CS 422

DATA MINING

HOME WORK – 5

SUBMITTED BY :

SACHIN KRISHNA MURTHY

CWID : A20354077

MapReduce

You are given a large number of files with integers. Design and write very detailed pseudocode for the solution of the following problem:

Find the average for those numbers across all files.

Write the pseudocode for the mapper, reducer, provide very detailed comments for each line.

Solution :

MapReduce Computation :

- One or more chunks (collection of elements) from a distributed file system are given to each of the Map tasks, where these Map tasks turns these chunks into a sequence of key-value pairs. The code written by the user for the Map function determines the way key-value pairs are produced from the given data.
- After the formation of key-value pairs, the master controller collects these key-value pairs from each of the map task and is sorted by the key. Proceeding further these keys are divided among all the reduce tasks, so that all the key-value pairs with the same key wind up at the same reduce task.
- The Reduce tasks work on one key at a time and combine all the values associated with that key in some way. The code written by the user for the Reduce function determines the manner of combination of values.

Mapper :

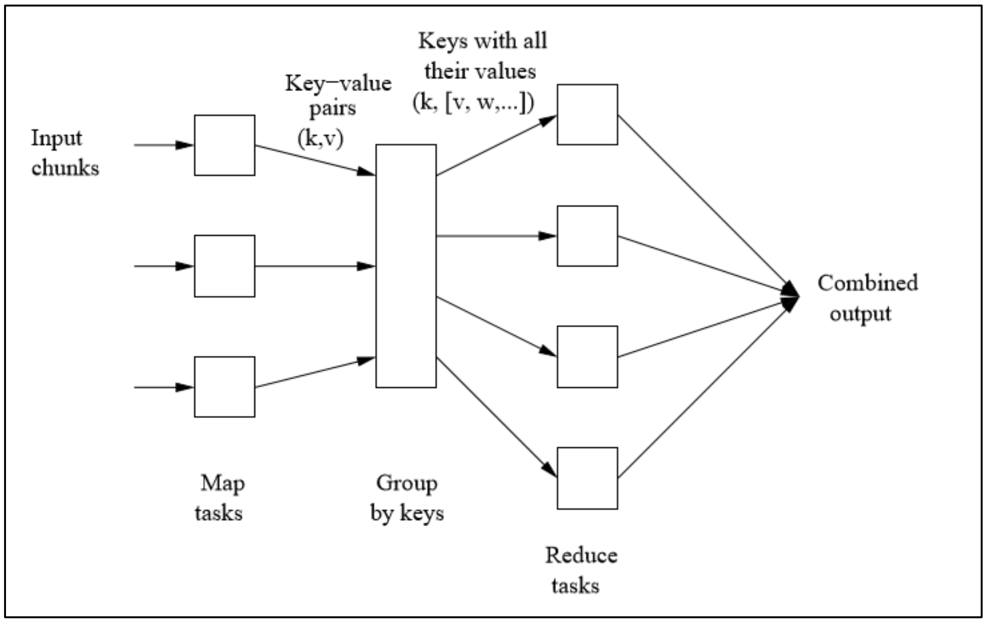
- The mapper performs the task by taking an input element as its argument and produces zero or more key-value pairs where the types of key and values are arbitrary.
- The input element may be a tuple or a document.
- The value of keys does not need to be unique.
- The Mapper can produce several key-value pairs with the same key, even from the same element.
- After the successful completion of the map tasks, the key-value pairs are grouped by key and the values associated with each key are formed into a list of values.
- The master controller process will be aware about the number of reduce tasks, for example consider r such tasks. The user informs the value of r to the MapReduce system.
- In the next step, the master controller picks a hash function that applies to keys and produces a bucket number from 0 to r−1.
- The output of the map task is a key and is hashed, also its key-value pair is put in one of r local files where each file is destined for one of the Reduce tasks.
- The master controller merges the files from each Map task that are destined for a particular Reduce task and feeds the merged file to that process as a sequence of key list of value pairs for the accomplishment of grouping by key and distribution to the Reduce tasks.
- That is, for each key k, the input to the Reduce task that handles key k is a pair of the form (k,[v1,v2,...,vn]), where (k,v1), (k,v2),...,(k,vn) are all the key-value pairs with key k coming from all the Map tasks.

Reducer :

- The input to the Reduce function is a pair consisting of a key and its list of associated values.
- The output of the Reduce function is a sequence of zero or more key-value pairs.
- These key-value pairs does not need to be of the same type from those sent from Map tasks, but usually they are of the same type.
- A Reduce task receives one or more keys and their associated value lists. That is, a Reduce task executes one or more reducers.
- The outputs from all the Reduce tasks are merged into a single file.
- Also partitioning the reducers is done by hashing the keys and associating each Reduce task with one of the buckets of the hash function among a small number of Reduce tasks.

