Distributed Data Analytics Framework for Cluster Analysis of Parking Violation

Evan Liu, Xi Yang, Nan Lin, Fiorella Tenorio, Paul Intrevado, Diane Myung-kyung Woodbridge
{eliu16,xyang68,nlin7,ltenoriocubas,pintrevado,dwoodbridge}@usfca.edu
Data Science Program
University of San Francisco

***Abstract*—In this research, we employed distributed systems to explore the similarities in parking ticket records using unsupervised machine learning techniques on a large dataset. Using 37 million ticket records (9 GB) collected by the New York City Department of Finance, we applied an algorithm to cluster existing tickets and dive deeper to find the distribution of precincts within different clusters. Amazon Web Services including S3, EC2 and EMR, and tools like MongoDB and Apache Spark were used in this endeavor. In this study, computational time and cost for different EMR settings were evaluated. We conclude that there are significant computational advantages to using distributed systems when implementing unsupervised learning on a large dataset as well as storing and managing data. We also observed that it is time efficient for a cluster with more workers instead of fewer workers with large memory space for the utilized data set. However we observed a trade-off between the execution time and the total cost for the cluster configuration.**

***Index Terms*—Distributed computing, Distributed databases Distributed information systems, Machine learning, Transportation**

I. Introduction

In the fast growing cities, parking has been one of the major challenges for the citizens and law enforcement agencies. A recent study showed that citizens in the United States spend 17 hours on average yearly searching for parking spaces resulting in \$20 billion for parking fees and \$345 per driver in wasted time and fuel [1]. Limited parking availability also causes parking violations which are huge financial burdens for the citizens. For the last ten years New York City grew the revenues from fines by 35 percent, raising \$993 million mostly (55%) from parking violations in 2016 [2]. Unfortunately, many cannot afford the fine which causes huge uncollected court debt, while the local government has to waste effort and labor. For instance, New York City had \$600 million uncollected parking violation fees in 2016 [3].

As parking system improvement is closely related to congestion, air pollution, waste of fuel and accidents, there have been many studies analyzing parking behavior and proposing a new parking system utilizing Internet of Things (IoT). Khana and Anand's recent work presented an IoT-based parking system that informs the user available parking slots using passive infrared (PIR) and ultrasonic sensors installed on the parking spaces [4]. Stenneth et al. utilized GPS and accelerometer sensors embedded in a smartphone to detect locations of a vehicle and a pay booth and analyze the driver's behavior to provide timely alerts [5]. Understanding a driver's parking behaviors using IoT sensors can benefit to develop a dynamic pricing model based on supply and demand. A study by Holguin-Veras, et al. showed that New York and New Jerseys Time of Day Pricing Initiative impacted 7.4% of trips, yielding behavioral changes including parking facility usage, time of travel, payment type, etc. [6].

For improving a real-time parking behavior monitoring, analysis and recommendation, many companies are investing for commercializing and deploying IoT devices and infrastructure. Verizon recently announced 40% growth in IoT network connections for transportation systems in 2017 [7]. Developing a robust pipeline to stream, store and analyze real-time data is a critical piece of infrastructure for IoT industries. This will potentially benefit to improve the effectiveness of parking enforcement patrol by optimizing routing plans [8]. For storing, managing and processing real-time sensor data from multiple sources including drivers, parking enforcement officers, parking meters and sensors on the vehicles and roads, it is critical to develop a scalable system that can store, manage and process data [9]. Unfortunately, relational databases, which are most widely used for storing and managing data, does not offer native mechanisms for data redundancy and availability in case of a database failure or scalability beyond a single server. Developing a machine learning model using big data on a single machine requires long execution time and often fail to provide a real-time feedback. While high-performance computing (HPC) or distributed computing showed improved execution time, HPC is more susceptible to failure and necessitates higher expenses [10].

Understanding aforementioned issues, the purpose of this paper is to store parking ticket data generated by New York City (NYC) and to analyze data to find, if present, patterns about the vehicles being ticketed using a scalable data pipeline. In this paper, the authors try to cluster parking tickets and find common patterns between the ticketed vehicles and determine if there are certain precincts in NYC where specific vehicles are more prone to being ticketed, on a given day, and maybe even at a given time using unsupervised machine learning on distributed computing. The paper focuses performance, time efficiency and expenses of the applied algorithms and compares the results with various deployment settings.

