LAB ASSIGNMENT #1

Team ID #5

Class ID: 21 (Sudheer Nimmagadda)

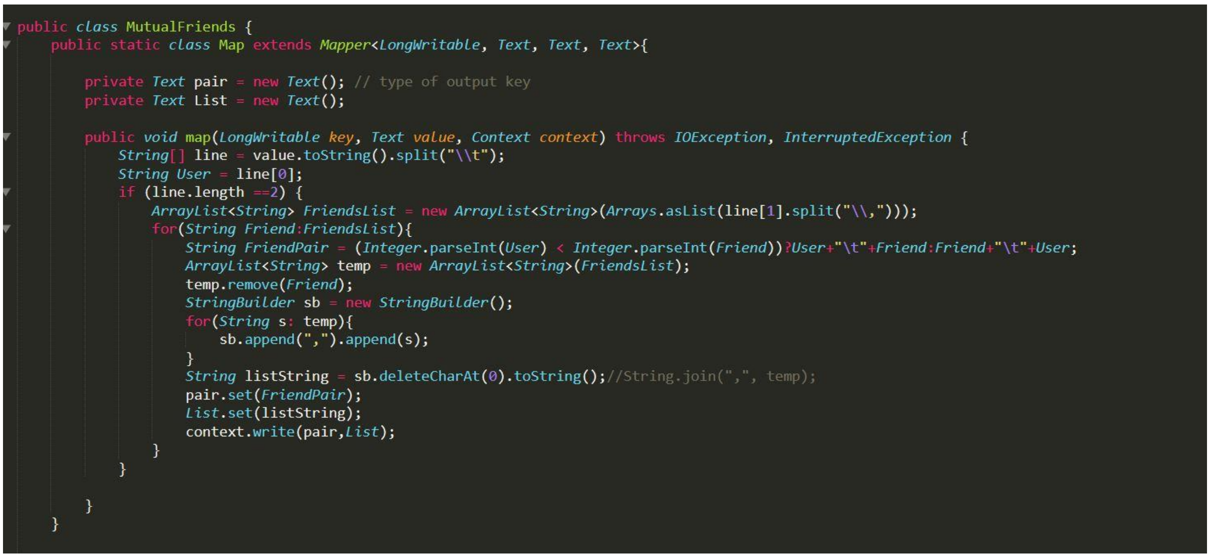
Class ID: 25 (Jaya Prakash Ravella)

1. For finding facebook common friends we take the input as given in the question and the output is the mutual friends.

Work Flow:

Mapper Phase Code:

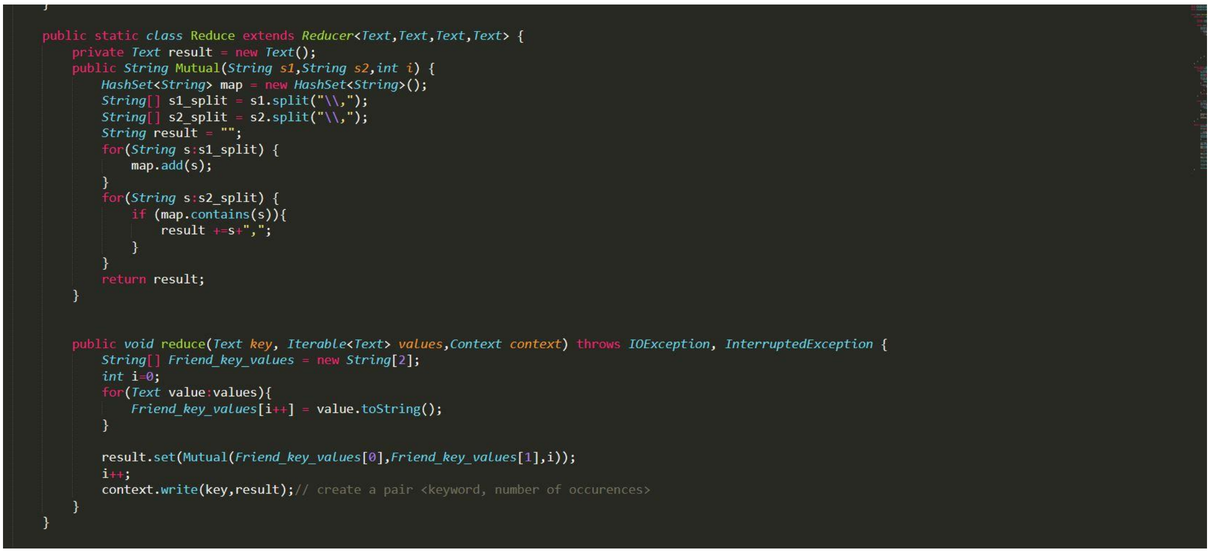
We create a mapper class as shown in the code screenshot below. Each line of the input file is splitted by “tab” delimiter. Then its size is computed as two, where the first part is source or base user and the rest of the split considered as list of friends of the user. Then the keys are prepared as (1, 2) or (2, 1) based on the integer values of 1 & 2 in the input.



```
public class MutualFriends {
    public static class Map extends Mapper<LongWritable, Text, Text, Text>{

        private Text pair = new Text(); // type of output key
        private Text List = new Text();

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String[] line = value.toString().split("\\t");
            String User = line[0];
            if (line.length ==2) {
                ArrayList<String> FriendsList = new ArrayList<String>(Arrays.asList(line[1].split("\\,")));
                for(String Friend:FriendsList){
                    String FriendPair = (Integer.parseInt(User) < Integer.parseInt(Friend))?User+"\t"+Friend:Friend+"\t"+User;
                    ArrayList<String> temp = new ArrayList<String>(FriendsList);
                    temp.remove(Friend);
                    StringBuilder sb = new StringBuilder();
                    for(String s: temp){
                        sb.append(",").append(s);
                    }
                    String listString = sb.deleteCharAt(0).toString();//String.join(",", temp);
                    pair.set(FriendPair);
                    List.set(listString);
                    context.write(pair,List);
                }
            }

        }
    }
```

