Leveraging Distributed Computing Systems for Prediction and Analysis of San Francisco Fire Department Response Times

Xu Lian*, Sarah Melancon*, Jon-Ross Presta*, Adam Reevesman*,
Brian J. Spiering, Diane Myung-kyung Woodbridge
{xlian2,smelancon,jpresta,arreevesman,bspiering,dwoodbridge}@usfca.edu
Data Science Program
University of San Francisco

***Abstract*—Predicting the San Francisco Fire Department (SFFD) emergency response time is useful for both callers and the SFFD. In this paper, we leverage machine learning and distributed computing to attempt a solution to this problem. While driving time can be estimated using the Google Maps API, there are many more factors that affect the response time. We obtained a publicly available SFFD call history dataset, using BigQuery, containing information about the location of the station, the location of the incident, and the time the call was received. We combined features from this dataset with driving time estimates from the Google Maps Distance Matrix API.**

Our dataset was over a gigabyte, calling for a distributed framework to efficient training of the machine learning models. We fit Linear Regression, Decision Tree Regression, and Random Forest Regression models in the Apache Spark machine learning library (MLlib) on an Amazon Web Services (AWS) Elastic MapReduce cluster. We benchmarked training time for each model on different cluster sizes.

***Index Terms*—Distributed computing, Distributed information systems, Machine learning, Emergency services**

I. Introduction

According to the United States Census Bureau, San Francisco is the second-most densely populated large city in the United States, where over 880,000 residents live in 46.89 square miles [1]. The San Francisco Fire Department (SFFD) serves many critical roles for the residents and visitors by responding to fires, natural disasters, hazardous materials incidents, and emergency medical services (EMS). SFFD processes emergency calls for fire suppression in addition to prevention, dispatching the proper emergency units, providing guidance and instructions to callers. SFFD also responds to various natural disasters including earthquakes, tsunamis, and severe storms and flooding. The Department of Public Health, Paramedic Division merged with SFFD in 1997. Since then, firefighters have been trained and licensed by the San Francisco Emergency Medical Services Agency and responsible for both life-threatening and non-life-threatening EMS calls. The Division of Homeland Security (DHS), works within SFFD, to officiate chemical, biological, radiological, nuclear and explosion incidents in addition to terrorism threats in San Francisco [2].

Improving the system and changing infrastructure can reduce emergency response time, which in turn can bring crucial benefits including improving survival rates, preventing property loss, [3]. Emergency medical service (EMS) response time is a key measurement for prehospital system performance for preventing serious trauma or cardiac arrest [4]. In order to reduce response time, researchers and practitioners have proposed and implemented various plans including monitoring real-time traffic volume and direction, sharing real-time emergency fleet availability and location, and providing real-time route guidance for dispatched vehicles [5], [6].

However, it is also crucial to precisely predict arrival times in order to better manage resources and provide instructions, such as first aids and cardiopulmonary resuscitation (CPR), to a caller until dispatched emergency crews arrive.

Unfortunately, predicting precise arrival time of the emergency vehicles is challenging in major cities, especially San Francisco due to its population density, traffic, parking, limited resources, terrain, etc. In 2014, local media criticized SFFD since it did not dispatch enough ambulances and not respond in a timely manner [7]. Trowbridges recent study showed that urban areas and cities experience increased EMS response time and a higher probability of delayed ambulance arrival, due to longer trip distances, traffic congestion and trip time variability, and higher rates of traffic and pedestrian fatalities []. Between 2000 and 2013, the population increased by 7.8SFFD receives over 120,000 calls yearly which is mostly for EMS and fires, with aging equipment and limited resources including little over a total of 1,400 employees in 51 stations throughout the San Francisco County (Figure 1).

The goal of this project is to predict the time it will take for SFFD to arrive on the scene of an incident. This could help SFFD understand whether they are responding quickly enough. Furthermore, we can analyze the differences in response time in different areas or to different types of emergencies to better manage resources.

II. Background

In order to predict response times, it is critical to store, merge and analyze data affecting response times. Based on

* These authors contributed equally.

