USER BEHAVIOR ANALYSIS USING SMARTPHONES

by

Seyedfaraz Yasrobi

July, 2017

Director of Thesis: Dr. Nasseh Tabrizi, PhD

Major Department: Computer Science

Users activities produce an enormous amount of data when using popular devices such as smartphones. These data can be used to develop behavioral models in several areas including fraud detection, finance, recommendation systems, and marketing. However, enabling fast analysis of such a large volume of data using traditional data analytics may not be applicable. In-memory analytics is a new technology for faster querying and processing of data stored in computers memory (RAM) rather than disk storage. This research reports on the feasibility of user behavior analytics based on their activities in applications with a large number of users using in-memory processing. We present a new instantaneous behavioral model to examine users activities and actions rather than results of their activities in order to analyze and predict their behaviors. For the purpose of this research, we designed a software to simulate user activity data such as users swipes and taps, and studied the performance and scalability of this architecture for a large number of the users.

# USER BEHAVIOR ANALYSIS USING SMARTPHONES

A Thesis

Presented to the Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Masters of Science in Computer Science

by

Seyedfaraz Yasrobi

July, 2017

USER BEHAVIOR ANALYSIS USING SMARTPHONES

by

Seyedfaraz Yasrobi

APPROVED BY:

DIRECTOR OF THESIS:

Dr. Nasseh Tabrizi, PhD

COMMITTEE MEMBER:

Dr. Venkat Gudivada, PhD

COMMITTEE MEMBER:

Dr. Qin Ding, PhD

CHAIR OF THE DEPARTMENT
OF COMPUTER SCIENCE:

Dr. Venkat Gudivada, PhD

DEAN OF THE
GRADUATE SCHOOL:

Dr. Paul J. Gemperline, PhD

This thesis is dedicated to my parents Baha and Faegheh for their endless love, support and sacrifice.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# CHAPTER 1: INTRODUCTION

Big data analytics has become significantly more important for both academic and professional communities over the past two decades and has had a great impact on different areas such as the Internet of Things (IoT), retail, healthcare, social networking, and finance [1]. This impact creates an opportunity to invest in devices such as smartphones, which are providing a continuous connection to the Internet, along with various interaction methods with their users [2].

Smartphone users continuously produce data directly and indirectly while using their phone. According to a study regarding interaction time [3], which is total accumulated time of phone usage during the day, 90% of the users had a range of 20-100 minutes of interaction time per day. On the other hand, Statista [4] reports that the number of smartphones in the world is expected to grow into 2.5 billion in 2019, an increase of 1 billion over the number of smartphones in 2015. Furthermore, according to Statista by 2018, 36 percent of the world's population is projected to use smartphones.

The increasing number of smartphones along with long interaction times of their users makes them an ideal candidate to be a source of data for applications in the domain of big data analytics. The common point about most of the data sets and studies regarding human interactions with smartphones [3] is that the focus of analysis is the result of user activities such as app usage details and battery usage, which has a slower rate of production compared to activities themselves. Analysis of activities which have a greater rate of production, such as a user's taps and swipes, require a superior architecture which can process the large

amounts of data coming in simultaneously from numerous sources. This architecture has to be capable of analyzing these streams of data flowing from smartphones into the data analytics component.

We utilize Apache Kafka as our distributed stream processing engine, it allows for the ingestion of stream/IoT data to be robust and scalable, which is important for a system that may be collecting data from millions of devices at a time. Agarwal and Prasad [5] demonstrate that stream processing is particularly useful in the implementation of real-time application usage analytics. Our research expands on this by processing less-structured idata within the domain of mobile devices.

Thesis presents a new real-time behavioral model to examine users' activities and actions rather than results of their activities to analyze their behaviors. Prediction of users' demographic information is part of user profiling which is an important subject in the area of personalization. The importance of user profiling arises from the fact that users are reluctant to give their demographic information away.

This scalable architecture utilizes Hadoop framework including Kafka and Spark applications which is a popular platform for distributed computing, to enable analysis of high volume streams coming from a large number of smartphones. We have studied the challenges and performance issues of this architecture under a variety of circumstances, and implemented performance optimization methods that increase the response time of this system.

Contributions of this thesis:

- Demonstrated the opportunities and challenges that large-scale smartphone applications create, using their sensors and interaction methods.
- The new architecture for analysis of stream/IoT data.
- Performance analysis of proposed user behavior analytics architecture and its scalability for a large number of simulated users.

