BigData MapReduce Analysis – Most Revenue Generating NYC Taxi Zones

Udeshika W. Dissanayake
School of Science
RMIT University
Melbourne, Australia
s3400652@student.rmit.edu.au

***Abstract*—Publicly available BigData sets are becoming more and more popular with the advancement of IT & data infrastructure. Analyzing and exploring such BigData sets in order to obtain insights is becoming more and more popular with the recent developments in web services and their analysis frameworks. MapReduce is one of such powerful framework which unleash the potential of parallel computing to analyze petabytes of data. This work uses MapReduce computational framework to explore New York City Taxi and Limousine Trip Data to determine most revenue generating Taxi Zones. The results demonstrate that the developed algorithm can effectively be scaled on large datasets on commodity computing clusters.**

Keywords—MapReduce, BigData, AWS, Hadoop, EMR,

I. Introduction

New York City (NYC) Taxi and Limousine Commission (TLC) has been gathering and maintaining yellow and green taxi trip records that include pick-up and drop-off zones, date and time, taxi fare, and other auxiliary information like passenger count and trip distance [1]. These taxi records for last ten years are publicly available at NYC website [1] and at Amazon Web Services (AWS) Registry of Open Data [2]. A number of important insights on NYC taxi patterns, trends, and evolvements can be extracted by analyzing these large datasets. Simplified data processing approach on large computer processing clusters suites best for such detailed analysis of large datasets. MapReduce is one of such powerful yet simple parallel processing framework developed to perform simplified data processing on large processing clusters [3], [5]. Amongst a number of implementations of MapReduce programming framework, Apache Hadoop is a popular open source implementation [4], [6]. This study uses Apache Hadoop implementation of MapReduce algorithm on AWS Elastic Map Reduce (EMR) cluster commodities to investigate most revenue generating taxi zones in NYC. The developed implementation is run in different sized clusters and different sized datasets to analyze the performance and scalability.

II. Implementation of MapReduce Algorithm

Three different MapReduce algorithms were developed as part of this study. The first and the most simplest implementation out of all is "MapReduce without Combiner".

Algorithm 1. MapReduce **without** Combiner

```
1:  class Mapper
2:      method Map(docid a, doc d)
3:          for all text PULoc ε doc d do
4:              Emit(text PULoc, float fare)
5:  class Reducer
6:      method Reduce(text PULoc, float [fare1, fare2, ...])
7:          sum_fare ← 0
8:          for all float fare ε float [fare1, fare2, ...] do
9:              sum_fare ← sum_fare + fare
10:         Emit(text PULoc, float sum_fare)
```

Figure 1: Algorithm1 - MapReduce implementation without combiner

As can be seen in Figure *1*, each Map tasks emit the key/value pair of 'Pick-Up Location' and 'Taxi Fare' for each record. The Reduce tasks take in those key/value pairs and sum up and emit the total 'Taxi Fare' for each 'Pick-Up Location'. The taxi zone with highest total taxi fare is the most revenue generating taxi zone.

The second algorithm developed includes the intermediate step of local aggregation called Combiner – "MapReduce with Combiner" See *Figure 2*. The combiner performs local aggregation on each mapper output and passes its output to reducers. This approach significantly reduced the amount of data transfer between Mapper and Reduces nodes, hence increase the efficiency of the implementation for most scenarios.

Algorithm 2. MapReduce **with** Combiner

```
1:  class Mapper
2:      method Map(docid a, doc d)
3:          for all text PULoc ε doc d do
4:              Emit(text PULoc, float fare)
5:  class Combiner
6:      method Reduce(text PULoc, float [fare1, fare2, ...])
7:          sum_fare ← 0
8:          for all float fare ε float [fare1, fare2, ...] do
9:              sum_fare ← sum_fare + fare
10:         Emit(text PULoc, float sum_fare)
11: class Reducer
12:     method Reduce(text PULoc, float [fare1, fare2, ...])
13:         sum_fare ← 0
14:         for all float fare ε float [fare1, fare2, ...] do
15:             sum_fare ← sum_fare + fare
16:         Emit(text PULoc, float sum_fare)
```

Figure 2: Algorithm 2: MapReduce implementation with Combiner

The third algorithm implementation is "MapReduce with In-mapper Combiner" – see Figure *3*. An associative array is introduced in Mapper class and local aggregation is implemented inside the Mapper tasks. An additional two methods had been introduced in Mapper class in this implementation. No separate Combiner is required in this approach as the combining task is incorporated into the Mapper.

