Project Introduction**: This project involves calculating co-occurrence of all each of the pair of the words which appear next to each other. Each pair of the words which appear consecutively are taken and their count is summed up upon further re-occurrence. The output is expected to be a list with all the consecutive word pairs and their co-occurrence sum.

• **Driver Description**: The driver class i.e., *WordCo* comprises of the main method where job configuration has been defined. The main method in this driver class instantiates a job object which is configured by calling the *setJarByClass* method and specifying the jar file containing driver, mapper and reducer and reducer. The *setJobName* method is called to set the name for the job which is to appear in reports and logs. The *setInputPaths* and *setOutputPaths* is called on job object to set the input and output paths using command-line arguments. The *setMapperClass* and *setReducerClass* is called to specify the mapper and reducer classes. The input and output format of the key-value pair of is required to be specified well before running the Map Reduce program. In this case input format for the key of the mapper is *LongWritable* and reducer is in Text format. And the output format for value in mapper and reducer is in *IntWritable* format.

• **Mapper Description**: The mapper class i.e., *WordCoMapper* has a map function, which when called maps, the input k,v pair with key as offset where the line starts and value as the line itself, to an output k,v pair with key as word pairs and value as 1. The process is done by splitting the input value in fields and retrieving them in the form of pairs from string array using *StringBuilder* Class.

• **Reducer Description**: Upon the intermediate shuffling and sorting, all the all the values corresponding to the same key is consolidated and forwarded to reducer function. In this example, the reducer class i.e., *WordCoReducer* has input (k,v) pair with key as word pairs in Text Format and value as counts of consolidated k,v pairs. The output (k,v) pair contains key as word pairs in Text format and value as sum of all the number of times a particular word pair co-occurred in the file, and is emitted by calling *write* method on context object.

• **DataFlow Description**: First off, the data to be processed is stored in Hadoop File System on the data node. The map task is used to process the data locally on each block where it is stored. The HDFS input file is retrieved in the form of k,v pair with with key as offset where the line starts and value as the line itself. This (k,v) pair is mapped using the defined map function, which in this case is emitting mapper output key-value pair. Now further, the reducer plays its role by processing the intermediate consolidated data using the reduce function i.e. to sum up all the values of all the values related a particular key emitted by a set of mappers, emitting the output key-value pair with key as the word pairs and value as count of co-occurrence. This output (k,v) pair is stored in already defined in defined HDFS directory which can further be accessed, displayed and modified using command-line script.

• Test Data and Results: The program is run and tested using Linux command-line scripts. It includes copying the data of local disk to hadoop file system using -cat while defining the directories where it is copying from and storing to. Further the driver, mapper and reducer classes are required to be compiled and exported as a .jar file. The next command runs the map reduce program assigning the tasks and

processing the data locally at the blocks. Further the output file *WordCoOccurrenceOutput/part-r-0000* of the Writable MapReduce program can be accessed or displayed using script commands