# CHAPTER 2: RELATED WORKS

## 2.1 User Activities in Smartphones

User activities are defined as any data that can be produced by users' actions directly and indirectly. These activities are categorized into different types [3]:

- App sequence and usage
- Screen interactions
- Microphone
- Camera
- Sensors
- Wi-Fi and Bluetooth

In the past several years there have been numerous studies researching the challenges and the opportunities that smartphone data streams creates.

Miguel-Hurtado et al. [6] studied user swipes and tried to predict the gender of the users. They have reported the accuracy of 78 percent using swipe gesture data from two different directions.

According to MindMining project by Hoppe et al. [7], building a rich user profile is feasible using semantic technologies in addition to machine learning techniques. This project deals with highly heterogeneous Web-based information, which is mainly navigation traces of users on the web.

Smartphone sensor data was studied by Aichinger et al. [8] to predict location of crash spots in road networks. They utilized smartphone GPS and motion sensor data to automatically recognize critical car driving situations and near-misses such as emergency braking, evasion maneuvers or sudden driving speed changes.

Mirsky et al. [9] developed an automated system to prevent attackers from accessing private information of users by use of data streams coming from smartphones such as CPU consumption, accelerometer readings, etc. They have created an algorithm called pcStream that detects anomaly by use of data streams.

## 2.2 Data Analytics on Data Streams

Liu et. al. [10] have proposed a distributed video stream processing architecture using Apache Kafka, Storm, and Hadoop. Their results show that the solution performs well regarding scalability, fault tolerance, and efficiency.

Integration of Apache Kafka and Spark to process and analyze web usage logs was further studied by Agarwal et al. [5]. Their proposed framework shows how this technology can be used to create up=to-date statistical profiles of web usage patterns.

In the application of machine learning algorithms to streaming data study by, Nair et al. [11] they have used Spark based machine learning model for predicting health status of users based on the twitter data streams. Their research implies the necessity of big data architecture such as Hadoop ecosystem for processing of such a large stream of data.

Mestre et al. [12] have implemented an efficient spark-based adaptive windowing for entity matching. This research has shown that how this problem can be solved and optimized by use of a distributed computing framework such as Hadoop and Spark.

## 2.3 Machine Learning

Machine learning with its automated learning power unleashes big datas power to aid data scientists to gain knowledge in a variety of applications such as computer vision, speech processing, natural language understanding, neuroscience, health, and Internet of Things (IoT). The major challenges of using machine learning in big data are to perform the analysis in a reasonable time [13].

Machine learning is a type of artificial intelligence that provides computers with the ability to learn autonomously, using a combination of methodologies developed by the statisticians and computer scientists, to learn relationships from data while also placing emphasis on efficient computing algorithms. Machine learning is also used in the analysis and diagnosis of medical images in radiological medicine, predict the susceptibility of soil liquefaction, and forecast models of consumer credit risk. The techniques used within these countless applications of machine learning techniques each fall under one of two categories of supervised or unsupervised learning [14–17].

In this study, we have examined the performance of the following five machine learning algorithms.

a) Logistic Regression: Linear regression attempts to fit a line to data that has only two levels or outcomes, whereas, logistic regression models the chance of an outcome based on a transformation known as a logit [18]. In this study, we examined Spark's two optimizers which are both available for logistic regression algorithm. These optimizers are Stochastic Gradient Descent (SGD) and Limited Memory Broyden Fletcher Goldfarb Shanno algorithm (L-BFGS) [19].

The SGD approximates the gradient by accessing a single element within the dataset during each iteration. It works best for machine learning algorithms that are large in magnitude because it does not require loading the entire dataset. It is also inexpensive in regards to

computations, and the dataset can be processed quickly. Those benefits are key factors that influence many to use SGD to optimize logistic regression and SVM algorithms [20]. LBFGS is a quasi-Newton method optimizer that uses prior iterations to estimate the Hessian matrix of the objective function. These previous iterations have a sequence of gradient vectors that are used to solve unconstrained nonlinear minimization problems. Logistic regression utilizes LBFGS to optimize its performance among the machine algorithms that we mentioned previously.