II. Background

A. NoSQL and MongoDB

In the last two decades, tech companies started tracking detailed user behaviors through websites and IoT devices in real-time, which caused a huge volume of data with an evolving schema. There are 2.5 quintillion bytes of data created everyday and the schema of tables in a relational database has to be changed often to store unstructured data or data with frequent schema changes [11]. For storing data with explosive volume growth, the needs of an affordable but robust system arose. In late 2000, many research and open source projects including Google BigTable [12] and Amazon Dynamo [13] demonstrated great performance and scalability using newly developed non-relational databases, NoSQL (Not Only SQL). Many of the new database management systems support distributed data sources by dividing and storing data in different servers (shards) and improve data availability by maintaining replicas in multiple servers. In addition, many NoSQL supports storing schemaless data and is designed to store data which are closely related as an aggregate in the same server node.

MongoDB, one of the most popular NoSQL databases stores data in schemaless JSON document format allowing users to add and remove fields in a document easily. MongoDB is designed to scale out and split up data across multiple servers. MongoDB takes care of loading data across a cluster, balancing data distribution in multiple servers and routing user requests to the right server which has the data. This enables users to focus on programming rather than low level system architecture and data distributions. In addition, MongoDB supports indexing for improving query performance and aggregation pipelines to design a complex pipeline by combining multiple simple queries sequentially [14].

B. MapReduce and Apache Spark

Hadoop's MapReduce, introduced in 2004, implemented efficient distributed techniques in an attempt to speed up large scale data analysis [15]. MapReduce splits data into smaller chunks across different nodes, and subsequently maps and processes a task, e.g., filtering and sorting, in parallel. The output of a mapped task becomes the input of a reduce operation, which performs a summary operation. This highly-effective model allows users to design programs with successive Map and Reduce operations, and is a popular and powerful programming paradigm.

Spark was designed in UC Berkeley's AMPLab in 2009 and open-sourced in 2010. Although Spark adopts MapReduce concepts, it runs up to 100 times faster than Hadoop MapReduce, utilizing in-memory computing and an advanced task-execution engine [16]. Spark also has built-in libraries that allow for efficient iterative computation. Included with version 0.8 in 2013, MLlib was the library within Spark to support Machine Learning features. The early version of MLlib APIs worked with Sparks native Resilient Distributed Datasets (RDD)—a fundamental data structure of Spark—and has since been placed into maintenance in favor of MLlib for DataFrames, a spreadsheet-like collection of data organized in columns and rows [17].

There is a paucity of academic research that benchmarks Spark MLlib performance, either evaluating the library itself under varying conditions. A number of researchers from DataBricks authored a 2016 paper providing an academic introduction to MLlib [18], and discussing its performance across versions. This research provides a similarly extensive look at performance, examining results generated when running Spark MLlib on a specific dataset [19], [20].

III. System Overview

A. System Workflow

In this research, we built a pipeline that is scalable to store and process parking violation data which could be potentially high frequency and high volume. Technologies and platforms were selected to build an unsupervised clustering model with high availability and fast performance. We therefore selected Amazon Web Services (AWS) as the primary platform to host storage, data extraction, transform and load (ETL) processes and machine learning tasks. The overview of data pipeline is noted in Figure 1.

1) Data Storage: Data collected from NYC OpenData [21] was stored in Amazon Web Service (AWS) Simple Storage Service (S3), a cloud storage service. As a potential application will ingest data from parking enforcement solutions and related data in real time, we made a decision to use cloud platforms that can handle a large volume of datastream. S3 provides replicated hosting in multiple data centers allowing high availability, as well as interoperability with other AWS components including Elastic Map Reduce (EMR) and Elastic Compute Cloud (EC2). In this study, we used four years of NYC parking tickets data, from 2015 to 2018 with an initial total volume of over 8 GB. After applying preprocessing steps in Section III-B1, we stored 3.2 GB data into S3.