Reducer Phase Code:

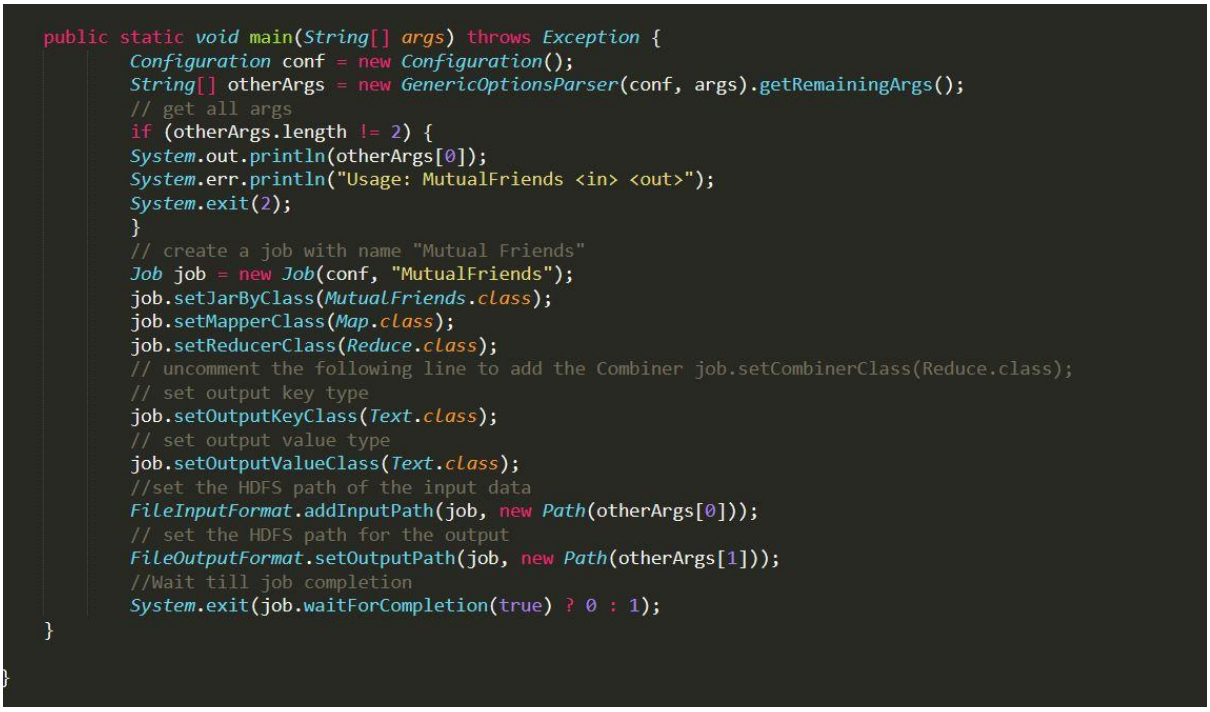
We created a reducer class where the data is grouped based on the key values (1, 2) or (2, 3) and their list of friends are as produced. Then finally reduced to find the mutual friends of (1, 2). Here we used a function Mutual where it takes arguments the friend key values and then it computes the mutual friends and adds to the result.



```java
}

public static class Reduce extends Reducer<Text,Text,Text,Text> {
    private Text result = new Text();
    public String Mutual(String s1,String s2,int i) {
        HashSet<String> map = new HashSet<String>();
        String[] s1_split = s1.split("\\,");
        String[] s2_split = s2.split("\\,");
        String result = "";
        for(String s:s1_split) {
            map.add(s);
        }
        for(String s:s2_split) {
            if (map.contains(s)){
                result +=s+",";
            }
        }
        return result;
    }


    public void reduce(Text key, Iterable<Text> values,Context context) throws IOException, InterruptedException {
        String[] Friend_key_values = new String[2];
        int i=0;
        for(Text value:values){
            Friend_key_values[i++] = value.toString();
        }

        result.set(Mutual(Friend_key_values[0],Friend_key_values[1],i));
        i++;
        context.write(key,result);// create a pair <keyword, number of occurences>
    }
}
```

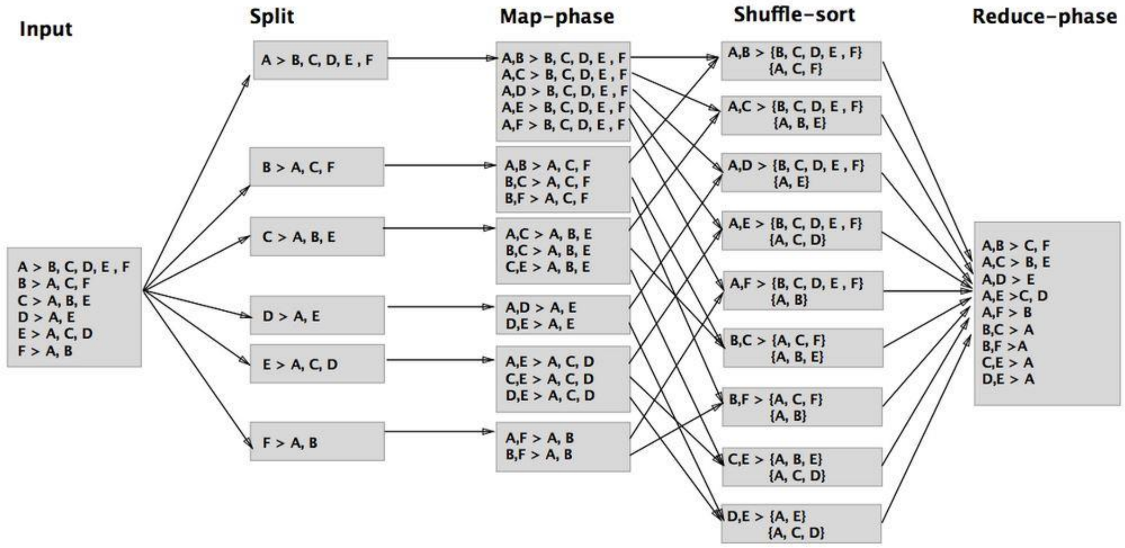
Driver Phase Code:

A main method in the driver to set mapper and reducer class which takes the input and produces the output.