b) Support Vector Machine (SVM): The Support Vector Machine algorithm [20] uses training examples to create a hyperplane that separates the dataset into classes. The complexity of classes may vary, but the simplest form of the SVM algorithm has only two possible labels to choose from. To reduce misclassifications, a decision boundary is obtained while training the SVM algorithm. This decision boundary is known as the optimal separation hyperplane. The only optimizer available for SVM on Spark is SGD optimizer.

c) Decision Trees: Classification via decision trees begins with a series of questions about the various features of the dataset. Each question is housed in a node that points to at least one child node that responds to the question. A hierarchical tree is formed as a result of these questions and thus, allowing the classification of an item based on how we answer the questions. A classification occurs once we have reached a leaf node [21].

d) Random Forests: Random forests and boosted trees are other forms of classifiers that are based on decision trees and yield more precise classifications by its multiple decision trees. Random forests are one of the methods of tree ensembles where tree predictors are combined in such a way that each trees prediction depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. In the process of training a random forests predictor model, the root node corresponds to whole input space and this input space will be partitioned into multiple disjoint partitions. At each of these nodes, a decision tree model will be trained based on its data partition [22].

e) Gradient Boosted Trees: One of the common methods for improving the accuracy of any machine learning algorithm is by boosting it. This method has been introduced by Sapphire [23] and utilizes a combination of weak classifiers to create a sturdier classification model. Gradient Boosted Trees classification algorithm allows for first decision tree model to be trained based on the training dataset, improving the accuracy of the model iteratively by retraining the model. During each training, tuples will be reweighted so that the model can decrease its error rate. Boosting does not suffer from overfitting regularly, but because of its iterative nature takes a longer time to train the model.

### 2.4 Machine Learning Data

In this study, HIGGS dataset from UCI machine learning repository [24] has been adopted and produced using Monte Carlo simulations. The first 21 features are kinematic properties measured by the particle detectors in the accelerator and the last seven features are functions of the first 21 features; these are high-level features derived by to help discriminate between the two classes.

# CHAPTER 3: BIG DATA INFRASTRUCTURE

## 3.1 Hardware Components

Big Data analytics require a scalable hardware infrastructure with parallel processing capability. This system should have enough memory, bandwidth, and throughput to run multiple tasks simultaneously, and perform parallel processing of advanced analytics algorithms in a matter of seconds. Since the central concept of Big Data computing is distributed processing, in this study, we implemented the framework over a cluster of servers. We have arranged a cluster of 16 servers providing a robust hardware base for big data analytics tasks. Four of these servers act as administrative nodes, and 12 servers operate as worker nodes. Each of the 16 servers has two Intel(R) Xeon(R) quad-core CPU 5620 2.40 GHz processors, meaning there are eight real cores or 16 virtual cores on each server. The servers consist of 16 GB DDR3 RAM and a 1 TB hard disk. The operating system used is a Linux Ubuntu server 14.04 64-bit distribution. The switch used is Juniper EX4200, which is a high-performance, low-latency, and provides a one Gigabit Ethernet (GbE) access environment.

## 3.2 Software Components

To analyze the performance of machine learning algorithms on Spark, we began by building a Hadoop cluster using YARN as the resource manager. YARN is a platform that provides consistent operations, security, and data governance tools across Hadoop clusters. We chose YARN as the resource manager because it outperforms Sparks standalone mode for handling

clusters. YARN allows for multi-tenancy, dynamic allocation of cluster resources, and data center expansion [25]. A bash script was created that ran machine learning models on the HIGGS dataset with data size of 8 gigabytes, 28 attributes, and 11 million instances [24]. To test the performance of the machine learning algorithms, we utilized a maximum of 7 machines with 8 virtual cores and 16 gigabytes of RAM. The amount of data partitions were also taken into account when analyzing the performance of the machine learning algorithms. There was a range of 40 to 60 partitions used. The performance time of the machine learning algorithms was first averaged in regards to the number of machines and then averaged by the number of cores.