2) Data Management: For a database, we launch a total of 13 EC2 instances, AWS cloud computing platforms, and installed MongoDB. Each instance represents mongos, a routing service node which takes user requests to a right instance which contains requested data, 3 configuration nodes with one primary (master) and two secondaries (slaves) which manages metadata of other nodes and the overall database and 3 shards which each includes one primary and two secondaries. Each shard contains data after preprocessing the original data in S3. Each primary node is in charge of read and write operations and copies data to secondaries. Secondary nodes maintains replicated data in case a master node fails due to networking, power outage and other system failures. Figure 2 describes MongoDB configuration in this study. MongoDB can take care of data balancing and routing from instances with different hardware specifications.

3) Data Analysis: AWS EMR uses Hadoop's YARN (Yet Another Resource Negotiator) as our cluster manager and automatically provisions hardware resources (EC2 instances) and installs the required software for running Apache Spark.

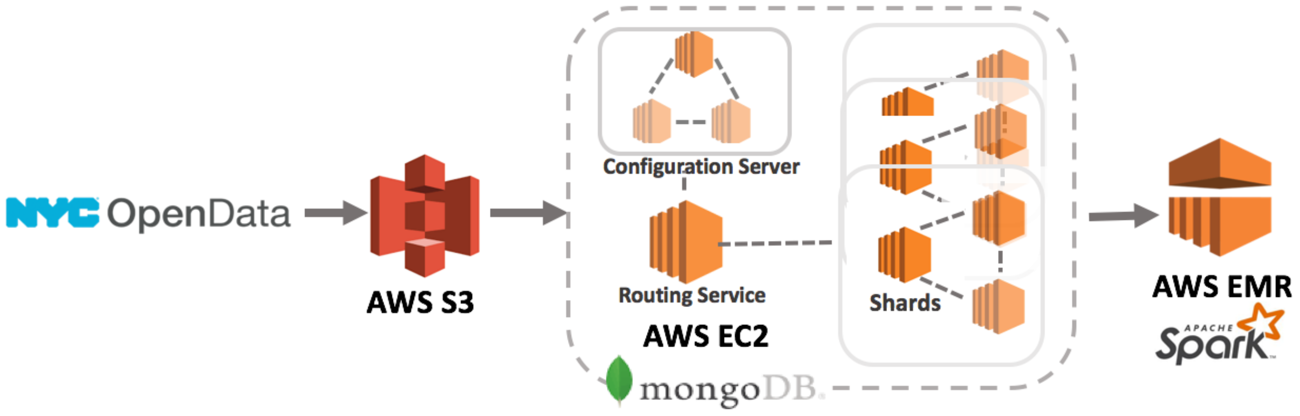


Fig. 1: System workflow

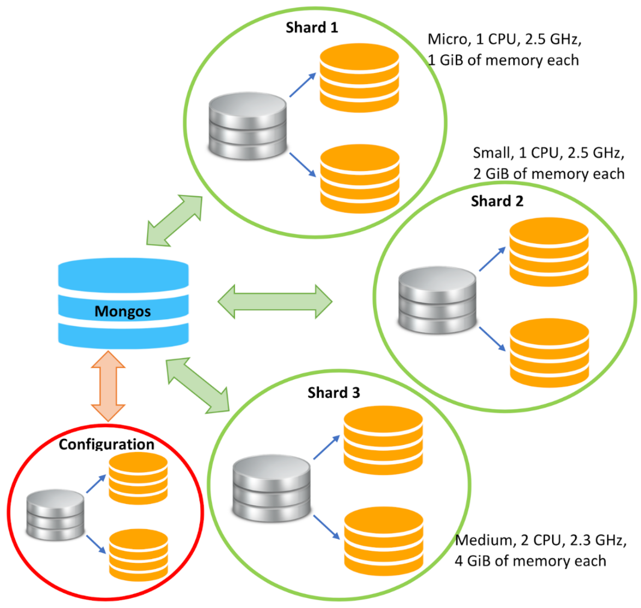

Shard 1
Micro, 1 CPU, 2.5 GHz, 1 GiB of memory each
Small, 1 CPU, 2.5 GHz, 2 GiB of memory each
Shard 2
Mongos
Shard 3
Configuration
Medium, 2 CPU, 2.3 GHz, 4 GiB of memory each

Fig. 2: MongoDB setting

Apache Spark comes with Spark SQL and Spark MLlib, which enable us to do distributed data preprocessing, feature engineering and modeling. In order to compare the efficiency of different clusters, we launched 4 types to clusters with different configurations. The settings and output will be shown in Table I.

