CS433 Parallel and distributed programming

Inverted Index

Tianhao Li 5140309349
Chao Gao 5142029014

2016.1.16

1. Introduction

In this project, we will start the distributed programming and have our first attempt on Hadoop which is an open-source software framework. And we choose to implement **Inverted Index** problem by using Java.
An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file or just a document. Inverted index is widely used in search engines for fast full text searches.

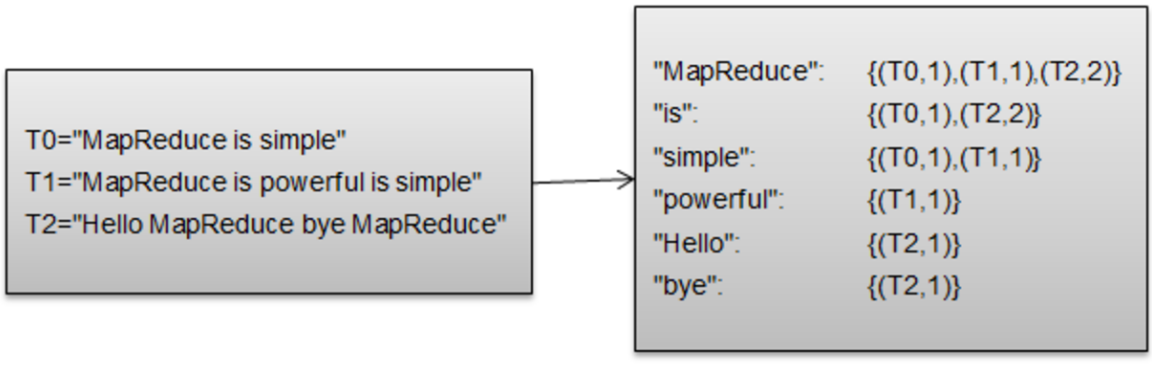
2. File structure

- report.pdf ———— this file
- InvertedIndexer.java ———— source code
- ReadMe.md ———— user guidance for running the code
- test.sh ———— test scripts
- result ———— test result

3. Program mechanism and implementation

3.1 Inverted index

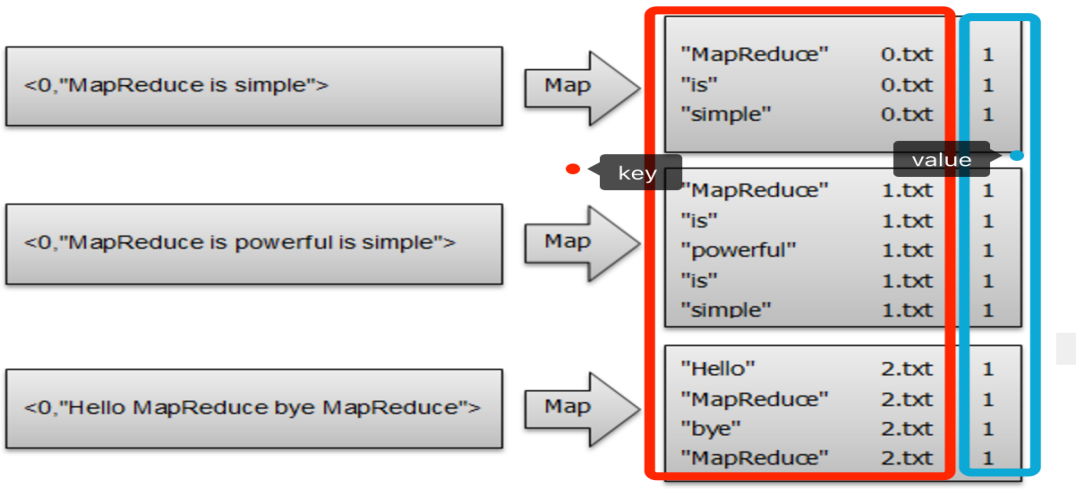
Simply speaking, inverted index stores a mapping from content to documents, in contrast to a forward index which maps from documents to content.
In this project, from a individual word, we want to get the documents(URI) that the word appears and calculate the frequency.
Below is the graph simply that shows what inverted index does.