Algorithm 3. MapReduce **with** In-Mapper Combiner

```
1:  class Mapper
2:      method Setup()
3:          H ← new AssociativeArray
4:      method Map(docid a, doc d)
5:          for all text PULoc ε doc d do
6:              H{PULoc} ← H{PULoc} + fare
7:      method Cleanup()
8:          for all text PULoc ε H do
9:              Emit(text PULoc, float H{PULoc})
10: class Reducer
11:     method Reduce(text PULoc, float [fare1, fare2, ...])
12:         sum_fare ← 0
13:         for all float fare ε float [fare1, fare2, ...] do
14:             sum_fare ← sum_fare + fare
15:         Emit(text PULoc, float sum_fare)
```

Figure 3: Algorithm 3: MapReduce implementation with In-mapper Combiner

III. Experimental Results

Algorithms outlined in previous section were run on AWS EMR cluster for NYC yellow taxi data of first six months of 2019. The accumulative revenue over first six months in 2019 were observed for each taxi zone. The top ten revenue generating taxi zones are plotted in Figure *4*. As can be seen in Figure *5*, these top revenues generating taxi zones are either based in air-ports or at Manhattan town center.

Figure 4: Top ten revenue generating taxi zones in NYC – accumulative revenue over first six months in 2019

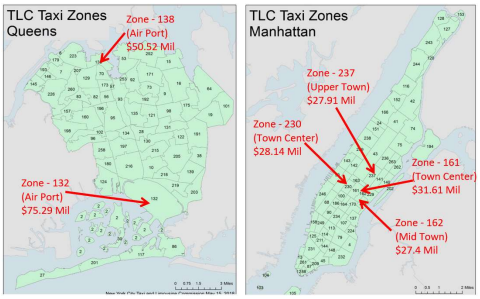

Figure 5: Geographical locations of top revenue generating NYC taxi zones: Air Port zones and Manhattan town centers shows the most revenue

Monthly growth in revenue of each taxi zones were calculated and shown in Figure *6*. It is clearly evident that months of February, March, and May are showing positive growth of revenue for most of the taxi zones. In contrast, months of April and June shows negative revenue growth for most of the taxi zones.

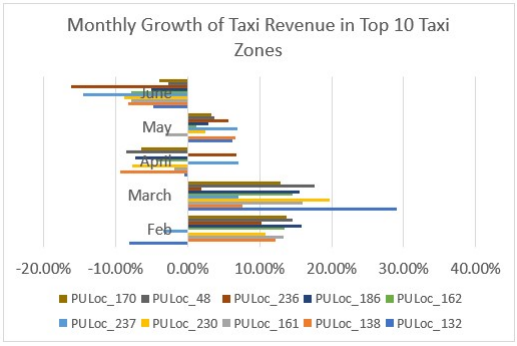

Figure 6: Monthly growth of taxi revenue at most revenue generating taxi zones - first six months in 2019

In order to demonstrate that the developed MapReduce algorithm is capable of handling larger input files and to check the behavior of the performance over input file size, a experiment was conducted on two node cluster for different number of input files. In general, each of the NYC TLC yellow taxi data is ~650MB in size. The number of input files has been changed from 1 to 6 (i.e ~650MB to ~4GB in 650MB steps) and the total running time was recorded. The running time was plotted against the file size and shown in Figure *7*.

To measure the scaling character in Hadoop cluster and to estimate the speedup character in the MapReduce algorithm, the experiment was run on different size (vary the number of nodes from 2 to 10) Hadoop EMR clusters. The running time was plotted against the cluster size as can be shown in Figure *8*.

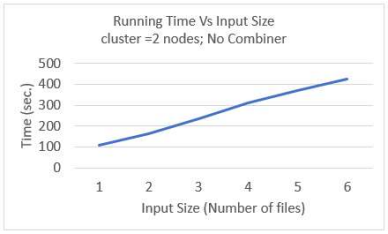

Figure 7:Running Time (sec) against Input file size. Each file is ~650MB.

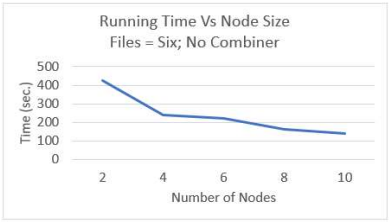

Figure 8: Running Time (sec) against different cluster size.

Finally, the performance comparison of the three MapReduce algorithms was conducted on two node cluster for six files (i.e. total of ~4GB data). As can be seen in Figure *9*, total running time is recorded for three algorithms. It is clearly evident from Figure *9*, that the implementation under Algorithm 2 - "MapReduce with Combiner", has 8% performance improvement compared to the standard implementation. Additionally, the implementation under Algorithm 3 – "MapReduce with In-mapper Combiner", shows more than 25% performance improvement when compare with standard implementation without combiner.

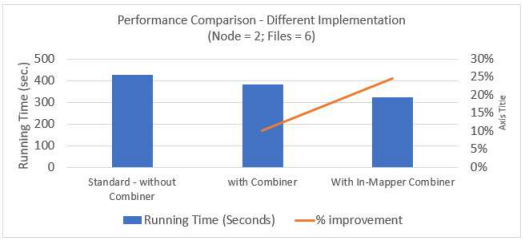

Figure 9: Performance comparison of three MapReduce algorithms.

A various different data analysis studies have been performed on NYC TLC dataset using different BigData tools [7]-[9]. Similar results on those publications shows similar and comparable results on most busiest and popular taxi areas in NYC [7]-[9].