Combiner :

- We can push some of the tasks of the reducers to the map, when the reduce function is both associative and commutative.
- For instance consider that the map task produces many pairs such as (w,v1), (w,v1)…and so on.
- In this case we could apply the reduce function within the map task before the output of the map task is subjected to groping and aggregation.
- Now, the pairs with key w generated by a single Map task would be replaced by a pair (w,m), where m is the number of times that w appears among the documents handled by this Map task.
- Thereafter it is still necessary to do grouping and aggregation, since there will typically be one key-value pair with key w coming from each of the Map tasks.



<u>**Pseudocode**</u>

<u>**Main Class :**</u>

```
Main(){

        folder = Directory where all files are located
        listOfFiles = list of all .txt files in the folder

        /* mapper returns map of format <key,value> where value is [sum,count] */
        for each file in listOfFiles
                map = mapper.readFile(file);
                add map to mapList
        end for

        /* Combiner returns map of format <key,value> where value is [average,count] */
        for each map1 in mapList
                conbineMap = combiner.combineMethod(map1);
                add conbineMap to combineMapList
        end for

        /* Reducer returns map of format <key,value> where value is the average */
        reducemap = reduce.reduceMap(combineMapList);
        average = average value for key = 1 in reducemap;

} //end of main class
```

<u>**Mapper Class :**</u>

```
Mapper {

/* The input is the file to be read */
        mapMethod (File file) {

                line = first line read from file
                fileSum = 0,
                fileCount = 0;

                /* File is read and Calculation is performed until it reaches the end of file */
                while (not end of file) {
                        numList = list of numbers in line

                        for each number in numList

                        /* fileSum = sum of all the elements in the file calculated iteratively */
                                fileSum = fileSum + number

                       /* file count = total number of elements in the file incremented iteratively*/
```

```
                        fileCount ++;
                    end for
                    line = next line read from file
                end while

                /* Inserting [sum,count] to map for key=1 */
                map.put(1, [fileSum, fileCount]);

                return map;
        end of method  mapMethod

} // end of class
```

Combiner Class :

```
public class Combiner {

/* Combine method takes map of format <key,[sum,count]> and calculates the average for each map
entry and returns map of format <key,[avg,count]> */

        combineMethod(List<Map<Integer, [sum,count]>> mapList) {

/* Sum of all the values is calculated for an individual file */
                fileSum = sum value for key = 1 in mapList;

                fileCount = count value for key = 1 in mapList

/* Average for the particular individual file is calculated */
                fileAverage = fileSum / fileCount;

                combineMap.put(1, [fileAverage, fileCount]);

                return combineMap;
        end of method combineMethod

} // end of class
```

Reducer Class :

```
public class Reduce {

/* Reduce method takes map of format <key,[avg,count]> and calculates the average for each map entry
and returns  the total average */

        reduceMethod(List<Map<Integer, [average,count]>> combineMapList) {
```

```
                    sum =0,
                    count = 0;

                    for each combineMap in combineMapList
                              fileAvg = average value for key = 1 in combineMapList
                              fileCount = count value for key = 1 in combineMapList

/* sum of values in the file is taken as the product of average of file values and total number of file
values */
                              sum = sum + (fileAvg * fileCount); // total sum calculated
                              count = count + fileCount;          // total count calculated
                    end for

                    average = sum / count;    // Calculating Final average of all the files

                    reducemap.put(1, [sum, average]);

                    return reducemap;
          end of method reduceMethod

} //end of class
```

Source Code :

Main Class :

```
/* The main class takes input from the folder and calls the other three classes to perform MapReduce
action */

public class AppMain {

          public static void main(String[] args) {

/* Object creation for the three classes mapper, combiner and reducer */
                    Mapper mapper = new Mapper();
                    Combiner combiner = new Combiner();
                    Reducer reducer = new Reducer();

/* A method to filter the size in a given directory to get only .txt files */
                    FilenameFilter filter = new FilenameFilter() {
                       public boolean accept(File dir, String name) {
                          return name.endsWith(".txt");
                       }
                    };
```

```
                         /* Declarations */
                        Map<Integer, FileValue> map = null;
                        Map<Integer, FileAvgValue> conbineMap = null;
                        Map<Integer, Float> reduceMap = null;
                        List<Map<Integer, FileValue>> mapList = new ArrayList<Map<Integer,FileValue>>();
                        List<Map<Integer, FileAvgValue>> combineMapList = new
ArrayList<Map<Integer,FileAvgValue>>();

                    /* Getting directory path of the input files */
                        File folder = new File("C:\Users\Sachin K\Desktop\DM Assignment 6 ");

                    /* Getting required files from given folder */
                        File[] listOfFiles = folder.listFiles(filter);

                        for (int i = 0; i < listOfFiles.length; i++) {
                           File file = listOfFiles[i];
                           map = new HashMap<Integer, FileValue>();
                           map = mapper.mapMethod(file);        //Mapper called for each file

                       /* Mapper returns Map of format <int,[sum,count]> */
                           mapList.add(map);
                        }

                        for (Map<Integer, FileValue> map1 : mapList) {

                                   conbineMap = new HashMap<Integer, FileAvgValue>();

                               /* Combiner is called for each map in the map list received from mapper */
                                   conbineMap = combiner.combineMethod(map1);
                                   combineMapList.add(conbineMap);
                        }

                        reduceMap = new HashMap<Integer, Float>();

                    /* Reducer is called for the whole combine map received from combine method */
                        reduceMap = reducer.reduceMethod(combineMapList);
                        Float average = reduceMap.get(1);
                        System.out.println("Average : " + average);

              }

}
```