TABLE I: Cluster Types and Specifications

	YARN 1	YARN 1 2	YARN 1 3	YARN 1 4
Type of EC2 instance(s)	M4.2xLarge	M4.xLarge	M4.xLarge	M4.xLarge
Number of instances	3	3	5	6
Number of worker nodes	2	2	4	5
Number of cores per worker	16	8	8	8
RAM per worker (GB)	32	16	16	16
Disk Storage per worker (GB)	32	16	16	16
Cost per hour ($)	0.36	0.18	0.3	0.36

B. *Algorithms*

As mentioned before, a question we wanted to answer is: given the time of day and a vehicle characteristics, in which precincts is that vehicle more prone to be ticketed?

To answer this, we begin by implementing an unsupervised machine learning algorithm, k-means clustering, in order to find similarities among observations (using its characteristics and time information) and group them together [22]. We will discuss the variables used in the clustering process in the Feature Engineering section. Once we trained our k-means model, each observation in our dataset will be assigned to a cluster.

Then, we map the precinct to its corresponding clustered observations. Once mapped, we calculate the distribution of tickets per precinct within each cluster. Finally, we can identify the precincts that a vehicle is more prone to be ticketed.

1) Data Preprocessing: As stated above, we will use features from two major aspects: vehicle-related features and timing. We got rid of columns from original dataset primarily due to three reasons:

- Columns with ID information (about the car, the person being ticketed, or the ID of the ticket itself).
- Features related to violation details, such as officer information and violation description, are removed since we want to focus on the vehicle characteristics and time.
- Location-related data since these variables won't be part of the clustering process.

At the end of this process, we end up with a data set of 3.2 GB.

2) Feature Engineering: At this point, each observation in our data set contains information about one ticket which includes the violation time and vehicle characteristics.

The first set of engineered features are vehicle-related features, such as vehicle color, vehicle make, vehicle body type, registration state and plate type. All of these variables are categorical and some has many distinct categories. Upon detailed analysis, we concluded that some were due to human error. Among those, categories with a small occurrence are categorized as ”OTHERS”, helping reduce the number of dimensions fed to one hot encoding.

The second feature set is violation time. To understand daily, weekly and other seasonal patterns better, we extracted hour (from 1 to 24), day of the week (from 1 to 7) and month (from 1 to 12) from the violation date.

Since k-means can only handle numeric data, we numerically encoded all categorical features. For converting categorical features to a numerical representation, we applied string indexer and one hot encoding algorithms. The string indexer algorithm converts categorical string values into integer indexes. One hot encoding expands a column to as many columns as distinct strings in the category and creates a sparse vectors where only one column contains a 1 and others are 0 [23]. After applying these two methods, we converted all of our features into a single vector filled with numerical values. By conducting this feature engineering process we were able to apply the algorithm to the data.

3) Machine Learning Model: k-means is one of the most popular and traditional clustering algorithms. It aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean distance. k-means performs the process by repeatedly assigning data points to initial centroids and updating centroids until convergence.
For m dimensional feature vectors, let us assume that each data point are $X_{(1)}, X_{(2)}, ..., X_{(n)}$, where we need to set a hyper-parameter k. There are the two steps to our approach:

Algorithm 1 K-Means Algorithm

```
Initialize cluster centroids $u_1, u_2, ..., u_k \in R^m$ randomly
while $u_j$ is not converged do
    foreach $i$ do
        $u^{(i)} := arg\ min_j ||x^{(i)} - \mu_j||^2$
    end
    foreach $j$ do
        $\mu_j := \frac{\sum_{i=1}^{m} \{c^{(i)}=j\} x^{(i)}}{\sum_{i=1}^{m} \{c^{(i)}=j\}}$
    end
end
```

To choose the hyper-parameter k, we chose an optimal value of k based on a scaled WSSSE metric. Then, we use the optimal k in different AWS EMR configurations so that we can compare the performance, time efficiency and expense between them. More detail can be found in the Experiment Output section.