IV. Conclution

Hadoop MapReduce framework was used to implement three MapReduce algorithms on AWS EMR platform and used that to analyze NYC TLC Taxi data. Top ten revenue generating NYC taxi zones were identified and their monthly revenue growths were investigated. The performance comparison of each algorithm on different size clusters was also analyzed.

REFERENCES

[1] City of New York, "TLC Trip Record Data," Taxi and Limoucine Commission, 2019, Accessed on: Sep, 20, 2019. [Online]. Available: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[2] AWS, "Registry of Open Data on AWS", New York City Taxi and Limousine Commission (TLC) Trip Record Data, Accessed on: Sep, 20, 2019. [Online]. Available: https://registry.opendata.aws/nyc-tlc-trip-records-pds/

[3] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, pages 137-150, 2004.

[4] Hadoop, "MapReduce", Hadoop 1.2.1 Documentation, 2019. Accessed on: Sep, 20, 2019. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/

[5] "MapReduce," https://en.wikipedia.org/wiki/MapReduce [Accessed Sep 20, 2019]

[6] "Hadoop," https://en.wikipedia.org/wiki/Apache_Hadoop [Accessed Sep 20, 2019]

[7] U. Patel. NYC Taxi Trip and Fare Data Analytics using BigData. In *10.13140/RG.2.1.3511.0485*, 2015.

[8] ArcGIS API for Python, "Analyzing New York City taxi data using big data tools", 2019, Accessed on: Sep, 20, 2019. [Online]. Available: https://developers.arcgis.com/python/sample-notebooks/analyze-new-york-city-taxi-data/

[9] An Explorer of Things, "Analyze the NYC Taxi Data", 2018, Accessed on: Sep, 20, 2019. [Online]. Available: https://chih-ling-hsu.github.io/2018/05/14/NYC

V. APPENDIX

Below are the derived output result tables from MapReduce algorithms:

	Percentage Growth				
NYC Taxi Zone	Feb	March	April	May	June
PULoc_132	-8.05%	28.99%	-0.45%	6.21%	-4.78%
PULoc_138	12.27%	7.64%	-9.39%	6.63%	-8.25%
PULoc_161	13.32%	15.94%	-1.89%	-3.15%	-7.78%
PULoc_230	10.78%	19.68%	-7.75%	2.46%	-8.80%
PULoc_237	-3.39%	7.01%	6.99%	6.94%	-14.50%
PULoc_162	13.47%	14.61%	-4.38%	1.25%	-7.82%
PULoc_186	15.76%	15.49%	-7.32%	2.85%	-5.12%
PULoc_236	10.29%	1.91%	6.81%	5.71%	-16.22%
PULoc_48	14.60%	17.62%	-8.53%	3.78%	-2.67%
PULoc_170	13.76%	12.90%	-6.43%	3.31%	-3.94%

Table 1: Calculated percentage growth of NYC Taxi revenue for most revenue generating taxi zones in first half of year 2019.

	$ Million - Accumilative					
NYC Taxi Zone	After Jan	After Feb	After March	After April	After May	After June
PULoc_132	11.18	21.46	34.72	47.92	61.94	75.29
PULoc_138	7.58	16.09	25.25	33.55	42.4	50.52
PULoc_161	4.43	9.45	15.27	20.98	26.51	31.61
PULoc_230	3.99	8.41	13.7	18.58	23.58	28.14
PULoc_237	4.43	8.71	13.29	18.19	23.43	27.91
PULoc_162	3.86	8.24	13.26	18.06	22.92	27.4
PULoc_186	3.68	7.94	12.86	17.42	22.11	26.56
PULoc_236	3.79	7.97	12.23	16.78	21.59	25.62
PULoc_48	3.22	6.91	11.25	15.22	19.34	23.35
PULoc_170	3.27	6.99	11.19	15.12	19.18	23.08

Table 4: Cumulative revenue generated in NYC's most revenue generating taxi zones in first half of year 2019.

Below are some additional figures plotted using the obtained results from performed MapReduce algorithms:

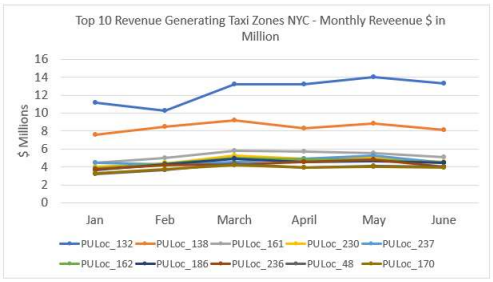

Table 2: Cumulative revenue generated in NYC's most revenue generating taxi zones for first half of year 2019

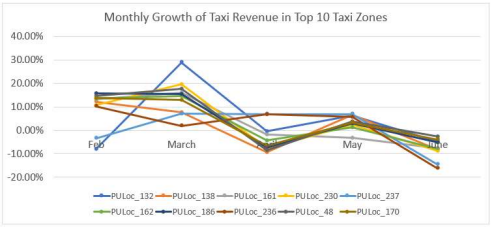

Table 3: Monthly percentage growth of revenue generated in most revenue generating taxi zones in the first half of year 2019