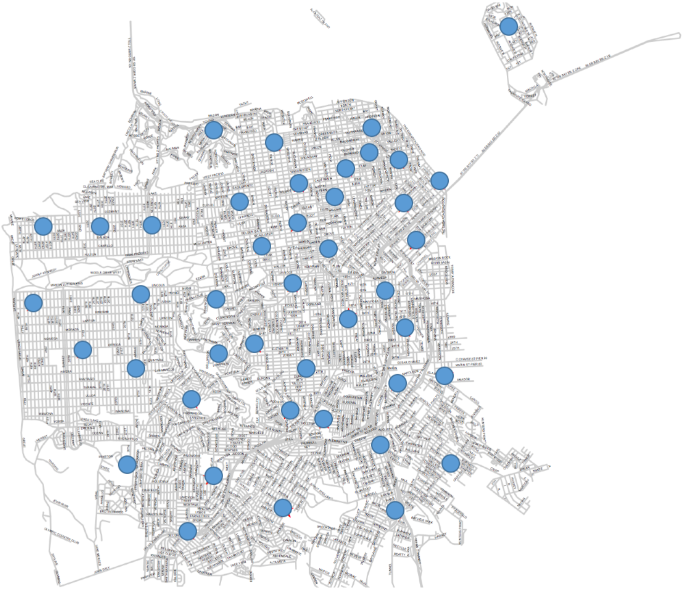

Fig. 1: San Francisco Fire Station Locations

Lam's recent study, traffic congestion is the main factor influencing the EMS response delay [8]. Storing real-time sensor data related to response time requires a large volume data storage. Unfortunately, depending on the data type and frequency, the data storage size may grow rapidly and unpredictably.

Cloud computing provides access computing resources to store and process data over the Internet hosted by cloud providers in their data centers. Cloud computing is highly scalable and allows for dynamically adding resources based on data processing and storage needs [9]. By sharing resources among users and centralizing maintenance services for hardware and software, cloud computing minimizes cost and resources for managing infrastructure. Cloud computing is a powerful tool for individuals and organizations to lower the cost of storing and processing a large volume of data [10], [11], [12]. Amazon Web Services (AWS) is the largest public cloud services providing various services including data storage, management, computing, analytics, etc. [13].

A. *Big Data and Scalable Data Storage*

Cloud storage is a type of cloud computing where its storage spans in multiple servers through multiple locations, but its virtualization makes its distribution seamless. Additionally, cloud storage provides high durability and availability by maintaining duplicated copies for their users [14]. AWS Simple Storage Services (S3) is a cloud storage system, adopting a pay-as-you-go charging model and offering effectually infinite storage capacity with high data durability, 99.99% availability, and low data access latency [15]. S3 also provides easy interoperability with other AWS services.

B. *Big Data and Scalable Data Processing*

Hadoop MapReduce is a programming paradigm for distributed computing introduced by Google. Hadoop MapReduce processes data in a key-value format and is composed of two steps, a map step and a reduce step. Once data is distributed to multiple machines, a map function, such as filtering, processes data in parallel and yielding an intermediate value. Outputs from a map function are redistributed so that outputs with the same key are located in the same machine. Then the data can be processed by a reduce function which is a summary operation. The output of a reduce function returns to a driver node and is passed to the client. This highly-effective model allows users to design programs with successive map and reduce operations.

Apache Spark is a variant model of Hadoop MapReduce, with improved performance and efficiency. Spark uses resilient distributed dataset (RDD), which is an abstract of a read-only collection of objects partitioned across a set of machines. RDD can be rebuilt if data is lost or a system fails and achieves fault tolerance using RDD lineage. RDD lineage tracks RDD dependency information including its parent RDD and operation used to create the current RDD in a directed acyclic graph (DAG) [16]. RDD lineages are used to efficiently group and schedule operations and to recover RDD in case of a system failure. Spark runs up to 100 times faster than Hadoop MapReduce because it uses in-memory computing and an optimized task-execution engine [17].

