Big Data Analytics Programming - Assignment 3

Milos Dragojevic - r0780142 - milos.dragojevic@student.kuleuven.be

1 Introduction

The goal of this assignment is to deliver a Spark implementation to compute San Francisco taxi trips length distribution. Further, the aim is to also create an efficient MapReduce application to reconstruct trips from segments and accurately compute airport taxi trips revenue.

2 Trip Length Distribution

Firstly, an algorithm to compute the distribution of trip lengths is implemented in Java. The same method is done in Spark.

The CPU time (user + sys) for the Java implementation is 1.1 seconds while for the Spark method it is 5 seconds. This was expected because the Java implementation executes the code in one stretch while the Spark method starts multiple processes to execute the Mapper, Combiner and Reducer. This requires additional time related to the overhead of the Spark framework which renders the implementation inefficient for a small task.

The MapReduce implementation uses a Mapper to extract from the trip records the coordinates of the start and end positions of the trips and calculate the distance in kilometres between these points. The distance between two points is calculated using the Spherical Earth projected to a plane distance[1]. The data is aggregated into bins with an interval parameter of 2 kilometres. For example, the bin 72 includes the count of trips with distances between 72.0 km and 73.99 km.

The Mapper receives for each trip as input a Text value and generates an output with a key-value pair of the following type `<Text,IntWritable>`. The key contains the ID of the bin corresponding to the calculated distance and the value is equal to 1.

Thereafter, a Combiner is used to sum up on each bin ID the list of all 1 values provided by the Mapper. For example, if there are four trips in bin 72, then the input of the Combiner is `<72,1>, <72,1>, <72,1>, <72,1>` and the output is `<72,4>`. Finally, the output of the Combiner is the input of the Reducer which performs the same task as the Combiner and outputs the final result.

Figure 1 clearly shows on a y log scale that the majority of trip lengths are short as shown by the peak for bins 0 to 6. Another peak is present around bin 18 which corresponds to the distance from the airport to the city centre.

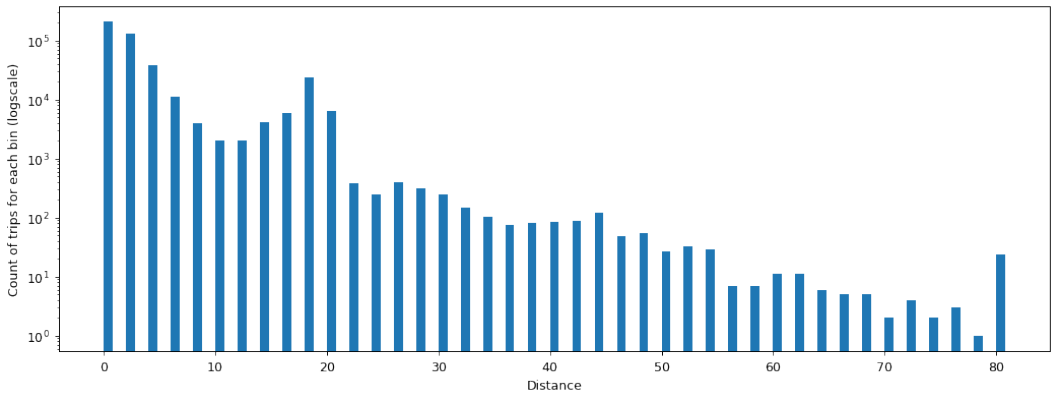

Figure 1: Trip length distribution on a y log scale. Note that bin 80 represents all trip distances bigger or equal to 80km.

3 Airport Ride Revenue

3.1 Identifying erroneous GPS points

To identify erroneous GPS points, trips were plotted using maps which showed that some points were located in impossible places such as in the ocean or even in Europe as the longitude or latitude were found to be close to 0°. Example of such impossible data points are visualised in figure 2. This allowed to determine the bounds of the zone of points to keep which is delimited by latitudes between 36.50°N and 39.80°N. With regards to the longitude bounds, the right limit of the area is 119.10°W and the left limit of the area is defined by the equation of a line adjacent to the coastline allowing to discard any point to the West of this line.

