HADOOP MAP-REDUCE ON UBER DATA SET

VISHWANATH BALASUBRAMANIAN, SURAJ KOSHY THOMAS & ALEN JAGAN

vishwanathbalasubramanian1997@gmail.com,surajkoshythomas@karunya.edu.in&alenj777@yahoo.com

INDEX:

INTRODUCTION:

Uber is an American technology company headquartered in San Francisco, California, United States, operating in 570 cities worldwide. It develops, markets and operates the Uber car transportation and food delivery mobile apps. Uber drivers use their own cars, although drivers can rent a car to drive with Uber. The name "Uber" is a reference to the common (and somewhat slangy) word "Uber", meaning "topmost" or "super", and having its origins in the German word *über*, meaning "above". Uber has been a pioneer in the sharing economy and the changes in industries as a result of the sharing economy have been referred to as "Uberification" or "Uberisation".

Uber uses your personal data in an anonymized and aggregated form to closely monitor which features of the Service are used most, to analyze usage patterns and to determine where we should offer or focus our Service. We may share this information with third parties for industry analysis and statistics.

While the ever-present specter of data misuse is nipping at Uber's heels, there's no doubt that the anonymous, aggregated data they collect insights from is nothing short of amazing.
All of this data is collected, crunched, analyzed and used to predict everything from the customer's wait time, to recommending where drivers should place themselves via heatmap in order to take advantage of the best fares and most passengers. All of these items are implemented in real-time for both drivers and passengers alike.

INTERSETING FACT:

"UBER, the world's largest taxi company owns no vehicles."

PROBLEM STATEMENTS:

Uber is a very famous cab service across the world. Trips and services, they offer every day is literally huge and good for analysis. Given to you is the uber dataset with details of every trip. Come up with a mapreduce solution to find out:

1. Dates on which each basement has >1000 active vehicles.

2. Basements which has top3 average trips.

Create a jar with your mapreduce code and test it on the dataset placed in hdfs. Clearly report every stage for you project work.

ABSTRACT:

Across borders, cultures, and languages, Uber has connected people who need a reliable ride with people looking to earn money driving their car. Getting a ride from an Uber driver is beautiful in its simplicity: simply open the app, set the pickup location, request a car, get picked up and pay with the tap of a button. But there's a great deal of data wrangling going on to make all of this happen in a (relatively) smooth process. Add to that the fact that sometimes there are things out of even Uber's control, like poor city transportation infrastructure, traffic jams, uncooperative drivers and much more. Of course, collecting all this information is just one step in the big data journey. The real question is — how does Uber determine the best way to make decisions using this information? How do they glean actionable points out of the data they collect?. This is the part where the Big-Data comes into play. The main reason for using this data set is because it has a live dataset which can be used as a very good learning method to understand working of Big-Data. We took the public data provided by the Uber company in this process. It uses Hadoop MapReduce to analyze and arrive at:

1. Dates on which each basement has >1000 number of active vehicles.
2. basements which has top 3 average trips

DATASET DESCRIPTION:

Column 1: dispatching_base_number

Column 2: date

Column 3: active_vehicles

Column 4: trips

TOOLS REQUIRED:

Since the project is done in windows there are some tools which comes under prerequisites. They are

1.VITRUALBOX (which we used, there are other virtual machines that can be used).

2.PUTTY (which acts as a terminal for running the Hadoop commands).

3.WinSCP (which is used to transfer files from windows to Linux OS used in VirtualBox).

PROBLEM STATEMENT I:

"Dates on which each basement has >1000 active vehicles."