IV. Experiment Output

A. Data

The data used for the experiments is the New York parking tickets data set from NYC OpenData which the time frame spans from the fiscal year of 2015 to 2018 [21]. After an initial cleaning of the raw data set, the size of the data is

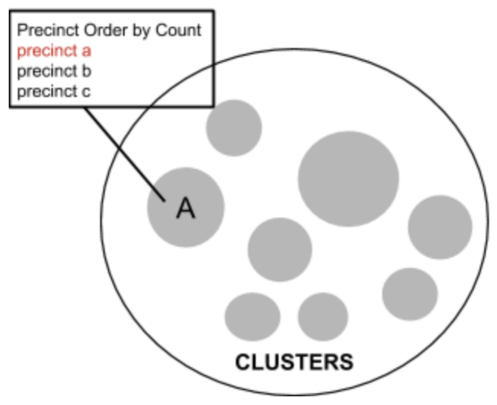


Fig. 3: Example of applying K-Mean: The algorithm clusters parking tickets based on the vehicle characteristics and time. Mapped to each clustered observation is the precincts that are prone to be ticketed

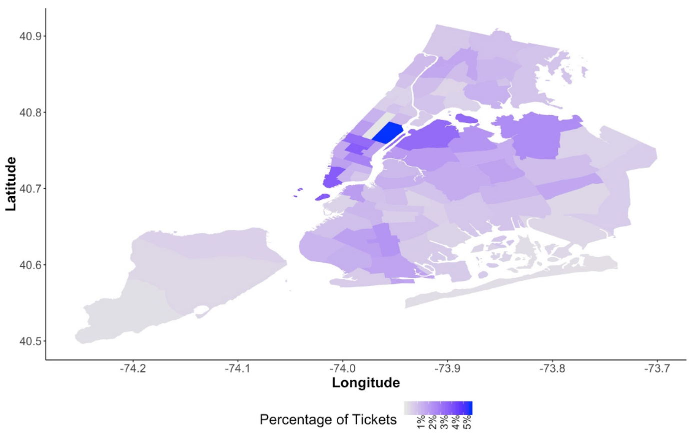


Fig. 4: Heat map of the number of tickets per precinct in 2018

3.2 GB which includes 36,566,368 observations. The data is stored in S3.

The features of the cleaned data set include the following.

- Plate Type: Passenger, Commercial, Taxi, Bus, etc.
- Registration State: The state where the vehicle was registered
- Vehicle Color : The color of the vehicle
- Vehicle Body Type: SUV, SEDAN, Truck, Trailer, etc.
- Issue Month: The month in which the ticket was issued
- Issue Quarter: The quarter in which the ticket was issued
- Issue Weekday: The day of the week in which the ticket was issued
- Violation Time: The hour of the day at which the ticket was issued

As an extension of our data we were able to create a heatmap for a years worth of parking tickets. The frequency of parking tickets within each precinct are shown in Figure 4. The darkest region of the heatmap shows the precinct with the most parking tickets while the lighter colored regions indicates

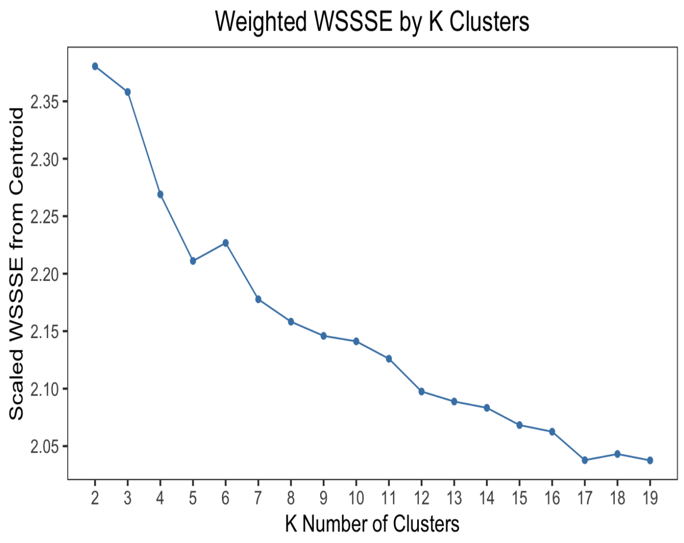


Fig. 5: Weighted WSSSE and k

TABLE II: Cost for building a model and average distances from centroids, when $k = 12$.