### 3.2.1 Apache Spark

Spark is a framework for the parallel processing of Big Data. Spark is designed to use Hadoop MapReduce with some modifications that enable it to perform more efficiently. Apache Spark has a streaming API and independent processes for continuous micro-batch processing across intervals with varying, but short time duration. Spark runs up to 100 times faster than Hadoop in certain circumstances. Spark has some features for real-time analytics and is supportive of applications such as machine learning, stream processing, and graph computation [26]. MLlib is Spark's machine learning library, focusing on learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives. MLlib is built on Apache Spark, which is a fast and general engine for large-scale processing that is up to 100x faster than Hadoop MapReduce or 10x faster compared to disk. It supports Java, Scala, and Python. Various big data frameworks take advantage of in-memory processing and of those frameworks the one that is mostly used is Apache Spark, a descendant of Hadoop. Using its in-memory capabilities Apache Spark executes machine learning algorithms efficiently by keeping the dataset in RAM and eliminating the repetitious pulling of data from the hard

disk [27]. For example, the machine learning algorithm known as logistic regression is over 100 times faster when using Spark instead of Hadoop [28].

Spark has caused a surge of interest due to companies desire to utilize real-time analytics, analysis of streaming IoT data sets, and efficient implementation of iterative algorithms. But regardless of its benefits, the unfamiliarity of many companies with big data cluster specifications such as the number of machines, number of cores per machine, and the interoperability of machines, often results in lower hardware usage efficiency.

Every data analysis ecosystem has two main components which are filesystem and processing system. Spark, handles files in the form of Resilient Distributed Datasets (RDD) and processes them with its in-memory processing engine. In the following sections, we discuss these components as well as our choice of test data set and algorithms.

### 3.2.2 In-memory Processing Engine

In-memory processing has been studied since the 1980s, but recently it has gained popularity due to the availability of ultra-fast memories, each with its massive capacity often offered at the lower cost [29, 30]. There are several significant differences between processing in main memory versus processing on hard disks. The most quantifiable difference is the fact that main memory processes computations notably faster than hard disks [31]. This is because in-memory processing places the computation near the data as means to reduce data movement [32]. Despite the volatility and vulnerability of main memory, the processing speed and other benefits make using in-memory processing worthwhile.

In-memory processing with its increased performance in response time has been proven beneficial in a range of areas such as fraud detection, risk management, and decision-making, where machine learning algorithms are used to conduct analytics. Together with real-time analytics, it provides unmatched productivity and profit gains that explain why companies such as Facebook, Twitter, and Amazon take advantage of their capabilities [33, 34].

In-memory processing involves querying data in computers RAM instead of its hard disk. Being able to interact directly with RAM as opposed to the traditional method of hard disks I/O operations greatly increases response times and throughput. Acker et al. concluded that response times were 5 to 19 times better and throughput increased by seven times when using in-memory processing platforms. It has been reported that in-memory processing is easier to set up and maintain which is ideal when analyzing large datasets [35], reduction in memory prices, and higher user satisfaction that enables faster data access that contributes to faster decision making. [36–38].

### 3.2.3 Resilient Distributed Datasets

Resilient Distributed Datasets (RDDs) are data structures that partition the dataset into multiple partitions, and aid with parallel computing by supporting iterative operations on the Spark. The resilient feature of RDD that logs the transformations used to build a dataset will capture enough information to re-compute a lost or damaged partition. Furthermore, RDD gives users control over how the data is partitioned, the storage strategy for each RDD, and indicate which RDDs will be reused. Due to its parallel computing capabilities, RDD significantly improves the performance of large dataset computations on an in-memory computing platform [26].

# CHAPTER 4: USER BEHAVIOR ANALYSIS USING SMARTPHONES

## 4.1 Data Collection

This thesis studies the performance of user behavior analysis model with the designed architecture. Therefore we must observe and analyze the efficiency of the system under different circumstances. The response time of the system is one of the important factors, and it should not fall behind in data processing while analyzing a large number of streams in the short amount of time.

An experiment has been designed to observe the performance of the system. In this experiment, A large number of simulated data are streamed into the cluster and then analyzed to see how many of these streams are being processed in real-time depending on the complexity of the analysis.

## 4.2 Data Simulation

The data streams that are entering the cluster are the simulated version of the user behaviors, where each message is all information that we can gather from smartphone sensors. These behaviors can be a single swipe or a tap inside an application. For example, in a game that requires its users to react multiple times in a short amount of time, users will produce a significant amount of data in a short amount of time. Our data simulation software creates a large number of activities and sends them to the cluster for data analysis. See Fig. 4.1.