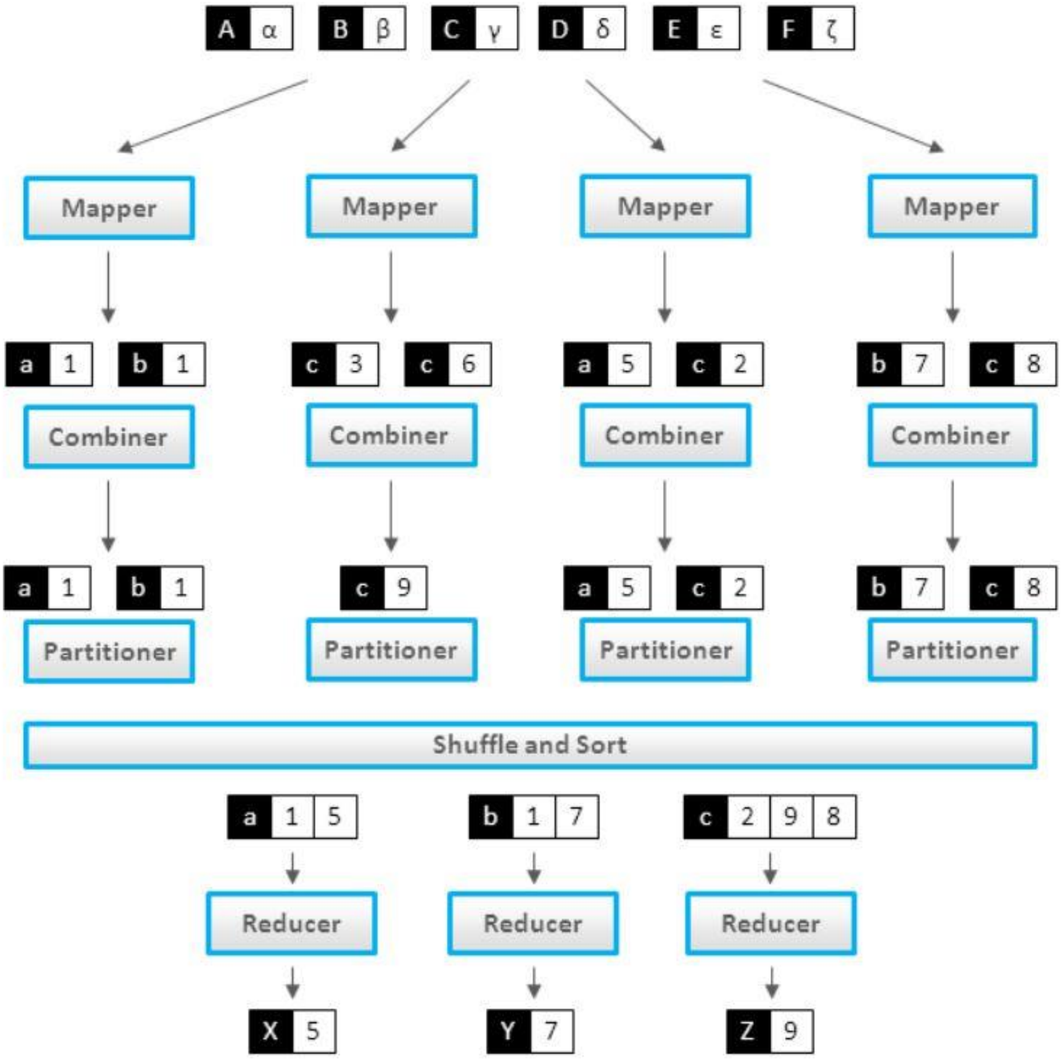
```java
    public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();
            String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
            // get all args
            if (otherArgs.length != 2) {
            System.out.println(otherArgs[0]);
            System.err.println("Usage: MutualFriends <in> <out>");
            System.exit(2);
            }
            // create a job with name "Mutual Friends"
            Job job = new Job(conf, "MutualFriends");
            job.setJarByClass(MutualFriends.class);
            job.setMapperClass(Map.class);
            job.setReducerClass(Reduce.class);
            // uncomment the following line to add the Combiner job.setCombinerClass(Reduce.class);
            // set output key type
            job.setOutputKeyClass(Text.class);
            // set output value type
            job.setOutputValueClass(Text.class);
            //set the HDFS path of the input data
            FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
            // set the HDFS path for the output
            FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
            //Wait till job completion
            System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

}
```

Map Reduce Diagram:

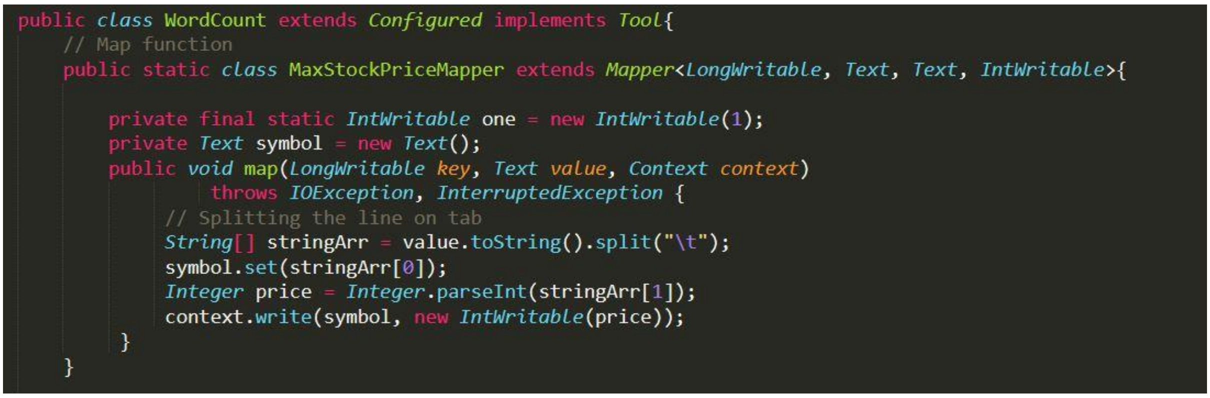


2. For this question we used multiple input files which consists of symbols along with values and the output is the maximum value for each of the symbol.

MapReduce diagram is as follows,



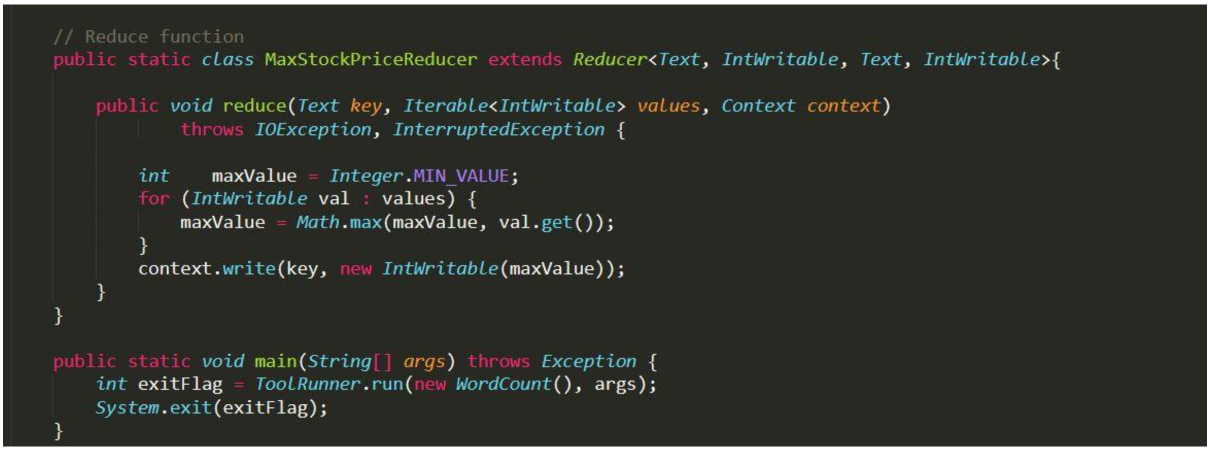
Mapper Phase Code-



```
public class WordCount extends Configured implements Tool{
    // Map function
    public static class MaxStockPriceMapper extends Mapper<LongWritable, Text, Text, IntWritable>{

        private final static IntWritable one = new IntWritable(1);
        private Text symbol = new Text();
        public void map(LongWritable key, Text value, Context context)
                 throws IOException, InterruptedException {
             // Splitting the line on tab
             String[] stringArr = value.toString().split("\t");
             symbol.set(stringArr[0]);
             Integer price = Integer.parseInt(stringArr[1]);
             context.write(symbol, new IntWritable(price));
        }
    }
```

Here we split the input file using “tab” delimiter and returns the key value pairs.

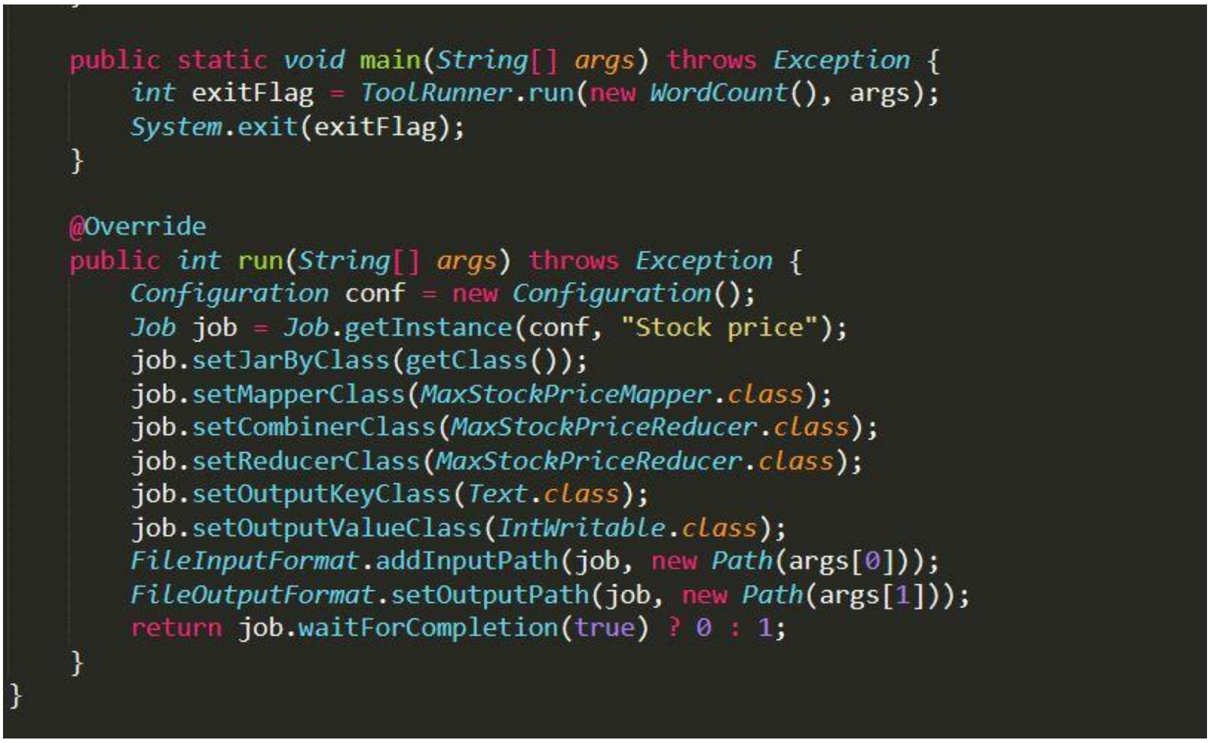
Reduce Phase Code:



```
    // Reduce function
    public static class MaxStockPriceReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {

            int    maxValue = Integer.MIN_VALUE;
            for (IntWritable val : values) {
                maxValue = Math.max(maxValue, val.get());
            }
            context.write(key, new IntWritable(maxValue));
        }
    }

    public static void main(String[] args) throws Exception {
        int exitFlag = ToolRunner.run(new WordCount(), args);
        System.exit(exitFlag);
    }
```

