Large scale Analytics on Taxi Rides to find the number of rides that can be aggregated

Akshaya Nagarajan
Software Engineering
San Jose State University
San Jose, California, United States
akshaya.nagarajan@sjsu.edu

Sivaranjani Kumar
Software Engineering
San Jose State University
San Jose, California, United States
sivaranjani.kumar@sjsu.edu

Pooja Patil
Software Engineering
San Jose State University
San Jose, California, United States
pooja.patil@sjsu.edu

Vignesh Kumar Thangarajan
Software Engineering
San Jose State University
San Jose, California, United States
vigneshkumar.thangarajan@sjsu.edu

Abstract**—** Taxi rides are one of the few problem spaces where we are still long way from optimization. To be more precise, we can think of carpooling options in all vendors like Uber, Lyft and even in the case of offline car sharing mechanisms, the efficient way of matching the similar rides is always a pain point. So, we have picked up this problem domain to study and to determine the number of rides that could have been combined. The impact of this study will shed light on the amount of fuel that could have been saved, the amount of pollution that could have been reduced and finally, the cost of rides that could have been minimized. We have picked New York city's yellow and green taxi rides data for four months, from May2018 to August 2018.

First step is to choose the platform between Amazon EMR with spark, Google Cloud DataProc with spark, HPC with Hadoop and Spark and Microsoft's Azure platform. We considered several parameters like cost, ease of implementation, toolkits available in all the above options.

The input size, data quality, calculation complexities are high. The data set is taken from New York City's yellow taxi and green taxi rides. The taxi rides are to be aggregated based on pickup location, pickup time and drop-off location. The output will be the percentage of taxi rides that can be pooled together. The condition for this is, to pool rides within five minutes with a pickup and drop-off locations within half a kilometer. The CSV files are converted to parquet format. Also new columns are introduced like Unix time stamp for calculating the time difference between any rides.

Over half a million rides were preprocessed using python script. The preprocessed data will be further processed by pyspark script in Google Cloud Platform. It is a difficult to process our data within reasonable time. We need to split the input files month wise.

Keywords— Big data, spark, Google cloud platform, Dataproc, Hadoop

I. INTRODUCTION

Big data Analytics examines the large amount of data to bring out the hidden patterns, useful insights and correlation from the data which can lead to operational improvement and new opportunities to better serve the world. Even though there are multiple tools available for analytical purpose, finding the optimum kind of tools for the right dataset is very important. We have been researching about lot of tools which best suited for the implementation of this project. Since the number of data to deal with is huge we have used GCP and used different input data formats, where we have incorporated KDD process for getting better insights.

II. DATA SET

A. Dataset Source

The data source has New York Taxi trip records. We chose ride data from May 2018 to August 2018. The

New York taxi data is already separated month-wise and tracked for Yellow Taxi, Green Taxi. An average file is of size 900 MB. The data is acquired in the format of CSV. The data is first preprocessed locally and then it is uploaded to google cloud data storage.

III. Problem definition- Rides Pooling

With surfeit of Taxi tracking data being present in the source dataset, pooling of taxi rides is based on following criteria:

a. Two rides can be pooled if the start distance is close to each other.

b. The start time of the two rides is nearly same.

c. The destination of the rides is near or same.

d. The total number of passengers is less than or equal to 5, so that they fit in a single taxi ride.

The rides to be pooled are checked if the distance between them is between 0.1 to 1 kilometer and time difference is between 1 to 10 minutes.

A. *Abbreviations and Acronyms*

GCP– Google Cloud Platform

SQL – Structured Query Language

HDFS – Hadoop Distributed File System

RDD – Resilient Distributed Dataset

IV. Data Preprocessing

As Data Preprocessing is an important step in any Data Mining task, the New York Taxi tracking data for Yellow taxi and Green taxi data's columns has a different naming convention. As part of Data Preprocessing all such columns were identified and a uniform naming format for all such columns is followed. For instance, the column with pickup and drop off date and time details will be named as 'Trip_Pickup_DateTime' for yellow taxi records and 'lpep_pickup_datetime' for green taxi records. Likewise, many other columns were identified. This will create a hitch while combining both the yellow and green taxi rides records.

In addition, all the pickup and drop location IDs in trip records are mapped to the taxi zones of New York city. The taxi zone details are given by the Zone ID and its corresponding latitude and longitude. Map the pickup and drop off location IDs to the taxi zone details data to obtain the pickup latitude, longitude and Drop location latitude. The pickup latitude, longitude and Drop location latitude, longitude columns are included additionally to the dataset. Additionally, all the empty fields are analyzed and removed.