	YARN 1	YARN 2	YARN 3	YARN 4
Number of instances	3	3	5	6
Number of cores per worker	16	8	8	8
RAM per worker (GB)	32	16	16	16
Time per model (sec)	156.1064	244.8653	202.6197	138.2841
Cost per model (USD)	0.94	0.73	1.01	0.83
Distance from Centroid (scaled WSSSE)	2.1277	2.1210	2.1089	2.1136

the precincts that were not heavily ticketed.

B. Results

Given this is an unsupervised method, the information we can extract from the output is the clusters of parking tickets based the vehicle characteristics and time. Before applying k-means, we determined the right number of clusters by using a scaled Within Set Sum of Squared Errors (WSSSE). For evaluating models, WSSSE is used to measure an average distance from the center of a cluster.

$$D_k = \sum_{x_i \in C_k} \sum_{x_j \in C_k} ||x_i - x_j||^2 = 2n_k \sum_{x_i \in C_k} ||x_i - \mu_k||^2$$

$$WSSSE_k = \sum_{k=1}^{k} \frac{1}{2n_k} D_k$$

Since our WSSSE value was difficult to interpret due to the size, we decided to scale the distance by taking the square root of our WSSSE which was divided by the data size. With the scaled WSSSE metric, we plotted its relationship for each value of k clusters from 2 to 20. As a result 12 was chosen as the best value of k where WSSE improvement slows down (Figure 5).

We developed K-means models on 4 different EMR clusters shown in Table I and compared speed, performance and

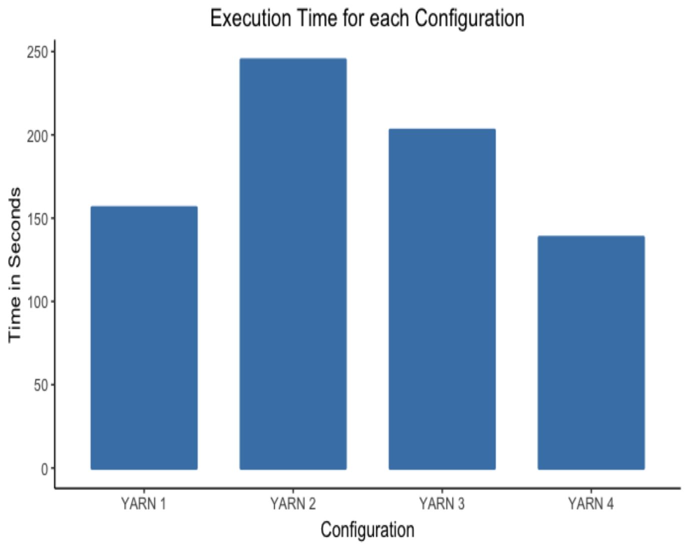


Fig. 6: Execution Time in Different YARN

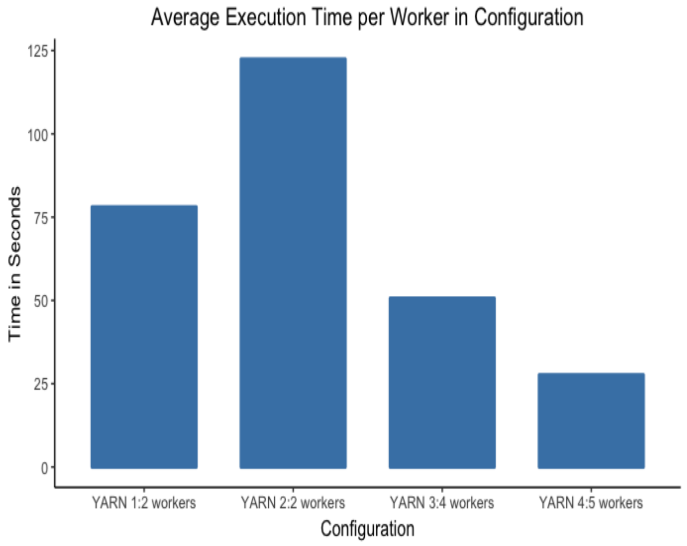
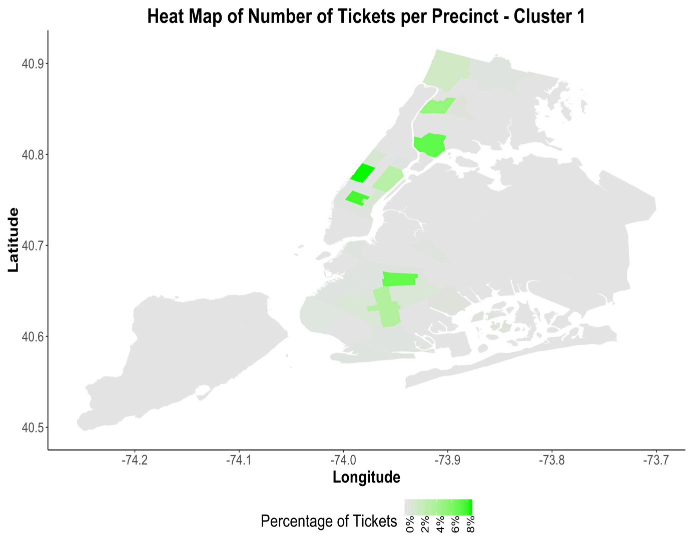