3.2 Map

Use the built-in class to process the input file, we can get some **<key, value>** pairs, that is the offset and content. Now we start to process the key-value pairs that we get previously, there are three informations we need: word, document URI, frequency, in order to use the built-in sort function in the MapReduce framework, so here we make **word and URI** together as the key, and make **frequency** as the value.

Below is the graph that simply shows how Map works.



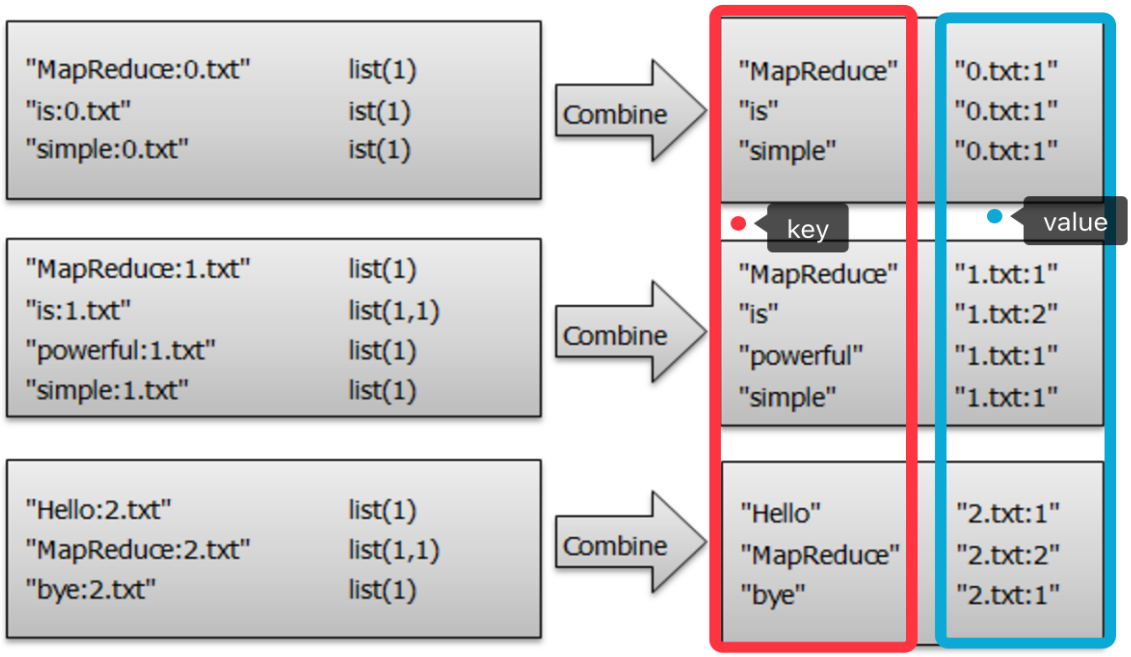
The code is as follows.

```
public static class TokenizerMapper extends Mapper<Object, Text, Text, Text> {
        private Text element = new Text();
        private Text one = new Text();

        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            //get file path
            InputSplit inputSplit = context.getInputSplit();
            String fileName = ((FileSplit) inputSplit).getPath().toString();

            //initial element
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                element.set(itr.nextToken() + ":" + fileName);
                one.set("1");
                context.write(element, one);
            }
        }
```

3.3 Combine

After the Map procedure, we take the Combine procedure to add values of the same key. Now we should reset the key-value pairs, make **word** as the key, and make **URI and frequency** as the value. Because we should make the records of the same word be processed by a single **reducer.**

Below is the graph that simply shows how Combine works.