[1] `https://en.wikipedia.org/wiki/Geographical_distance`

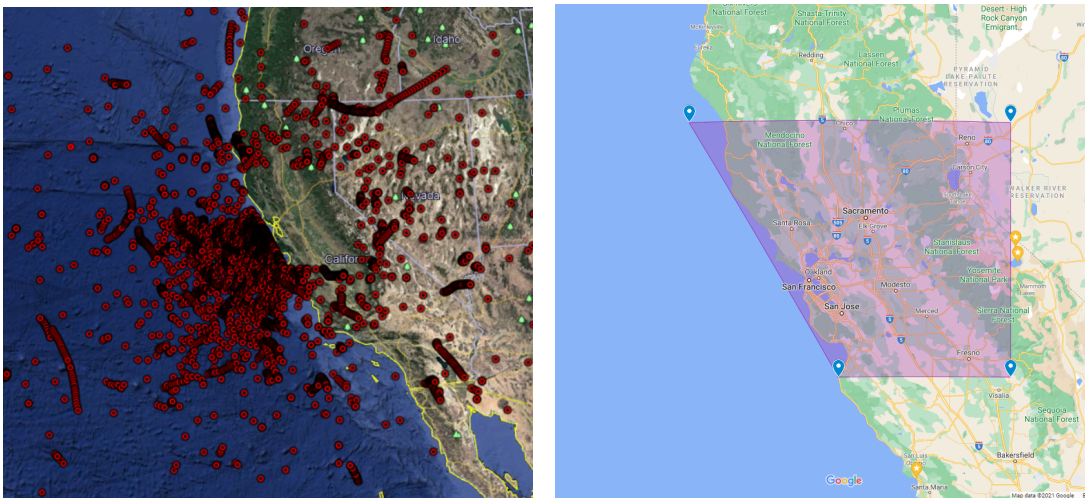

Figure 2: Example of impossible coordinate points in the data (left) and chosen valid trips area (right).

3.2 Reconstructing the trips

An analysis of the segment records has shown a few issues such as: records of one specific taxi are mixed up with other taxis records, some records are in the reverse chronological order, certain records contain invalid values (NULL), others have an empty status, huge time gaps (sometimes days) are present in M-M segments and finally some records overlap each other.

Looking at the full segments file shows that 55% of segments are E-E records which do not add any value to the calculation of the revenue as taxis would be empty in the segment, thus, these records can be dropped. Furthermore, based on the E-M and M-E transitions, the total count of trips in the dataset is approximately 11,192,000 trips. The analysis of segments further shows that for the M-M segments, 99.5% of the records have a duration of less than or equal to 210 seconds between the two positions defining a segment.

Based on the issues to solve and the analysis, it was decided that to reconstruct trips the best solution is to split each segment into two records containing each of the segment's positions. Thereafter, these records need to be sorted by taxi and in chronological order. Finally, based on the taxi status it is possible to calculate the total trip distance, to see if any trip position was in the vicinity of SFO Airport and to compute the trip revenue.

The solution was implemented with two MapReduce jobs, a first job to reconstruct the trips from the segments and a second to calculate the total daily revenue of airport trips from the reconstructed trips.

In the first job, the segment Mapper receives as input Text values such as the one below.

```
"450,'2008-05-25 09:16:58',37.61611,-122.38888,'M','2008-05-25 09:17:00',37.61506,-122.39206,'E'"
```

It discards segments with E-E transitions, checks the validity of both GPS positions and outputs `<Text,NullWritable>` key-value pairs similar to the following ones.

```
<"450,2008-05-25 09:16:58,37.61611,-122.38888,M",NullWritable>
<"450,2008-05-25 09:17:00,37.61506,-122.39206,E",NullWritable>
```

The segment Combiner processes the output of the segment Mapper by aggregating the records based on the key, effectively removing duplicates. Thereafter, a segment Partitioner is used to ensure that all the segments positions of a specific taxi are assigned to the same Reducer. The Partitioning is done based on the modulo division of the taxi number by the number of Reducers to ensure load balancing between the Reducers. After the segment Partitioner, the records are shuffled and sorted by key and provided as input to the segment Reducer which receives all the positions of one taxi in ascending order such as below.

```
<"450,2008-05-25 09:13:44,37.61661,-122.38425,E",NullWritable>
<"450,2008-05-25 09:14:32,37.61799,-122.38607,M",NullWritable>
<"450,2008-05-25 09:14:47,37.61798,-122.38606,M",NullWritable>
<"450,2008-05-25 09:16:01,37.61799,-122.38608,M",NullWritable>
<"450,2008-05-25 09:16:58,37.61611,-122.38888,M",NullWritable>
<"450,2008-05-25 09:17:00,37.61506,-122.39206,E",NullWritable>
```

Based on this sequence of points and the status change, it identifies the beginning and the end of each trip. It performs checks such as ensuring that the speed between two points is less than 180km/h, verify that the trip distance is greater or equal to 100 meters which is roughly the smallest distance that can be done at the airport to switch between two terminals. It also verifies if any of the trip positions is within the radius of SFO airport. It further checks if the time interval between M-M points for the same taxi is bigger than 210 seconds, if yes then it is concluded that a new trip has started for the same taxi. Finally, it calculates the total distance and total revenue of the trip. The Reducer only outputs key-value pairs for the airport trips like the following example.