AWS Elastic Map Reduce (EMR) provides fully managed hosted Hadoop framework on top of Amazon Elastic Compute Cloud (EC2) [18]. EC2 is a computing environment in the cloud, where a user can configure the operating system, CPU, memory, disk, etc. [19]. Based on Iosups study, EC2 has, in general, better performance compared to similar modern commodity systems. The study also showed that EC2 is reasonably economical for scientific computing, providing slow efficiency degradation with the increased number of instances [20]. AWS EMR provides cluster maintenance and big data processing tools including Spark, Hadoop MapReduce, HBase, Flink, etc. and allows a user to easily configure and launch a cluster which are easily resizable [21].

III. System Overview

A. *System Workflow*

For this project, the data science pipeline was designed around scalability, availability, and efficiency using distributed computing on cloud resources. Technologies were selected to build an ingestion and prediction system for SFFD response time data. The overall system workflow is noted in Figure 2.

1) Data Sources: SFFD call history is publicly available in Google Big Query [22]. Google Big Query is a data warehouse that enables storage, management, and querying of data using Googles Cloud Services [23]. Section IV-A provides more details about the SFFD call history data, including the size and features.

Additionally, the authors also collected estimated travel time using from Google Maps Distance Matrix API [24]. The Distance Matrix API provides the distance and estimated arrival time, but does not provide historical data.

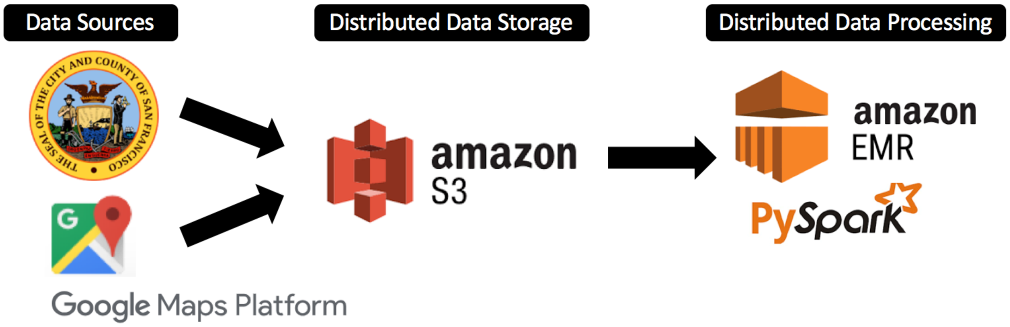

Fig. 2: System workflow

TABLE I: EMR Cluster Configurations

Cluster	Role	EC2 Type	Number of CPU	Memory
Cluster 1	Master	m4.large	2	8 GB
	Slave (Total 2 Nodes)	m4.large	2	8 GB
Cluster 2	Master	m4.xlarge	4	16 GB
	Slave (Total 2 Nodes)	m4.xlarge	4	16 GB

2) Data Storage: Data queried using Google Big Query and Map APIs was stored in AWS Simple Storage Service (S3). S3 was selected for its high scalability and durability for storing the relatively large volume of data.

3) Data Processing and Analysis: For feature engineering and machine learning, we developed a data pipeline that feeds data from S3 into EMR. The developed feature engineering and machine learning algorithms were written in PySpark, the Spark Python API and processed in parallel on EMR. In this study, the authors configured two cluster settings specified in Table I.

B. Data Preprocessing and Feature Engineering

We retrieved the fire station, address, call type, received timestamp, on scene timestamp, zipcode, and driving distance and duration for each call. The response time was defined as the received timestamp subtracted from the on scene timestamp. The hour of day and day of week that the call was received were extracted from the received timestamp.

If the station was not listed on the fire department's website, then the call was removed from the dataset. Calls were also removed if the distance between the fire station and address was not between .01 and two miles, if the response time was not between one and thirty minutes, or if the response time was more than twenty times larger than the estimated driving time. (Algorithm 1).

Before fitting machine learning models, missing values were imputed with “unknown.” The distance and duration variables were log transformed and the call type, zipcode, station area, hour of day and day of week were one hot encoded.