Yellow Taxi	Green Taxi
tpep_pickup_datetime	lpep_pickup_datetime
tpep_dropoff_datetime	lpep_dropoff_datetime
Pickup_latitude	Start_latitude
Pickup_longitude	Start_longitude
Dropoff_latitude	End_latitude
Dropoff_longitude	End_longitude

Figure 1Comparision of columns between different vendors

V.Our Approach

Apache spark is one of the fastest and general-purpose cluster computing system for large scale data processing. We used this extremely in our project for aggregate function to calculate the count, sum and average. In addition to this, columnar format is used to efficiently read and write data from the storage as it speeds up the time it takes to return the query.

A. *Timestamp Difference*

Dates given in the data are not numbers, they are in date format. To compare them, we will need to convert the date format to number format for instance to seconds. To do that UNIX_TIMESTAMP function has been used to accomplish this. The calculation for the time difference will be from one minute or sixty seconds to the maximum of ten minutes or six hundred seconds.

```
dateFormat="MM/dd/YYYY"
df_with_pdt = df_with_id.withColumn("pdt",
from_unixtime(unix_timestamp(df_with_id["pickup_da
tetime"]),dateFormat))
df_with_ddt = df_with_pdt.withColumn("ddt",
from_unixtime(unix_timestamp(df_with_id["dropoff_d
atetime"]),dateFormat))
```

B. *Distance Difference*

The rides are consolidated mainly based on the difference in the distance. Rides which are started and ended close to each other will be picked. The data should contain the pickup and drop-off location details, distance will be calculated based on these two locations.

```
FROM rides t1 inner join rides t2
on t1.pdt=t2.pdt and t1.ddt=t2.ddt
and t1.id < t2.id
and (t1.passenger_count + t2.passenger_count) <= 5
t1.pickup_datetime, t2.pickup_datetime,
t1.Pickup_Latitude, t2.Pickup_Latitude,
(acos(sin(radians(t2.Pickup_Latitude)) *
sin(radians(t1.Pickup_Latitude)) +
cos(radians(t2.Pickup_Latitude)) *
cos(radians(t1.Pickup_Latitude)) *
cos(radians(t1.Pickup_longitude) -
radians(t2.Pickup_longitude))
)*6371.010) as dist
```

Distance between two locations should be calculated given the latitude and longitude, which is the aggregation function of different columns and also various trigonometric functions are used. As the data includes totally four columns and eight more trigonometric functions. The expression which are used are not the easiest when consolidating billion of records.

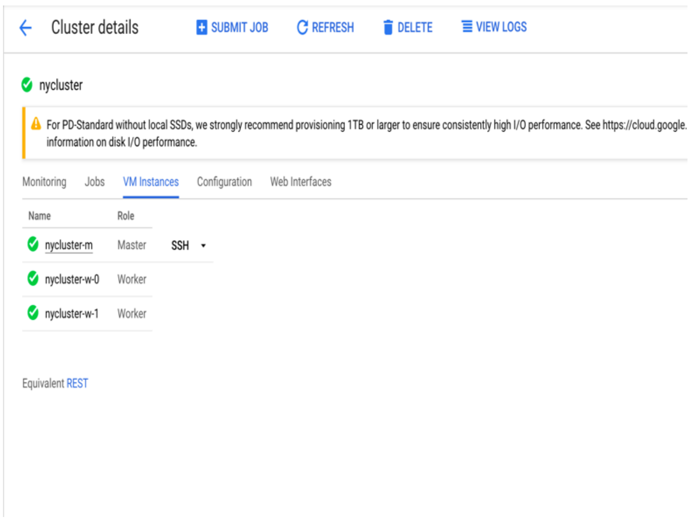
C. *Querying*

The data will be converted to the parquet format which is in turn loaded into spark data frame for further processing. Using self-join in the billions of records is a costly function. Because of this reason we need to take all the possible steps to reduce the time taken to optimize the query.

```
sqlContext = SQLContext(sc)
#df =
spark.read.csv("gs://prepped_data/green_tripdata_20
18-05_output.csv", header=True)
df1 = spark.read.csv("gs://prepped_data/small-
data/data.csv", header=True)
df1.write.parquet("gs://prepped_data/parquet_04")
parquetDF =
spark.read.parquet("gs://prepped_data/parquet_04/*.
parquet")
#parquetDF.show()
parquetDF.createOrReplaceTempView("rides")
sqlDf = spark.sql("SELECT * FROM rides")
df_with_id = sqlDf.withColumn("id",
monotonically_increasing_id()+1)
df_with_id =
df_with_id.na.drop(subset=["pickup_datetime"])
df_with_ddt.createOrReplaceTempView("rides")
```