To create valid data for simulation, we have defined some static rules for each field of the

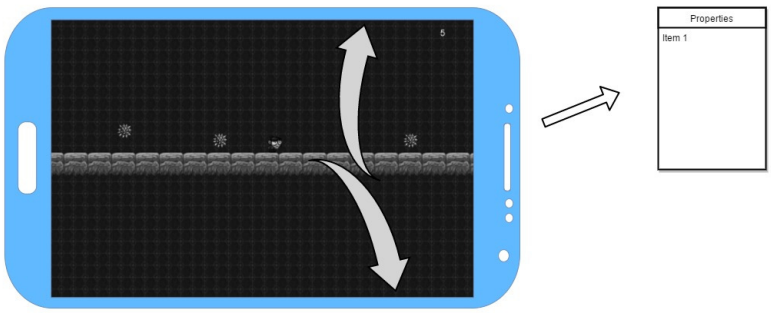


Figure 4.1: Details of game swipes

data, and then data is being produced using those static rules. On the other hand, since our online analysis only involves statistical operations and no content-based analysis like machine learning algorithms, the only thing that would matter is the length of the streams and their quantity. Table 4.1 and 4.2 showing the characteristics of the each field of the data that simulator is sending to the cluster.

Table 4.1: Information gained based on swipe location

| | |
|---|---|
| Total length (px) | Maxima speed (px/ms) |
| Total time (ms) | Average speed (px/ms) |
| Width (px) | Maxima acceleration (px/ms2) |
| Height (px) | Average acceleration (px/ms2) |
| Area (px2) | Average arc distance (px) |
| Average thickness (px) | Max arc distance (px) |
| Average pressure | Angle start to end (degrees) |

## 4.3 System Architecture and Topology

The smartphones initially connect to a name server, which will forward the IP address of a game server to which they connect. The game server runs a Kafka "producer" application, which will directly take the received data and forward it to the Kafka cluster. The cluster processes the streaming data then organizes it into topics and replicates it for easy consumption. Kafka also sends the processed data to a database for short-term storage in the HDFS

Table 4.2: Information gained based on accelerometer

| | |
|---|---|
| Starting acceleration (x) | Ending acceleration (x) |
| Starting speed (x) | Ending speed (x) |
| Starting location (x) | Ending location (x) |
| Starting acceleration (y) | Ending acceleration (y) |
| Starting speed (y) | Ending speed (y) |
| Starting location (y) | Ending location (y) |
| Starting acceleration (z) | Ending acceleration (z) |
| Starting speed (z) | Ending speed (z) |
| Starting location (z) | Ending location (z) |

this action will prevent data loss in case of any unexpected failures. The spark cluster is running a Kafka "consumer" application and will receive data in micro-batches for statistical analysis. It processes the data and adjusts its model accordingly. We also use Apache Flume to store data in the HDFS directly from Kafka, for offline data analytics. HDFS acts as our data persistence layer and runs as a cluster. See Fig. 4.2.

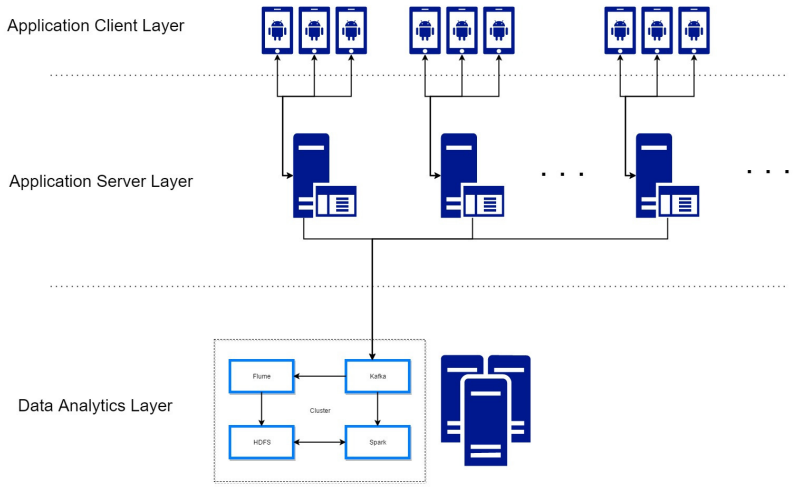


Figure 4.2: Details of system topology

Apache Kafka is a distributed message log framework, and its loose coupling of data producers and data consumers allows for the scalability and fault tolerance which we desired for this research. Our game servers connect to the Kafka cluster whenever they need to send

data, and our Spark Streaming Contexts (SSC) connect to our Kafka cluster to continuously consume the data for updating the predictive model. These topics are replicated across multiple brokers for fault-tolerance, and one broker is designated the "leader" for a topic. Additionally, each of the consumers has an index which is kept track of by the brokers. The coordination of the brokers is achieved through Apache Zookeeper. We only have one topic to which the SSC subscribe, but each data point contains multiple parameters to be used in the analytics. See Fig. 4.3.