Algorithm 1 Feature Engineering

```
for Each SFFD calls do
    Retrieve station_area, address,
    call_type, received_timestamp,
    on_scene_timestamp, zipcode,
    response_time = on_scene - received,
    hour_of_day = hour(received_timestamp),
    day_of_week = day(received_timestamp),
    station_address = lookupAddress(station_area),
    distance, duration = googleMapAPI(station_address,
                                      address)
    if station_area in listed stations and
      $0.01\ miles \leq distance \leq 2\ miles\ and$
      $1\ minutes \leq response\_time \leq 30\ minute\ and$
      $response\_time \leq 20 \times duration$ then
        Include current call in dataset
    end
    Impute any empty values with ’unknown’
end
```

C. Machine Learning Algorithms

Apache Spark's machine learning (MLlib) library [25] supports a distributed implement of common ML algorithms. Distributing the machine learning training process across a cluster reduces training time and increases the ability to handle larger datasets. Since the goal of the project is predicting a numeric outcome (time), we fit a variety of regression algorithms. We choose to fit linear regression (with and without regularization), decision tree regression, and random forest regression.

1) Linear Regression: Linear regression estimates a best fit line for a numeric response based on a linear combination predictors. Given the distributed framework of Apache Spark, linear regression is optimized with stochastic gradient descent (SGD). Linear Regression serves as a baseline model for the current regression problem.

2) Linear Regression with Elastic Net Regularization: Adding regularization to linear regression is reasonable choice given the number of the number predictor variables compared

TABLE II: Number of Calls by Call Type

Call Type	**Cout**
Medical Incident	2,950,934
Structure Fire	605,663
Alarms	486,984
Traffic Collision	186,443
Other	73,508
Citizen Assist / Service Call	68,976
Outside Fire	53,177
Vehicle Fire	22,318
Water Rescue	21,721
Gas Leak (Natural and LP Gases)	16,889
Electrical Hazard	12,716
Odor (Strange / Unknown)	12,287
Elevator / Escalator Rescue	11,918

to the number of observations. Regularization will reduce the chance the over-fitting and increase the ability of the linear regression to generalize to unseen data. We choose Elastic Net Regularization because it linearly combines L1 (lasso) and L2 (ridge) regularization, thus mitigating the limitations of separately fitting L1 and L2 regularization. [26].

3) Decision Tree Regression: A decision tree is a supervised machine learning technique greedy algorithm that performs a recursive binary partitioning of the feature space into homogeneous regions, in case of regression the homogeneous regions are based on the scalar prediction value [27]. Decision trees are a popular technique because they are easy to interpret, can capture non-linearities, and can model feature interactions. Apache Spark's implementation partitions data by rows, allowing for distributed training [28].

4) Random Forests Regression: A random forest is an ensemble of decision trees. Each individual tree is trained on a bootstrapped subset of the data and each binary split is optimized on a random subsample of possible features. This inherent randomness between and within the decision trees avoids over-fitting issues common with decision trees [29]. Given the estimating of each bootstrapped decision tree is independent, the algorithm is especially well-suited to a distributed framework. We fit the regression version of random forests to estimate the scalar predictor of time. We use Apache Spark's default hyperparameter with no tuning.

IV. Experiment Output

A. Data

The San Francisco Fire Department (SFFD) stores their call history publicly available in Google's Big Query Cloud service. The earliest recorded calls are from April 2000 and the database is updated with information up to the present time. We queried from the database exactly one time, on January 24, 2019; any subsequent record was not represented in our dataset. This span of time consisted of 4,557,045 unique calls. The data included (but not always) the type of call, the timestamp that the call was received, the timestamp that the unit was dispatched, the timestamp that the unit arrived on the scene, the address of the incident, and the battalion to which the incident was assigned. Table II shows a breakdown of incident counts by call type.

TABLE III: Data sets used for experiment

Data Set	**Features**
small	distance, duration
big	distance, duration , call_type, zipcode, station_area, received_hour, day of the week

In addition to the data that we received from the SFFD, we took advantage of the Google Map's Distance Matrix API.