Fig. 7: Average Execution Time per Worker in Different YARN

expense as shown in Table II. As a general result, a large number of workers in a cluster performs faster. Figure 6 shows that the execution time for ”YARN 2”, with only 3 workers and lowest memory settings, is the slowest among them all, and the other 3 are very similar. We might infer that YARN 4 is faster than the rest, which has the most workers. To further dig out the reasons, we plotted average execution time per worker in Figure 7. Comparing the ”YARN 1” and ”YARN 2” in this graph, given the same number of workers, bigger memory in each worker has shorter execution time. We observed that when the configuration contains more workers the execution time is shortened by comparing ”YARN 2”, ”YARN 3” and ”YARN 4”. For now, we can conclude that ”YARN 4”, with the biggest number of workers, performs



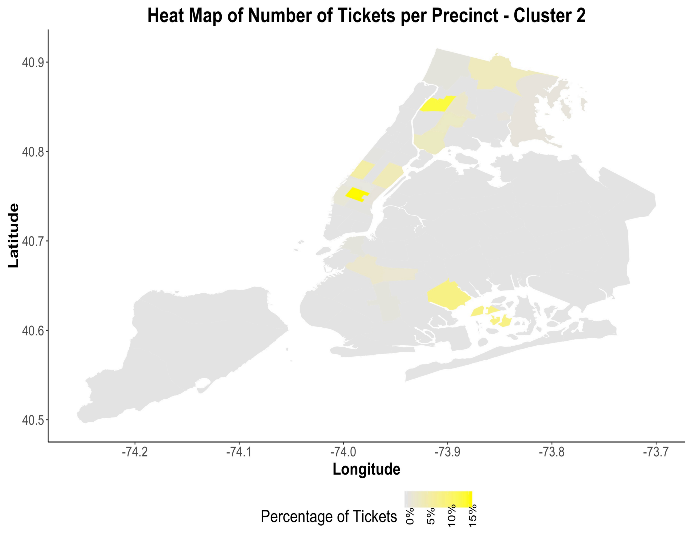


Fig. 8: Heatmaps on individual Clusters

faster than all comparative YARN configurations, because it reduces the time of each worker.

Another interesting fact we found in our experiment is that ”YARN 2”, with the longest execution time, actually has the lowest total cost as shown in Table II. Given this fact, the cluster with shortest execution time might not be a cost effective solution.

To grasp a better understanding of our clustering model, we imported our individual clusters into a heatmap as shown in Figures 8. From these mappings we were able to visualize the ticketed precincts and the frequency of being ticketed per each cluster. An interesting takeaway was that time square is more frequently ticketed probably due to having more traffic.

V. Conclusion

In this research, we applied k-means clustering to 37 million ticketed observations and compared the computational time and related cost by implementing different settings of AWS EMR clusters. We also visualized the results as distribution of tickets per precinct in different clusters to understand the patterns in locality of tickets.