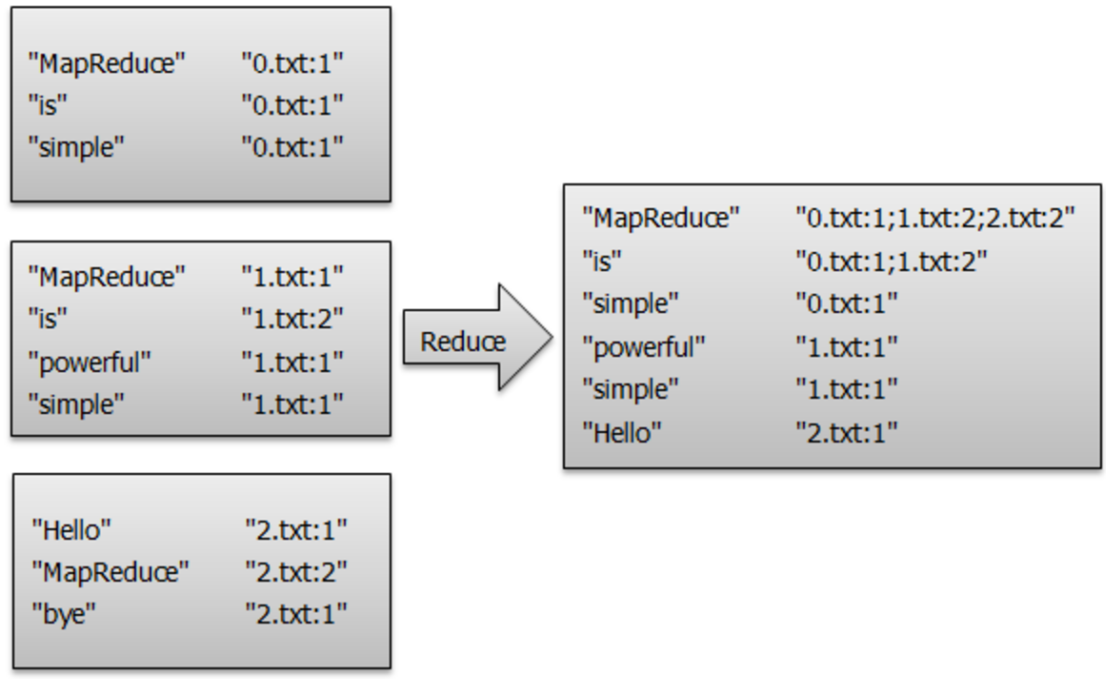
The code is as follows.

```
public static class SumCombiner extends Reducer<Text, Text, Text, Text> {
        private Text result = new Text();

        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
            //sum up the frequency
            int sum = 0;
            for (Text val : values) {
                sum += Integer.parseInt(val.toString());
            }

            //change unit format
            int splitIndex = key.toString().indexOf(":");
            result.set(key.toString().substring(splitIndex + 1) + ":" + sum);
            key.set(key.toString().substring(0, splitIndex));

            context.write(key, result);
        }
    }
```

3.4 Reduce

After previous two procedures, Reduce just produces the inverted index document.

Below is the graph that simply shows how Reduce works.



The code is as follows.

```
public static class InvertedIndexReducer extends Reducer<Text, Text, Text, Text> {
        private Text result = new Text();

        @Override
        protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text,
Text>.Context context)
                throws IOException, InterruptedException {

            //merge result
            String fileList = new String();
            for (Text value : values) {
                fileList += value.toString() + ";";
            }
            result.set(fileList);
            context.write(key, result);
        }
    }
```