Our target variable was the amount of time elapsed between the time the call was received and the time the firefighters arrived at the scene of the incident. We chose not to include any intermediate time information (like when the unit left the fire station) because the model could learn to use that as a feature, aka data leakage. Certain fire stations may be slower at getting on the road which would result in longer overall response time, and we would like to learn that information–not only the driving time. We did include additional information from the Google Maps API, including the estimated driving time from the firehouse to the scene of the incident, the estimated distance from the firehouse to the scene of the incident, and the estimated driving speed. The Google Maps API returns the estimated travel time if a driver were to leave the station at the time the API was called. The current results are based API calls from a single day. One future direction could be modeling historical traffics patterns and dynamically adjusting for current traffic. We also included information about the call itself, including the time of day, the day of the week, the zip-code, the station area, and the call type.

For benchmarking Spark ML performance with different data sizes, we used 1) small and 2) big data sets with different numbers of features Table III)

B. Results

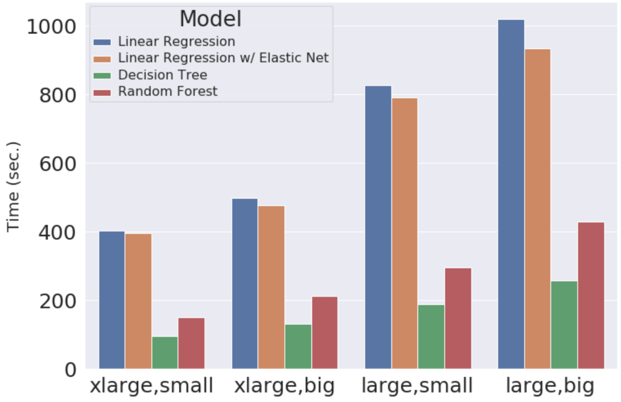

Fig. 3: Effect of Cluster & Data Size on Training Time

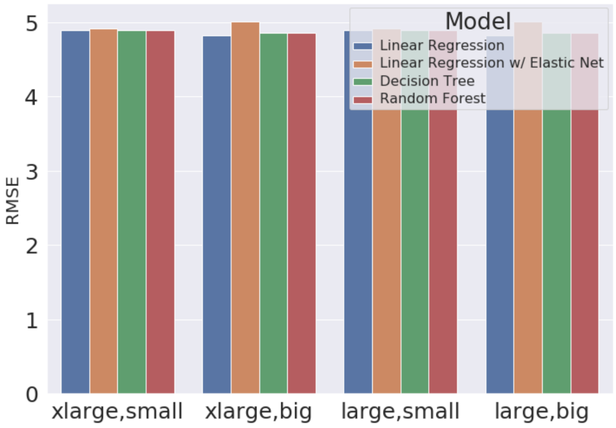

Fig. 4: Effect of Cluster & Data Size on RMSE

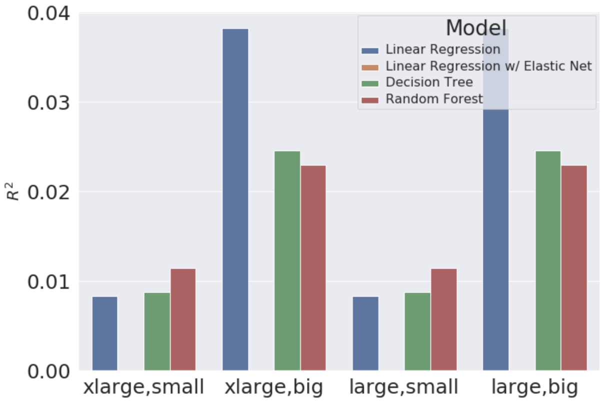

Fig. 5: Effect of Cluster & Data Size on R^2

The Root Mean Squared Error (RMSE) is a common metric regression and computed as the root of the squared sum of distances of predicted and true values. For the k-th record, the error in predicting firefighter response time in minutes is calculated. Then all errors over n total number of records are averaged and the square root is taken to calculate RMSE. RMSE has the nice property of being in the unit of the observations, so an RMSE of 5 indicates an average error in predicting response time of 5 minutes. (see Equation 1).