We used 4 years of historical ticket information to generate our results. In our implementation, we were able to generate efficiency by piping data through AWS S3, MongoDB to Spark. Regarding the results, we observed that clusters with more workers are most time-efficient as it reduces the data size for each cluster to handle. However, there is a trade-off between the execution time and total price for the cluster. We should always have the best-balanced execution plan based on our speed requirement and budget.

In our future work, we will encompass the entire scope of parking tickets in a general setting with real-time information on all cars that may have parked in a particular region.

References

[1] P. Oldfield. (2017) Searching for parking costs americans $73 billion a year. INTRIX. [Online]. Available: http://inrix.com/press-releases/parking-pain-us/

[2] A. Kim. (2018) When cities rely on fines and fees, everybody loses. Governing. [Online]. Available: http://www.governing.com/columns/public-money/gov-court-fees-fines-debt.html

[3] M. Murphy. (2017) New york city fine revenues update. New York City Comptrollers Office, Bureau of Budget. [Online]. Available: https://comptroller.nyc.gov/reports/new-york-city-fine-revenues-update/

[4] A. Khanna and R. Anand, “Iot based smart parking system,” in *Internet of Things and Applications (IOTA), International Conference on*. IEEE, 2016, pp. 266–270.

[5] L. Stenneth, O. Wolfson, B. Xu, and S. Y. Philip, “Phonepark: Street parking using mobile phones,” in *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*. IEEE, 2012, pp. 278–279.

[6] J. Holguín-Veras, Q. Wang, N. Xu, and K. Ozbay, “The impacts of time of day pricing on car user behavior: findings from the port authority of new york and new jerseys initiative,” *Transportation*, vol. 38, no. 3, pp. 427–443, 2011.

[7] Verizon. (2017) State of the market: Internet of things 2017. [Online]. Available: http://www.verizon.com/about/sites/default/files/Verizon-2017-State-of-the-Market-IoT-Report.pdf

[8] C. Lei, Q. Zhang, and Y. Ouyang, “Planning of parking enforcement patrol considering drivers parking payment behavior,” *Transportation Research Part B: Methodological*, vol. 106, pp. 375–392, 2017.

[9] A. Howard, T. Lee, S. Mahar, P. Intrevado, and D. Woodbridge, “Distributed data analytics framework for smart transportation,” in *IEEE 16th International Conference on Smart City*. IEEE, 2018, pp. 1374–1380.

[10] D. M.-k. Woodbridge, A. T. Wilson, M. D. Rintoul, and R. H. Goldstein, “Time series discord detection in medical data using a parallel relational database,” in *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1420–1426.

[11] B. Marr. (2018) How much data do we create every day? Forbes. [Online]. Available: https://www.forbes.com/

[12] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon’s highly available key-value store,” in *ACM SIGOPS operating systems review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[14] MongoDB. (2018) Mongodb for giant ideas. [Online]. Available: https://www.mongodb.com/

[15] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[16] L. Gu and H. Li, “Memory or time: Performance evaluation for iterative operation on hadoop and spark,” in *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*. IEEE, 2013, pp. 721–727.

[17] Apache Spark. (2018) Ml pipelines. [Online]. Available: https://spark.apache.org/docs/2.2.0/ml-pipeline.html

[18] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, “Mllib: Machine learning in apache spark,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[19] J. Ma, A. Ovalle, and D. M.-k. Woodbridge, “Medhere: A smartwatch-based medication adherence monitoring system using machine learning and distributed computing,” in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2018, pp. 4945–4948.

[20] C. Dong, L. Du, F. Ji, Z. Song, Y. Zheng, A. Howard, P. Intrevado, and D. Woodbridge, “Forecasting smart meter energy usage using distributed systems and machine learning,” in *IEEE 16th International Conference on Smart City*. IEEE, 2018, pp. 1293–1298.

[21] NYC OpenData . (2019) Sort by parking violations issued - fiscal year 2015 - 2018 city government. NYC OpenData. [Online]. Available: https://data.cityofnewyork.us

[22] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, “Scalable k-means++,” *Proceedings of the VLDB Endowment*, vol. 5, no. 7, pp. 622–633, 2012.

[23] D. Harris and S. Harris, *Digital design and computer architecture*. Morgan Kaufmann, 2010.