D. *GCP and Configuration*

Google Cloud Platform is the cloud based big data platform with the help of this we can process large volume of data very quickly and can scale at cost efficient way. We will be using GCP in our project to process the billions of records. In GCP clusters gets launched in a minute. There is no need for us to do node provisioning, setting up the infrastructure, configuration of Hadoop or even tuning the cluster, GCP will take care of all these tasks for us so that we can peacefully focus on the analysis part. GCP pricing is cheap compared to other cloud providers. We can create 10-node cluster with the application for as little as $0.15 per hour.

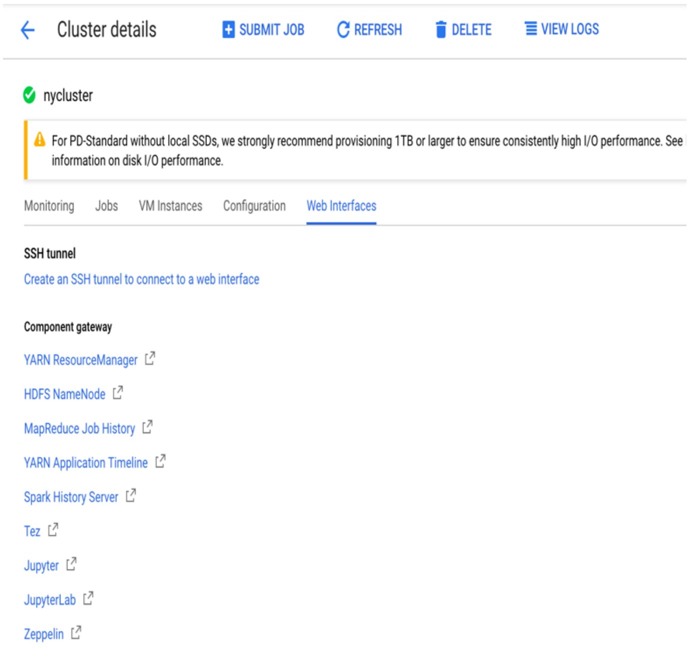


nycluster

For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consistently high I/O performance. See https://cloud.google.
information on disk I/O performance.

Name	Role	
nycluster-m	Master	SSH ▾
nycluster-w-0	Worker	
nycluster-w-1	Worker	

Figure 2. GCP Cluster configuration



nycluster

For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consistently high I/O performance. See
information on disk I/O performance.

SSH tunnel
Create an SSH tunnel to connect to a web interface

Component gateway
YARN ResourceManager
HDFS NameNode
MapReduce Job History
YARN Application Timeline
Spark History Server
Tez
Jupyter
JupyterLab
Zeppelin

Figure 3 Web interfaces for tools in GCP

E. *Auto Scaling*

Auto scaling is one of the important methods used in the cloud computing. It helps to maintain or monitor our applications and can even adjust automatically to all the changes. It is very easy to setup the application scaling which uses multiple resources from the different servers in less than a minute. It let us setup the target utilization for the resources for multiple levels. There will not be any need to navigate through console, because we can quickly view average utilization of all the scaled resources. Auto scaling is free to use in any cloud platform.

F. *Instances types*

Compute Engine Instances are the virtual machine which has been hosted on the Google cloud platform. We created instances using the GCP's console in dataproc cluster. We can add or delete instances in the console. While creating an instance inside project we have to define the zone, operating system and also the machine type of that instance. The additional storage space is available for an application running on the instances. The instance type of master node is n2-standard-2 (2-vCPU, 8.00GB memory). And the instance type of worker node is also n2-standard-2 (2-vCPU, 8.00GB memory) and each having 500 GB hard disk space.

VI. IMPLEMENTATION

There are different ways to achieve the aggregation of rides. Here we are going to see about the two approaches of aggregation. First method is, using SQL joins. The second method is using SQL window functions in spark context.