Reducer iterates through each value for the specific key and returns the maximum value for that key.

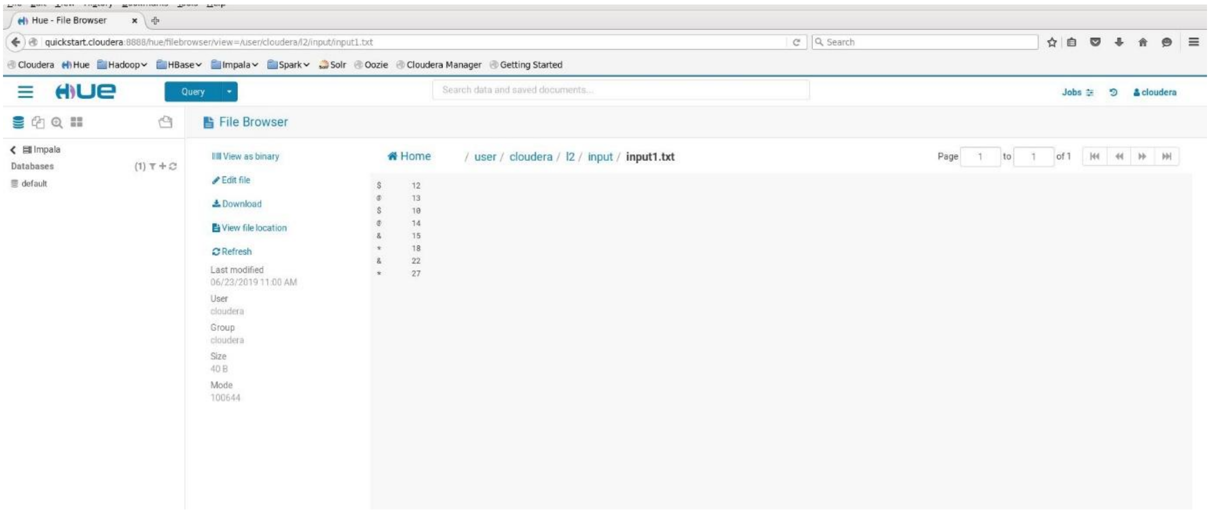
Driver Phase Code:



```
    public static void main(String[] args) throws Exception {
        int exitFlag = ToolRunner.run(new WordCount(), args);
        System.exit(exitFlag);
    }

    @Override
    public int run(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Stock price");
        job.setJarByClass(getClass());
        job.setMapperClass(MaxStockPriceMapper.class);
        job.setCombinerClass(MaxStockPriceReducer.class);
        job.setReducerClass(MaxStockPriceReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        return job.waitForCompletion(true) ? 0 : 1;
    }
}
```

In the Driver code we used the combiner to reduce the job burden to the reducer. So, with combiner it is fast while running the job along with the reducer.

Given two input files as shown below,

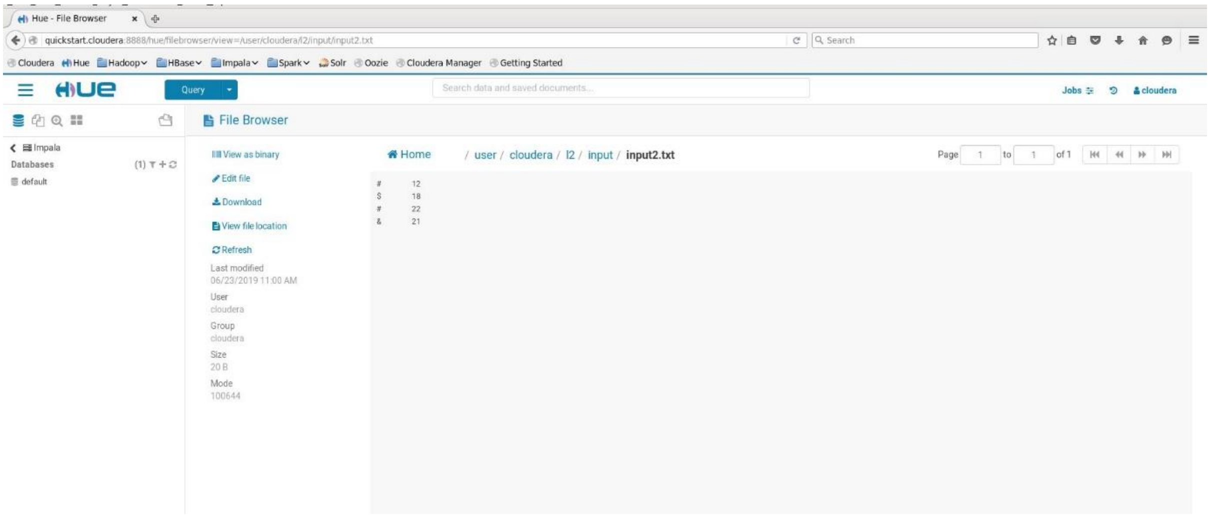


quickstart.cloudera:8888/hue/filebrowser/view=/user/cloudera/l2/input/input1.txt

Home / user / cloudera / l2 / input / input1.txt

Last modified
06/23/2019 11:00 AM
User
cloudera
Group
cloudera
Size
40 B
Mode
100644

```
$       12
@       13
$       10
@       14
&       15
*       18
&       22
*       27
```