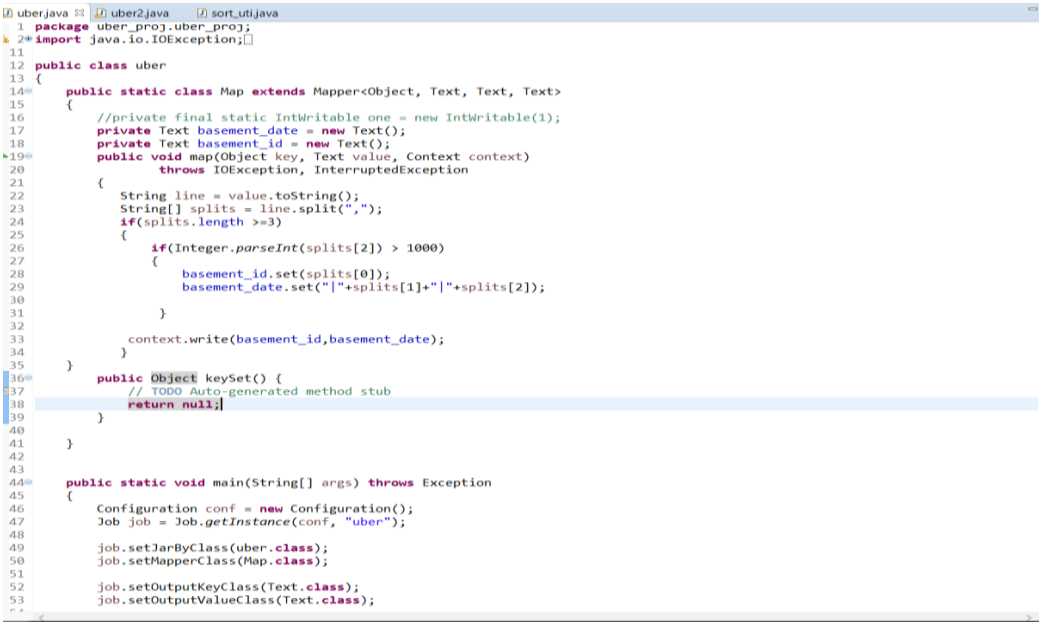
KEY VALUE:

The key value for this problem set is Basement_id and Basement_date (Both Date and Active Vehicles). The reason why Basement_id as the key is to collect all the basements that are available in the dataset. The value is both the date as well as the trips, which will be used to calculate the dates on which each basement has more than thousand active vehicles.

MAPPER:

In mapper function, we have read the data line by line. Splitting up of data is done over space and "|" and it is stored. The columns of dispatching_base_number is taken and they are passed as key from mapper to the framework. The columns of date and active trips are taken as the values from the mapper to the framework. The main advantage in this problem is that there is no need for a reducer phase since we don't have the need to do calculation work like addition or subtraction. All the selection work can be done in the mapper phase itself. Therefore, the compilation part of this problem becomes much simpler and much more efficient.

SCREENSHOTS:



uber.java | uber2.java | sort_uti.java

```
1  package uber_proj.uber_proj;
2  import java.io.IOException;
11
12 public class uber
13 {
14     public static class Map extends Mapper<Object, Text, Text, Text>
15     {
16         //private final static IntWritable one = new IntWritable(1);
17         private Text basement_date = new Text();
18         private Text basement_id = new Text();
19         public void map(Object key, Text value, Context context)
20                 throws IOException, InterruptedException
21         {
22             String line = value.toString();
23             String[] splits = line.split(",");
24             if(splits.length >=3)
25             {
26                 if(Integer.parseInt(splits[2]) > 1000)
27                 {
28                     basement_id.set(splits[0]);
29                     basement_date.set("|"+splits[1]+"|"+splits[2]);
30
31                  }
32
33              context.write(basement_id,basement_date);
34             }
35     }
36         public Object keySet() {
37             // TODO Auto-generated method stub
38             return null;
39         }
40
41     }
42
43
44     public static void main(String[] args) throws Exception
45     {
46         Configuration conf = new Configuration();
47         Job job = Job.getInstance(conf, "uber");
48
49         job.setJarByClass(uber.class);
50         job.setMapperClass(Map.class);
51
52         job.setOutputKeyClass(Text.class);
53         job.setOutputValueClass(Text.class);
```

OUTPUT:

```
[root@sandbox ~]# hdfs dfs -cat /uber_out_15/p* | head -10


B02598  |1/16/2015|1079
B02598  |1/16/2015|1079
B02598  |1/31/2015|1027
B02598  |1/31/2015|1027
B02598  |1/21/2015|1035
B02598  |1/30/2015|1106
B02598  |1/30/2015|1106
B02598  |1/30/2015|1106
```

PROBLEM STATEMENT II:

"Basements which has top 3 average trips"

KEY VALUE:

The key of the second problem is same as the previous one which is the dispatch_base_number column because we must segregate the different basements that are present in the dataset.