File Value class:

```
/* Objects of this are used as values in map returned from mapper class */

/* It contains sum and count */

public class FileValue {

        int sum;
        int count;

        public FileValue(int sum, int count) {  // Constructor
                this.sum = sum;
                this.count = count;
        }

        public int getSum() {    // getter
                return sum;
        }
        public int getCount() {    // getter
                return count;
        }

}
```

File Average Value class:

```
/* Objects of this are used as values in map returned from combiner class */

/* It contains average and count */

public class FileAvgValue {

        float average;
        int count;

        public FileAvgValue(float average, int count) {   // constructor
                this.average = average;
                this.count = count;
        }

        public float getAverage() {   // getter
                return average;
        }
        public int getCount() {    //getter
                return count;
        }

}
```

Mapper Class :

```
public class Mapper {

        public Map<Integer, FileValue> mapMethod(File file) {
                Map<Integer, FileValue> map = new HashMap<Integer, FileValue>();
                BufferedReader bf;
                try {
                        /* Reading file using buffer reader*/
                        FileReader fr = new FileReader(file);
                        bf = new BufferedReader(fr);

                       /* Reading first line of the file */
                        String line = bf.readLine();
                        int fileSum = 0, fileCount = 0;

                     /* Reading file until end of file is reached */
                        while (null != line) {

                             /* Splitting lines into individual integers or values */
                                String[] numList = line.split(" ");
                                for (String string : numList) {

                                      /* Calculating the sum of the values of file */
                                        fileSum = fileSum + Integer.parseInt(string);
                                        fileCount ++;
                                }
                             /* Reading next lines from the file */
                                line = bf.readLine();
                        }

                              /* Setting sum and count */
                        FileValue fileValue = new FileValue(fileSum,fileCount);

                      /* Inserting into the map for key=1 */
                        map.put(1, fileValue);

                     /* Closing file reader */
                        bf.close();
                } catch (FileNotFoundException e) {
                        System.out.println("FileNotFoundException");
                }catch (IOException e) {
                        System.out.println("IOException");
                }
                return map;
        }
}
```

Combiner Class :

```
public class Combiner {

        public Map<Integer, FileAvgValue> combineMethod(Map<Integer, FileValue> map) {
                Map<Integer, FileAvgValue> combineMap = new HashMap<Integer, FileAvgValue>();

             /* Getting sum for the individual file */
                int fileSum = map.get(1).getSum();

             /* Getting count for the individual file */
                int fileCount = map.get(1).getCount();

             /* calculating average within individual file */
                float fileAverage = (float)fileSum / (float)fileCount;

             /* Setting file average and count */
                FileAvgValue fileAvgValue = new FileAvgValue(fileAverage, fileCount);

             /* Putting into combine map where key=1 */
                combineMap.put(1, fileAvgValue);

                return combineMap;
        }

}
```

Reducer Class :

```
public class Reducer {

        public Map<Integer, Float> reduceMethod(List<Map<Integer, FileAvgValue>> combineMapList) {

                Map<Integer, Float> reduceMap = new HashMap<Integer, Float>();

                int num = 1;
                        float sum = 0;
                        int count = 0;
                        for (Map<Integer, FileAvgValue> map : combineMapList) {

                              /* Getting average of single file */
                                float fileAvg = map.get(num).getAverage();

                              /* Getting count of single file */
                                int fileCount = map.get(num).getCount();
                /* Calculating the total sum of file values in all files */
                    sum = sum + (fileAvg * fileCount);

                /* Calculating the total count of file values in all files */
                    count = count + fileCount;
                }

            /* Calculating the  total average */
                float average = sum / count;

            /* Inserting into map */
                reduceMap.put(num, average);

        return reduceMap;
    }

}
```

Example :

Consider 3 files with contents :
File1 {1 2 3 4}
File2 {5 6}
File3 {7 8 9}

Mapper :

Mapper reads file, returns sum and count for each file.