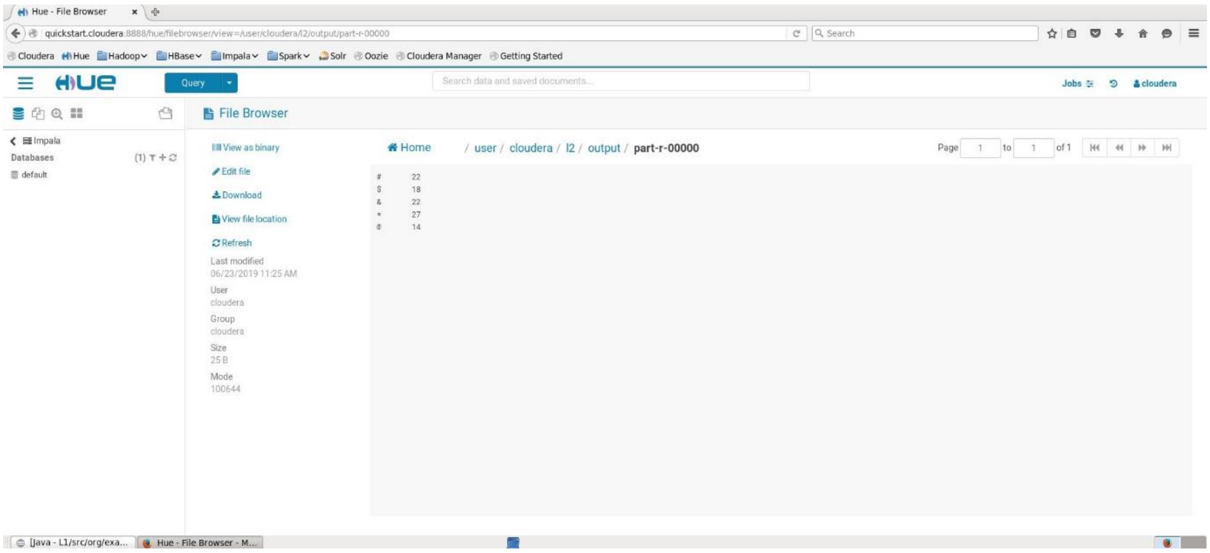
quickstart.cloudera:8888/hue/filebrowser/view=/user/cloudera/l2/input/input2.txt

Home / user / cloudera / l2 / input / input2.txt

Last modified
06/23/2019 11:00 AM
User
cloudera
Group
cloudera
Size
20 B
Mode
100644

```
#       12
$       18
#       22
&       21
```

The output will be as shown below,



quickstart.cloudera:8888/hue/filebrowser/view=/user/cloudera/l2/output/part-r-00000

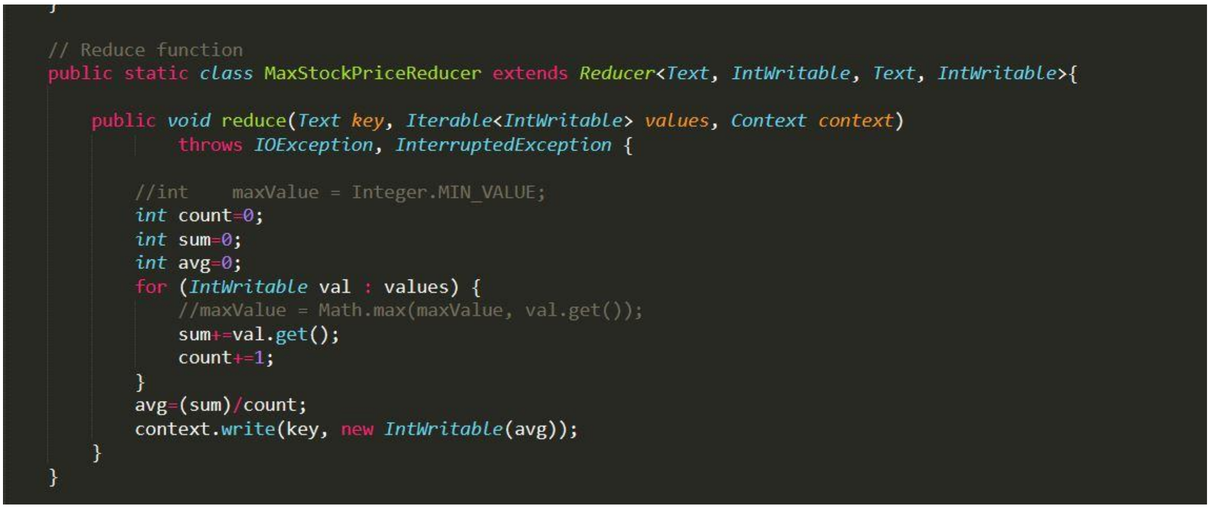
Home / user / cloudera / l2 / output / part-r-00000

Last modified
06/23/2019 11:25 AM
User
cloudera
Group
cloudera
Size
25 B
Mode
100644

```
#       22
$       18
&       22
*       27
@       14
```

For finding the average of response times the mapper code is same and Driver code too but with a small change in the logic in the reducer code.