4. Result and conclusion

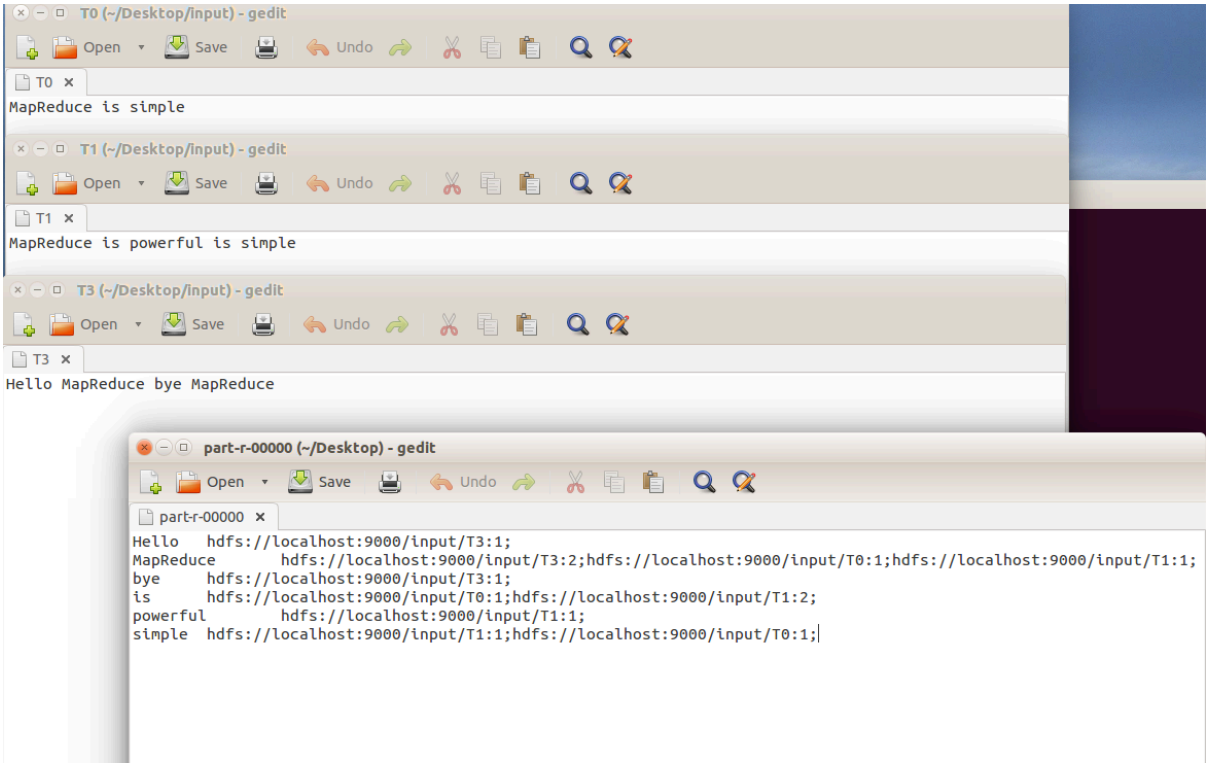
4.1 test scripts

```
#!/bin/sh
USER_NAME=emile
HADOOP_ROOT_PATH=../hadoop-2.7.3
HADOOP_LIB_PATH=../hadoop-2.7.3/share/hadoop

#Compile and Package the Code
javac -classpath ${HADOOP_LIB_PATH}/common/hadoop-common-2.7.3.jar:\
${HADOOP_LIB_PATH}/mapreduce/hadoop-mapreduce-client-core-
2.7.3.jar:${HADOOP_LIB_PATH}/common/lib/commons-cli-1.2.jar ./InvertedIndexer.java
jar cvf ./InvertedIndexer.jar *.class

#Clean
rm -rf ./master_output
${HADOOP_ROOT_PATH}/bin/hadoop fs -rm -r /user/${USER_NAME}/output
${HADOOP_ROOT_PATH}/bin/hadoop fs -rm -r /input

#Upload Data
${HADOOP_ROOT_PATH}/bin/hdfs dfs -mkdir /input
${HADOOP_ROOT_PATH}/bin/hdfs dfs -put ${HADOOP_ROOT_PATH}/input/* /input

#Run the Code
${HADOOP_ROOT_PATH}/bin/hadoop jar ./InvertedIndexer.jar InvertedIndexer /input output

${HADOOP_ROOT_PATH}/bin/hdfs dfs -get output master_output
cat ./master_output/*
```

Here we write the test.sh to test the program.

4.2 Result

We test the example that TA provided, the result is right.
Because the example output is too long, here we just show a very simple case to show the result.



4.3 Conclusion

The project is not difficult in fact. Getting familiar with Hadoop framework and the basic command takes most of the time. Writing the whole program is some kind of difficult in fact, but it would be much easier by referring to the code of **WordCount.java** in the MapReduce Package.

To make the test procedure much more efficient, we also write a bash script to do the test automatically. If you have any problem with using the test script, please refer to ReadMe.

5. Summary

In this project, we solve the Inverted index problem by using Hadoop. The project helps us has a better understanding about MapReduce, programming is not that difficult, but the environment setup is quite challenging, luckily we figure out the problems finally.

Thanks Prof.Deng for the guidance on Hadoop and TAs for answering our questions.