The value of this problem is trips columns which shows the total trips that are taken in the date in the basement.

This helps in finding the top 3 basements from the given dataset.

MAPPER:

Data is read line by line. Splitting up of data is done over space and it is stored. The basement is used as a key and the trips is used as the value. These are sent from the mapper to the framework.

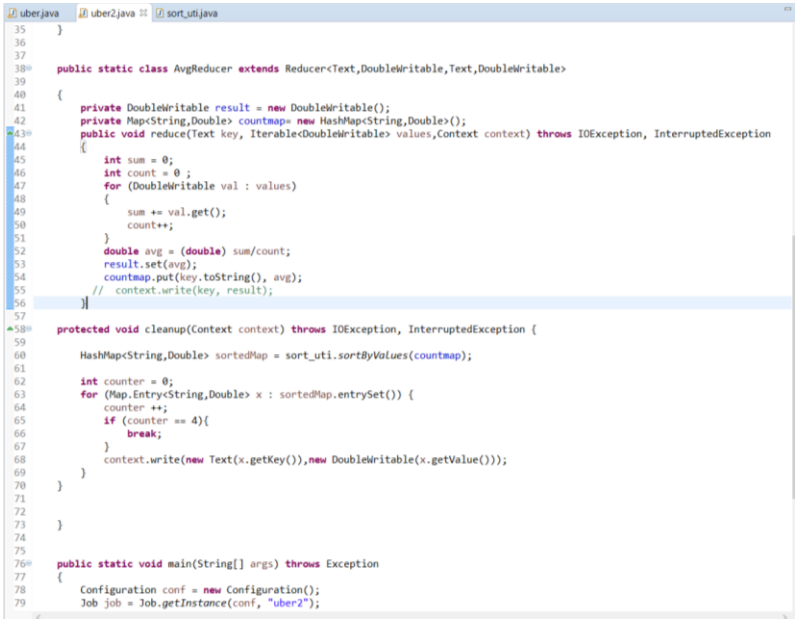
REDUCER:

It takes the list of values from framework. It sums up the trips that are taken by each basement and average is taken. The result is sent as an output.

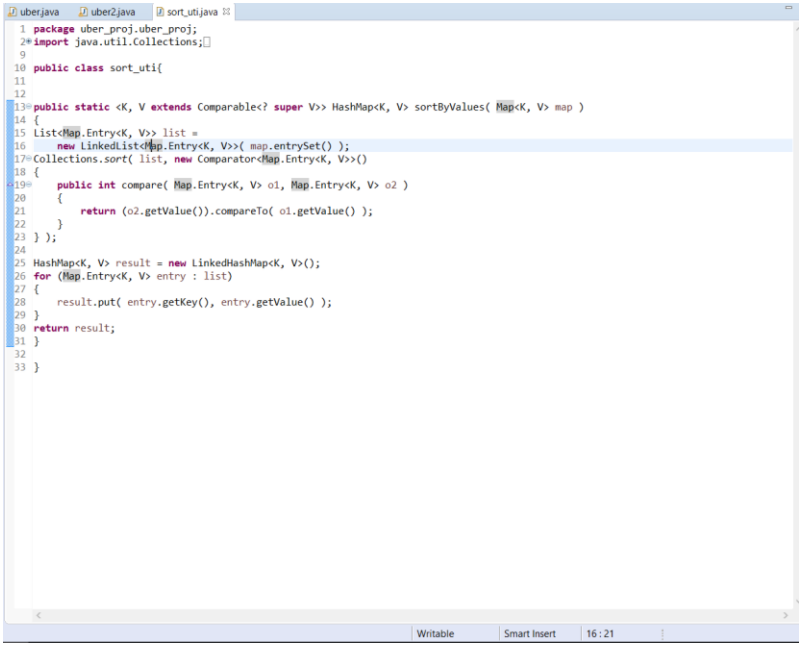
SORT PHASE:

After the completion of reducer phase the cleanup phase is done which takes care of the unwanted data and a connection is made to the sorting phase which basically use sort by value method in finding out the top three basements that has the most trips in the entire dataset.