SQL JOINS

1. The collected data from yellow taxi rides and green taxi rides contains different columns name for pickup location and drop-off location. So, feature extraction and pre-processing techniques are applied to it. It is then uploaded to the google cloud storage for rides processing thereon.
2. There will be lot of csv files and it is nearly impossible to load these huge files into one single data frame. So, the processing will be done by month wise. This is the first assumption we are making. That is, the rides between months cannot be combined as they won't be present in a single data frame.
3. Each csv file is then loaded into a spark data frame. The loaded data frame is then written to parquet data format. Parquet format is a columnar storage format for the Apache Hadoop ecosystem. It provides efficient data compression techniques to handle complex data in bulk.
4. The parquet data is then loaded into spark data frame and a temporary table view is created for this data frame. Once, we have this table, we can run SQL queries on top of it.
5. The first thing we would want to do is to add a monotonously increasing ID as one of the columns. The reason being, we are combining multiple vendors' data here. So, we need to uniquely identify each row.
6. We are using preprocessed data, but still it is good practice to drop the rows containing NA values, especially for the columns we are interested. Here we drop the rows which are having NA values in 'pickup_datetime' column.
7. The pickup datetime and drop-off datetime is converted to unix time stamp format and two new columns are populated with that value, namely ddt and pdt respectively.
8. The rides can now be aggregated by running a little complex query of calculating the distance between the two rides is anywhere between 0.1 to 1 kilometer. This condition requires, eight trigonometric functions:
   *(acos(sin(radians(t2.Pickup_Latitude)) * sin(radians(t1.Pickup_Latitude)) + cos(radians(t2.Pickup_Latitude)) * cos(radians(t1.Pickup_Latitude)) * cos(radians(t1.Pickup_longitude) - radians(t2.Pickup_longitude)))*6371.010)*
9. The query is inner join with additional conditions where, t1.id < t2.id, t1.unixtimestamp - t2.unixtimestamp <= 300

and t1.passenger_count + t2.passenger_count <= 5.

10. Thus, this SQL join statement will be running for all our records and finally it shows the number rows which will give us the percentage reduction of rides when aggregated.

VII.WINDOW FUNCTION

1. The first seven steps are same as SQL Joins. We need to convert the input csv files to parquet format for better efficiency.
2. We introduced new column, 'group_ride_number' using window function like rank based on the pickup latitude, longitude and drop-off latitude and longitude.
3. We also introduced new column, 'lag_ride_number' using window function like lag based on the pickup latitude, longitude and drop-off latitude and longitude.
4. Now, that we have context of previous and current row, we can loop through the data frame in O(n) time complexity to identify the rows which satisfies all the four conditions we discussed above.

VII.CONCLUSION

Vehicles are leaving lot of carbon footprints all over the earth. We need to optimize the usage of all vehicles which are running on carbon fuels. Taxies are on such kind, where we have huge scope on optimizing the passenger travel especially within the city. Several taxi players like Uber, Lyft have introduced ride sharing options. However, these vendors can build knowledge on taxi booking pattern from past taxi rides by using the techniques we have discussed in this paper.

REFERENCES

[1] NYC taxi and limousine comission, TLC trip data, http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

[2] Import public NYC taxi and Uber trip data, GitHub repository, https://github.com/toddwschneider/nyc-taxi-data

[3]TLC Votes to Require Uber and Lyft to Disclose Trip Data, StreetBlog NYC, https://nyc.streetsblog.org/2017/02/02/tlc-votes-to-require-uber-and-lyft-to-disclose-trip-data/

[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, " The Google File System", Google, 2003

[5] Apache Hadoop Arcives, https://archive.apache.org/dist/hadoop/core/

[6] Chambers, Bill "Spark: The Definitive Guide. O'Reilly Media. "virtually all Spark code you run, where DataFrames or Datasets, compiles down to an RDD", 2017

[7] Saggi Neumann "Spark vs. Hadoop MapReduce", November 2014, https://www.xplenty.com/blog/apache-spark-vs-hadoop-mapreduce/

[8] @squarecog, Announcing Parquet 1.0: Columnar Storage for Hadoop, 2013, https://blog.twitter.com/engineering/en_us/a/2013/announcing-parquet-10- columnar-storage-for-hadoop.html

[9] Spark Configuration, Apache Foundation, https://spark.apache.org/docs/latest/configuration.htm

[10] Uber ride-hails to give NYC trip data after TLC passes amendment, February 2017, https://www.amny.com/transit/uber-ride-hails-to-give-nyc-trip-data-after- tlc-passes-amendment-1.13055471