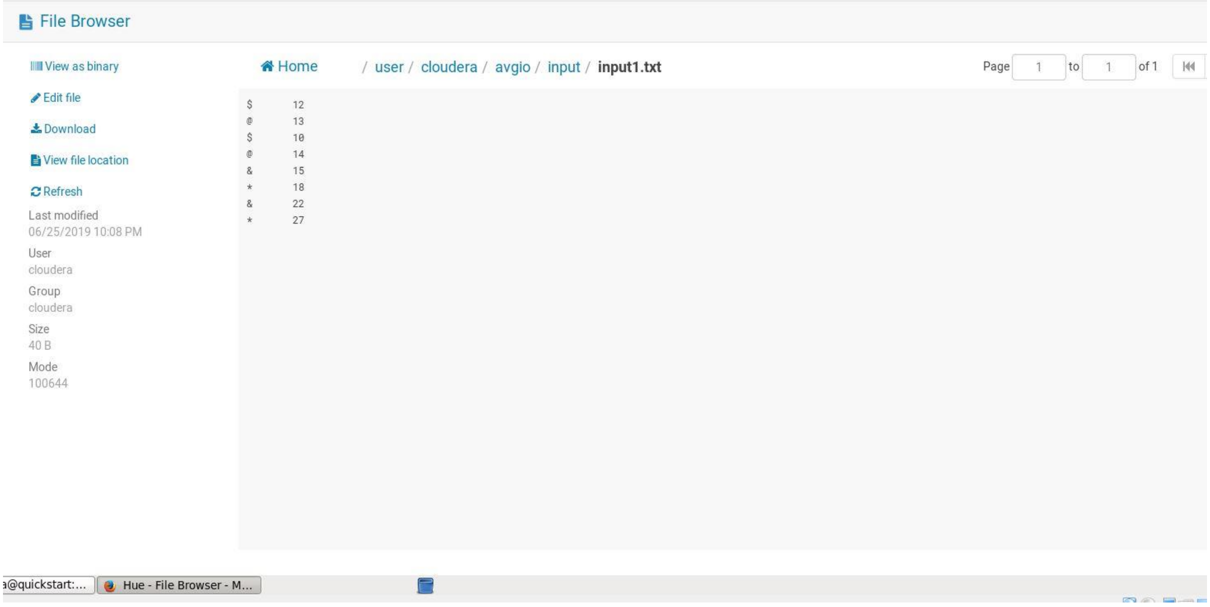
Reducer Phase Code:



```
    // Reduce function
    public static class MaxStockPriceReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {

            //int    maxValue = Integer.MIN_VALUE;
            int count=0;
            int sum=0;
            int avg=0;
            for (IntWritable val : values) {
                //maxValue = Math.max(maxValue, val.get());
                sum+=val.get();
                count+=1;
            }
            avg=(sum)/count;
            context.write(key, new IntWritable(avg));
        }
    }
```

In the above screenshot here we find the sum of values and count values for each key and we calculate average and write to the output key, value classes.

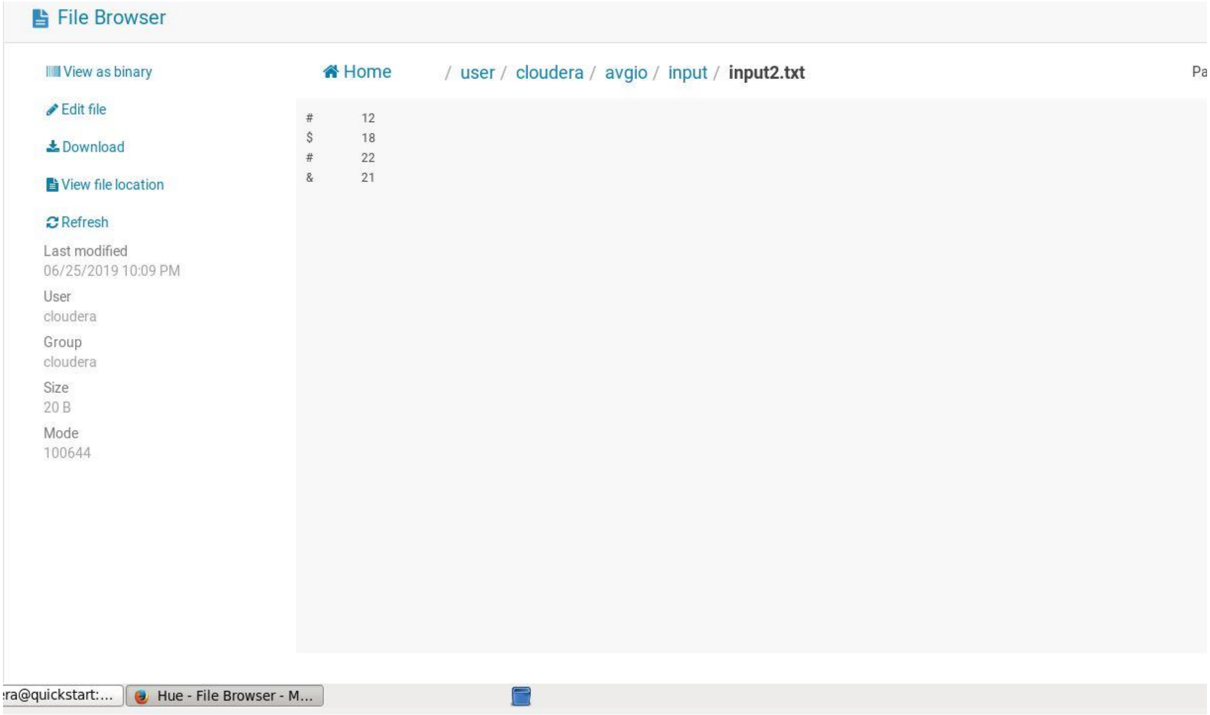
The input files will be as shown below,



/ user / cloudera / avgio / input / input1.txt

Last modified
06/25/2019 10:08 PM
User
cloudera
Group
cloudera
Size
40 B
Mode
100644