SCREENSHOTS:



```
uber.java  uber2.java  sort_uti.java

    }


    public static class AvgReducer extends Reducer<Text,DoubleWritable,Text,DoubleWritable>

    {
        private DoubleWritable result = new DoubleWritable();
        private Map<String,Double> countmap= new HashMap<String,Double>();
        public void reduce(Text key, Iterable<DoubleWritable> values,Context context) throws IOException, InterruptedException
        {
            int sum = 0;
            int count = 0 ;
            for (DoubleWritable val : values)
            {
                sum += val.get();
                count++;
            }
            double avg = (double) sum/count;
            result.set(avg);
            countmap.put(key.toString(), avg);
          //  context.write(key, result);
        }

    protected void cleanup(Context context) throws IOException, InterruptedException {

        HashMap<String,Double> sortedMap = sort_uti.sortByValues(countmap);

        int counter = 0;
        for (Map.Entry<String,Double> x : sortedMap.entrySet()) {
            counter ++;
            if (counter == 4){
                break;
            }
            context.write(new Text(x.getKey()),new DoubleWritable(x.getValue()));
        }
    }


    }


    public static void main(String[] args) throws Exception
    {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "uber2");
```



```
uber.java  uber2.java  sort_uti.java

package uber_proj.uber_proj;
import java.util.Collections;

public class sort_uti{


public static <K, V extends Comparable<? super V>> HashMap<K, V> sortByValues( Map<K, V> map )
{
List<Map.Entry<K, V>> list =
    new LinkedList<Map.Entry<K, V>>( map.entrySet() );
Collections.sort( list, new Comparator<Map.Entry<K, V>>()
{
    public int compare( Map.Entry<K, V> o1, Map.Entry<K, V> o2 )
    {
        return (o2.getValue()).compareTo( o1.getValue() );
    }
} );

HashMap<K, V> result = new LinkedHashMap<K, V>();
for (Map.Entry<K, V> entry : list)
{
    result.put( entry.getKey(), entry.getValue() );
}
return result;
}

}

Writable  Smart Insert  16 : 21
```

OUTPUT:

```
[root@sandbox ~]# hdfs dfs -cat /uber_out_17/p*
B02764  32448.28813559322
B02617  12288.559322033898
B02682  11228.966101694916
```

INFERENCE:

The analysis of Uber dataset helped us to understand and study the traffic of an area surrounding the basements and it also enables to make better arrangements for faster and efficient routes. Also from the output of these problem statements it can be inferred that some basements activity on dates are high which can be related to a holiday weekend. Therefore, we can make better predictions for the new products or new modes of transportation which is further explained in the future implementation part that is given below.

FUTURE IMPLEMENTATION:

- The datasets can examine the efficiency of a cab and we can correlate that to cab ratings.
- Passenger frequency can be determined based on the Uber datasets on trips and hence further arrangements could be made to make cabs available.
- We can work on most demanding days for a route and hence bring in efficient number of cabs to basement.
- We have used terminal to output data. We can use a more streamlined GUI to call and arrive at conclusions from the data.
- On a further scale uber can implement these datasets to test run electric cabs and self-driven cabs.

CONCLUSION:

The reason why Uber is so promising is not merely because they threaten to undermine the existing order, but because they can reduce the scope of regulation. They can do this by solving the problems that markets have had in ensuring that information flows and this project is just an insight of how to use these very large data that they get to a much more effective solution and also much more effective way to proceed to their future. Further, and most importantly, they can foster the competition that is needed to get the most out of taxi markets.