$$RMSE = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(predicted_k - true_k)^2} \tag{1}$$

We also used Mean Absolute Error (MAE), a similar metric to RMSE, which has the property of being in the same units as the values themselves. The biggest difference is that MAE is less likely than RMSE to be inflated in the presence of outliers. Equation 2 shows how to calculate MAE.

$$MAE = \frac{1}{n}\sum_{k=1}^{n}|predicted_k - true_k| \tag{2}$$

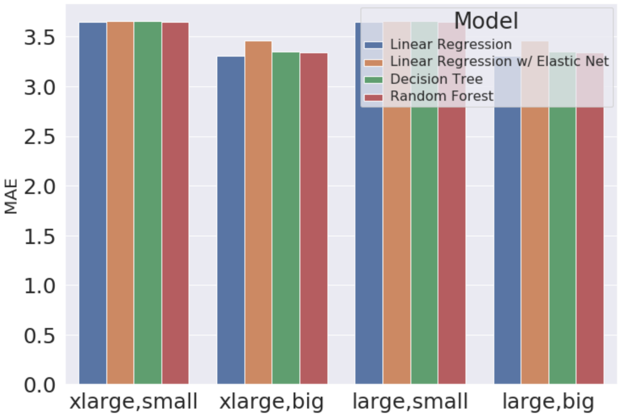

Fig. 6: Effect of Cluster & Data Size on MAE

The final metric that we used to evaluate our regression results is R^2. R^2 measures how much of the variance in the independent variable is predicted by the model versus how much overall variance exists in the independent variable. An R^2 of 1 represents a model that perfectly captures the variation in the response times. Note that we omitted the R^2 metric for the the Linear Regression with Elastic Net Regularization because the R^2 value gets warped by regularization.

Of the measurements that we took regarding the model training process on different computers and with different data, the time spend training the model had the biggest differences. Unsurprisingly, the extra large machines were able to fit the models faster than the smaller machines, and training on the big dataset took longer than training on the small dataset. What is worth noting is that it is faster across all models to have an extra large computer with big data than a large computer and small data.

When looking at RMSE and MAE, it is clear that the results are slightly better on the larger dataset. However, it does not appear that the size of computer has a significant effect on either of these two metrics. The R^2 metric displays some interesting variation among the size of the dataset. However, the results are nearly identical between different machine sizes.

V. Conclusion

The combination of low-cost cloud computing and Spark's easy-to-use interface provide a powerful resource for ML practitioners to create large scale, highly accurate models. For ML problems such as SFFD with large volumes and high cardinality (locations), Spark's scalable throughput enables ML teams to avoid common big data compromises such as reducing problem scope, widening predictive accuracy requirements, or avoiding the application of complex ML techniques. In addition, Spark's built-in ML algorithm library enables a smooth scaling from smaller to larger scale production; Spark ML's data structures, concepts, and parameters are consistent with other popular Python ML libraries.

Our research has demonstrated that a Spark cluster was a good choice to run ML algorithms at scale, though this current result does come with a number of caveats. Practitioners must carefully weigh the trade-off between improvements in predictive performance and run-time (cost) for their particular application. With this particular dataset, we found that not only did increasing the dataset size have a negligible improvement on our classification accuracy, but also that using complex tree-ensemble methods did little to improve results generated from simpler algorithms. In fact, the simpler Logistic Regression algorithm had a higher accuracy on the hold-out test set than either of the more complex algorithms.

With this in mind, one has to be careful in choosing their methods. Data size nor algorithmic complexity will alone maximize prediction accuracy. In particular, if a marginal improvements in predictability are not being sought, then a simpler method—perhaps one that also has greater interpretability—may be a more appropriate machine learning algorithm.

Given the data is continuously updated and the ease of model fitting, creating an automated machine learning that would fetch new data and fit the models to it would be an intriguing future direction. That system could be batch (train models from scratch with new data) or online (already trained models would be incremented updated with new data). Apache Spark supports this type of automated workflows with their Pipelines API and many of ML algorithms have a streaming implementation.