<"450 1211706872.0 37.61799 -122.38607 1211707018.0 37.61611 -122.38888 true 0.327 4.06 2008-05-25"
,NullWritable>

Then the Record Writer finally writes in the output file the following string.

"450 1211706872.0 37.61799 -122.38607 1211707018.0 37.61611 -122.38888 true 0.327 4.06 2008-05-25"

The first job reconstructs 11,104,678 taxi trips which represents 99.22% of all trips based on the E-M or M-E transitions which indicates that the implemented method is efficient. The second MapReduce job processes the output of the previous job to calculate the revenue. The revenue Mapper retrieves the date and the revenue of each trip and outputs `<Text,DoubleWritable>` key-value pairs like `<"2008-05-25",4.06>` to a revenue Combiner which sums up the revenue by day. The Combiner output is shuffled and sorted by key. Thereafter, the revenue Reducer sums up the total revenue by day: `<"2008-05-25",14861.71>`. Finally, the Record Writer writes in the output file the following string: `"2008-05-25 14861.71"` representing the revenue of airport trips by day.

3.3 Solution efficiency and Revenue

As in Hadoop a gzip input file is only processed by a single Mapper, the first decision is to use the uncompressed format of the data to allow more Mappers to process it in parallel. The given cluster nodes have 4 processors and 7GB of RAM. To best use these resources, more than one Mapper or Reducer should run in parallel on each node. On the given cluster the largest container allowed by the parameter `yarn.nodemanager.resource.memory-mb` is 1536MB. For the first MapReduce job it is determined that setting `mapreduce.map.memory.mb` to 512 MB for Mappers and `mapreduce.reduce.memory.mb` to 768 MB for Reducers is sufficient. The ratio of heap-size to container-size had to be reduced using the parameter `mapreduce.job.heap.memory-mb.ratio` to a value of 0.48. These settings allow for the first job up to $1536/512 = 3$ Mappers or up to $1536/768 = 2$ Reducers to run on the same node. By benchmarking, the empirical formula found in equation 1 defines the best performing number of Mapper/Reducer containers.

$$Number\,of\,containers = Number\,of\,running\,nodes * \frac{Maximum\,container\,size}{Requested\,container\,size} - 2 \tag{1}$$

On an 8-node cluster, the best performance for the first job is achieved with $(8 * 1536/512) - 2 = 22$ Mappers and $(8 * 1536/768) - 2 = 14$ Reducers. As the number of Mappers is indirectly defined by the split size of the input file via the parameter `mapred.min.split.size`, the program is able to determine the appropriate split size based on the file system blocksize (128MB), the input file size and the number of Mappers needed.

For the second MapReduce job, no tuning was necessary as the number of Mappers is defined by the number of output files of the first job. There are 768MB allocated per Mapper and 1536MB for the single Reducer that compiles the daily revenue of airport trips.

On cluster 1 with 8 running nodes, the best benchmark of the implemented MapReduce program has a **total runtime of 230 seconds** distributed as follows: 203 seconds for the first job to reconstruct the trips, and 27 seconds for the second job to calculate the total revenue of airport trips. The total revenue of airport trips is **$23,279,324.24** and a plot of the revenue over time is provided in figure 3.

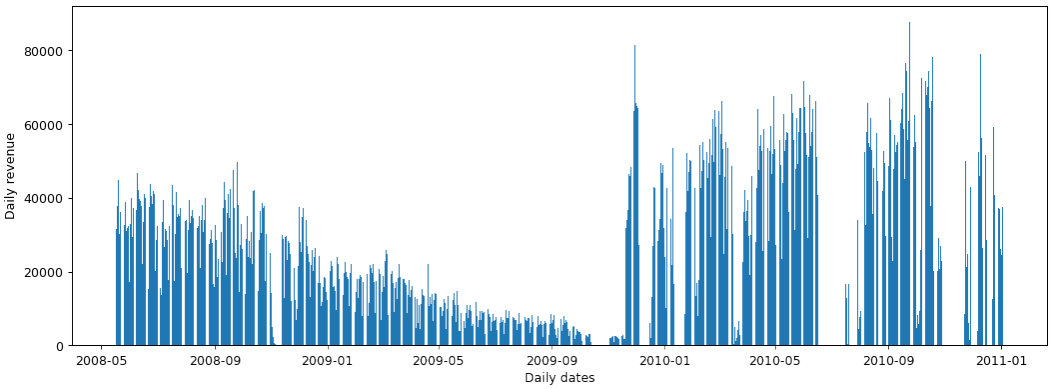

Figure 3: Revenue over time.

4 Conclusion

The implementation of MapReduce combined with sanity checks led to a short execution time and a precise revenue computation. Further research could focus on adding more sanity check such as adding a threshold for the minimum trip speed, maximum trip duration.