For File1:

```
    Sum = 10
    Count = 4
    Map1 = {1,[10,4]}    //map of type {key,value} where key =1 and value = [sum,count]
    return Map1
```

For File2:

```
    Sum = 11
    Count = 2
    Map2 = {1,[11,2]}
    return Map2
```

For File3:

```
    Sum = 24
    Count = 3
    Map3 = {1,[24,3]}
    return Map3
```

Main put all these three maps in a list and calls combiner for each map.

Combiner :

Combiner reads map , returns average and count for each file.
For Map1:

```
    Sum = 10
    Count = 4
    Average = 2.5
    map = {1,[2.5,4]}    //map of type {key,value} where key =1 and value = [Average,count]
    return map
```

For Map2:

```
    Sum = 11
    Count = 2
    Average = 5.5
    map = {1,[5.5,2]}
    return map
```

For Map3:

```
    Sum = 24
    Count = 3
    Average = 8
    map = {1,[8,3]}
    return map
```

Main put all these three maps in a list and calls reducer.

Reducer :

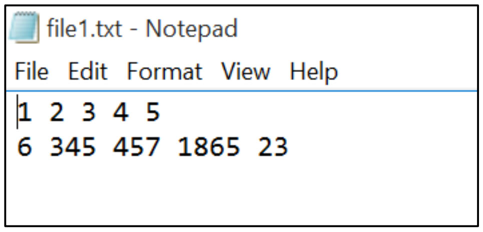
Reducer reads list of maps after combiner, returns average form list.
For key = 1

```
    Sum = (2.5 * 4) + (5.5 * 2) + (8 * 3)
        = 10 + 11 + 24
        = 45
    Count = 4 + 2 + 3
          = 9
    Average = 45 / 9
            = 5
    map = {1,5}  //map of type {key,value} where key =1 and value = Average
    return map
```

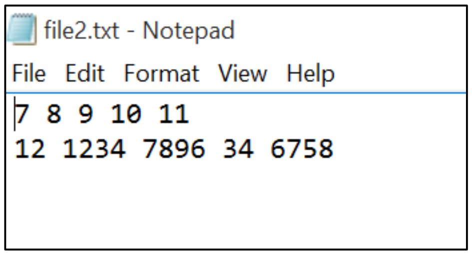
Main receives map with final average = 5 for key =1

Input :

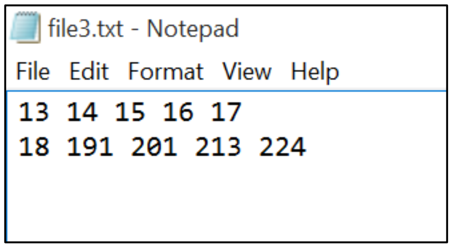
Considering 4 files as the input with the integer values as shown in the below screen shots :



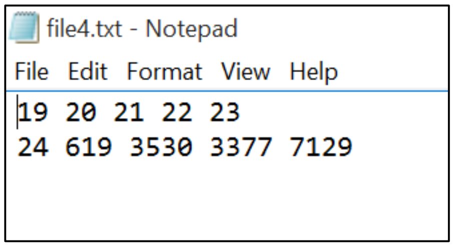
```
1 2 3 4 5
6 345 457 1865 23
```



```
7 8 9 10 11
12 1234 7896 34 6758
```

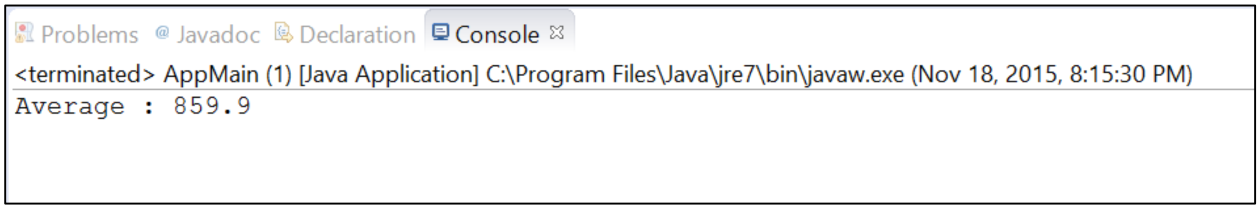


```
13 14 15 16 17
18 191 201 213 224
```



```
19 20 21 22 23
24 619 3530 3377 7129
```

Output :



```
<terminated> AppMain (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 18, 2015, 8:15:30 PM)
Average : 859.9
```