References

[1] (2017) U.s. census bureau quickfacts: San francisco county, california. United States Census Bureau. [Online]. Available: https://www.census.gov/quickfacts/sanfranciscocountycalifornia

[2] (2019) About us — fire department. City and County of San Francisco Fire Department. [Online]. Available: https://sf-fire.org/about-us

[3] R. Bowman, "Deaths expected from delayed emergency response due to neighborhood traffic mitigation," *Paper submitted to City Council, Colarado*, 1997.

[4] J. B. Myers, C. M. Slovis, M. Eckstein, J. M. Goodloe, S. M. Isaacs, J. R. Loflin, C. C. Mechem, N. J. Richmond, and P. E. Pepe, "Evidence-based performance measures for emergency medical services systems: A model for expanded ems benchmarking: A statement developed by the 2007 consortium us metropolitan municipalities' ems medical directors," *Prehospital Emergency Care*, vol. 12, no. 2, pp. 141–151, 2008.

[5] S. Yang, "Integrated management of emergency vehicle fleet," Ph.D. dissertation, 2006.

[6] R. Vlad, C. Morel, J.-Y. Morel, and S. Vlad, "A learning real-time routing system for emergency vehicles," in *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, vol. 3. IEEE, 2008, pp. 390–395.

[7] (2015) San francisco fire department - what does the future hold? San Francisco Fire Department. [Online]. Available: http://civilgrandjury.sfgov.org/2014_2015/14-15_CGJ_Report_SFFD_What_Does_the_Future_Hold_%207_16_15v2.pdf

[8] S. S. W. Lam, F. N. H. L. Nguyen, Y. Y. Ng, V. P.-X. Lee, T. H. Wong, S. M. C. Fook-Chong, and M. E. H. Ong, "Factors affecting the ambulance response times of trauma incidents in singapore," *Accident Analysis & Prevention*, vol. 82, pp. 27–35, 2015.

[9] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *2009 IEEE International Conference on e-Business Engineering*. IEEE, 2009, pp. 281–286.

[10] B. Furht and A. Escalante, *Handbook of cloud computing*. Springer, 2010, vol. 3.

[11] Z. Chaczko, V. Mahadevan, S. Aslanzadeh, and C. Mcdermid, "Availability and load balancing in cloud computing," in *International Conference on Computer and Software Modeling, Singapore*, vol. 14, 2011.

[12] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation computer systems*, vol. 28, no. 3, pp. 583–592, 2012.

[13] Amazon Web Services. (2019) Amazon web services (aws). [Online]. Available: https://aws.amazon.com/

[14] P. Gupta, A. Seetharaman, and J. R. Raj, "The usage and adoption of cloud computing by small and medium businesses," *International Journal of Information Management*, vol. 33, no. 5, pp. 861–874, 2013.

[15] Amazon Web Services . (2019) Amazon s3. [Online]. Available: https://aws.amazon.com/s3/

[16] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[17] 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.

[18] P. Deyhim, "Best practices for amazon emr," *Technical report*, 2013.

[19] Amazon Web Services. (2019) Amazon ec2. [Online]. Available: https://aws.amazon.com/ec2/

[20] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed systems*, vol. 22, no. 6, pp. 931–945, 2011.

[21] Amazon Web Services. (2019) Overview of amazon emr architecture. [Online]. Available: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview-arch.html

[22] DataSF. (2018) Sffd service calls. Google Cloud Platform. [Online]. Available: https://console.cloud.google.com/marketplace/details/san-francisco-public-data/sffd-service-calls?filter=category:public-safety

[23] (2019) Bigquery - analytics data warehouse. Google Cloud. [Online]. Available: https://cloud.google.com/bigquery/

[24] (2019) Distance matrix api. Google Maps Platform. [Online]. Available: https://developers.google.com/maps/documentation/distance-matrix

[25] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[26] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[27] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[28] Decision trees. Apache Spark. 2019. [Online]. Available: Decisiontrees

[29] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.