Report: San Francisco Taxi Network

Sarah Binta Alam Shoilee

August 2019

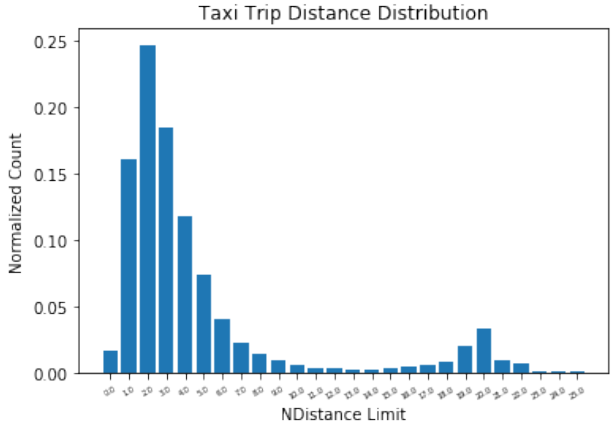
This experiment is dedicated to compare the performance between any standalone object-oriented program and Spark frame work. For this experiment, we are given with GPS tracks of taxis in San Francisco which consists trip records of a month from 2010. As a object orientated programming language, Java is chosen to implement the experiment. Due to unreliable data point from the GPS tracker, we need to make some assumption before distance calculation. Here, we only consider those trips that only start from San Francisco city and end in the same city. By doing so, we get rid of any data point that indicates impossible trips such as trip to the earth core. Then, we use the flat surface formula to count the distance of a potential trip. Due to the limitation of flat surface formula, it performs best up to 20km distance. To aid our experiment, we limit our distance to 25km. At the end, we count the bucket size and plot trip distribution. Since our distance is limited to 25 kilometers, we take 25 buckets to map each data to a bucket whose index is closer to the distance value. The algorithm for implementation is described as follow:

```
Result: Plot trip lengths
Read input file;
while InputFile is not empty do
    Parse each data field from input line;
    if Trip start location and end location resides in San Francisco City then
        Calculate distance;
        if Distance <25 then
            Add distance to the list;
        else
            Ignore;
        end
    else
        Ignore;
    end
    Calculate distance count;
    Round distance to closest integer bucket and bucket++;
end
Count each bucket size;
```

Algorithm 1: Calculate Trip Length Distribution

Next, we followed the same algorithmic logic, but using Spark framework. Upon initialization, input file data is distributed over the spark cluster. Each data will go through same location and speed as previous algorithm. We write a map function for location and speed check along with distance calculation. We mark every invalid data in this process. Using the Spark object filter, we get rid of the invalid data points. The map function only returns the distance as key. Then, map each distance to a ¡bucket, distance¿ pair by rounding the distance to it's nearest integer(bucket index). Then, we concatenate and sort all data based on bucket index(key). Next, we substitute the bucket counts with normalized count. Our distribution plot(Figure: 1) also support the fact of having limited bucket, as the bars are near zero(0) after 22km and higher frequencies with 8km range. Both of the above mentioned

Figure 1: San Francisco Taxi Trip Length Distribution over March, 2010



algorithm implements the same logic using two different approaches. In our test data, java implementation tends to execute in less time then spark method. After running few trials, we get the execution time ≈ 2.5 seconds for spark implementation and less than 1 second for object-oriented implementation. It is not surprising for the current case because the Java implementation simply read the file and then write operation directly from the file which supposed to execute in O(n) time and in this case n is not too big. On the other hand, distributed file processing uses parallel computation which takes some time in distributing file and also on concatenating the resulted files. DFS is more scalable and suitable for large amount of data.