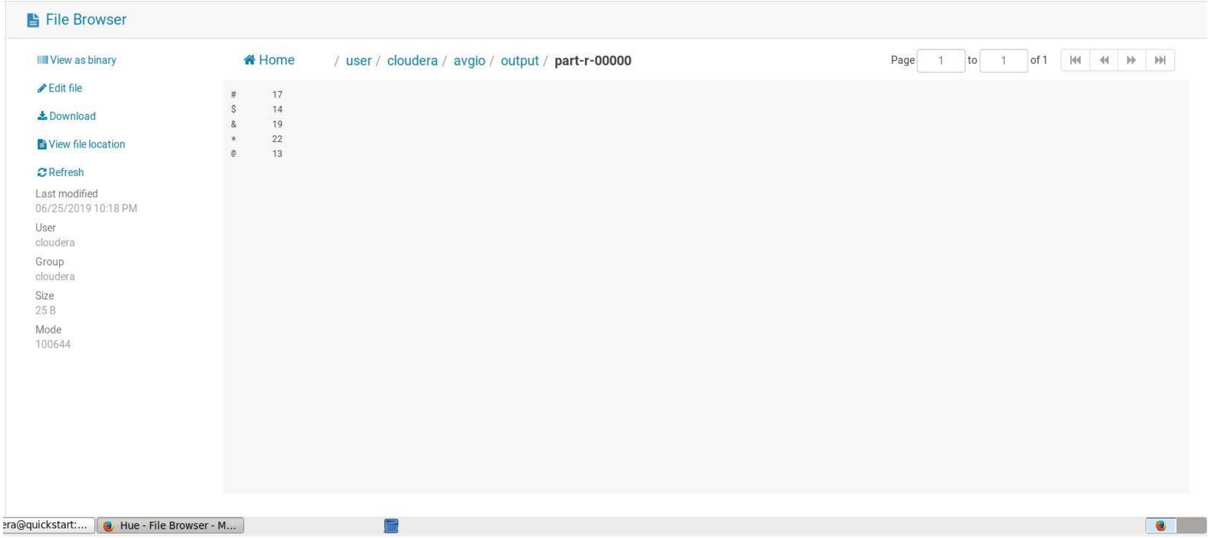
```
$       12
@       13
$       10
@       14
&       15
*       18
&       22
*       27
```



Last modified
06/25/2019 10:09 PM
User
cloudera
Group
cloudera
Size
20 B
Mode
100644

```
#	12
$	18
#	22
&	21
```

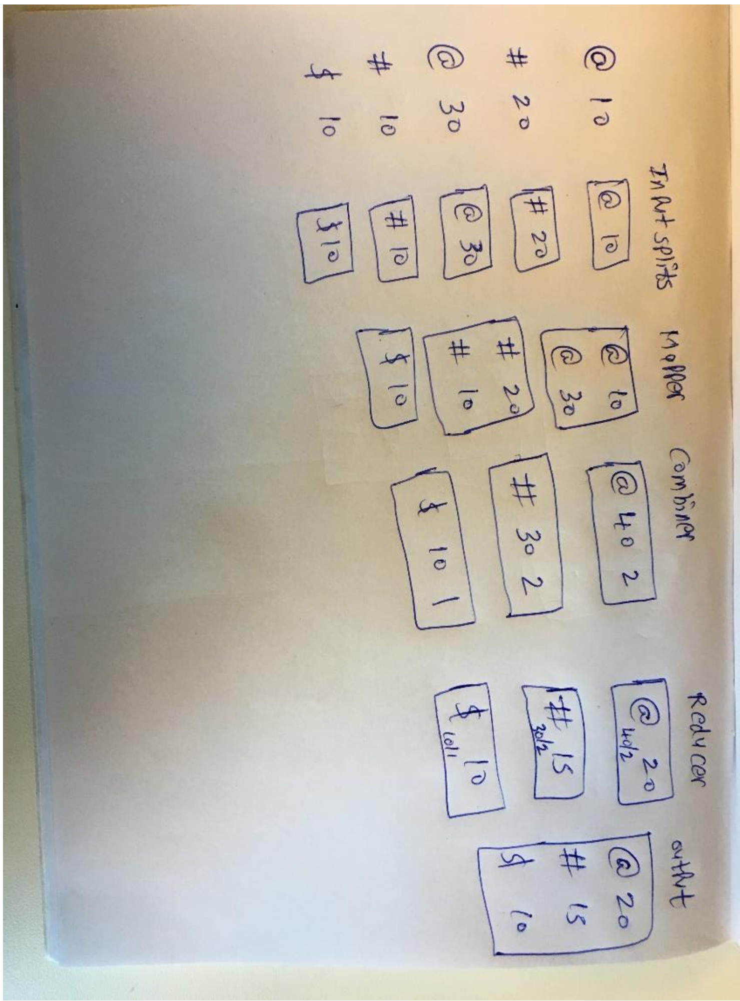
The output file will be as shown below,



Last modified
06/25/2019 10:18 PM
User
cloudera
Group
cloudera
Size
25 B
Mode
100644

```
#	17
$	14
&	19
*	22
@	13
```

MapReduce Diagram is as follows for the average,



3. Here we used Zomato data, the creation of tables and the queries related to are in the github wiki I will walk through the queries in the video.

4. Here we also used Zomato data, the instance directory creation , the collection and the nested queries in solr are in the github wiki I will walk through the queries and creating collection in the video.

References:

1. http://stevekrenzel.com/finding-friends-with-mapreduce

2. https://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/