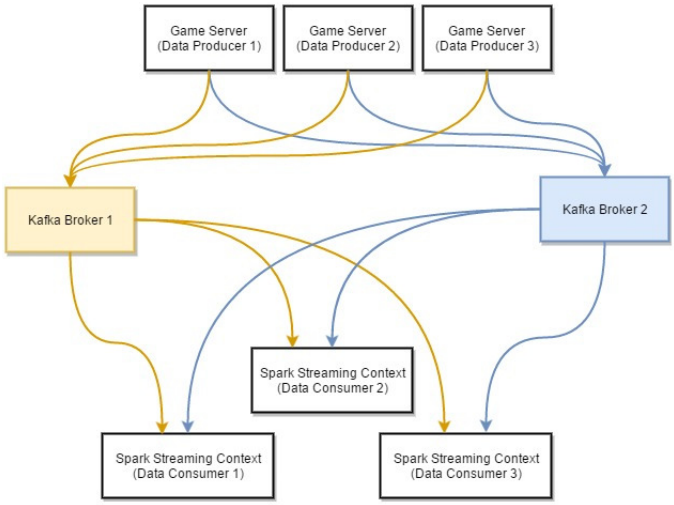


Figure 4.3: Details of Kafka cluster architecture

## 4.4 Data Analysis

The problem that we want to solve is a classification problem where we want to predict the gender of the users, and the architecture will perform this task as shown in Fig. 4.4 using two different types of machine learning algorithms. Our model has a combination of online and offline machine learning algorithms, which helps with building more complicated models with better throughput for real-time analysis.

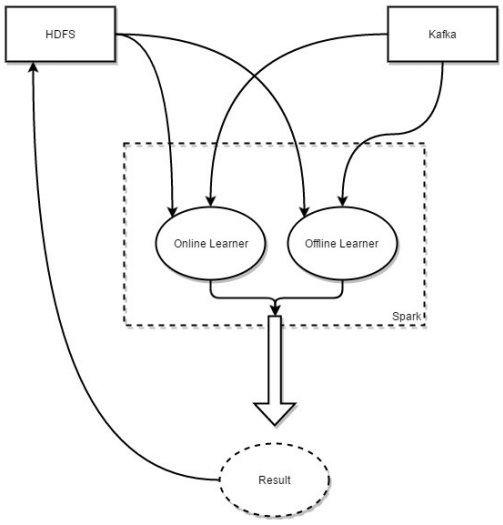


Figure 4.4: Details of the data analytics

# CHAPTER 5: EVALUATION AND RESULTS

## 5.1 Experimental Condition

For the purpose of evaluation of we setup five servers to send simulated user activity data, these servers will send data and results based on the number of the users and the number of the extracted features. On the receiving side, the data will be captured by a distributed message passing system. Finally, the captured data will be stored in HDFS for offline learning, and also it will be forwarded to our real-time data analysis system for online learning

For the purpose of offline learning, an experiment was set up to investigate the performance of the machine learning algorithms with 128 different configurations, the number of servers 3 through 10, and the number of virtual cores 1 through 16 with data chunk size of 64 megabytes. The minimum number of servers needed to perform machine learning algorithms were selected to conform with the replication factor of 3 that refers to the storing of data on different machines.

## 5.2 Analysis of Data Ingestion Component

In this section, we are presenting the benchmark for Apache Kafka which is our chosen distributed message passing system. Apache Kafka groups messages based on the topic, these topics lets data producers and consumers communicate synchronously. Apache Kafka distributes messages in each topic into multiple partitions and lets each of the Kafka brokers

to lead one of the partitions, so messages spread through the Kafka cluster. Fig. 5.1 illustrates the schematic of this distribution.

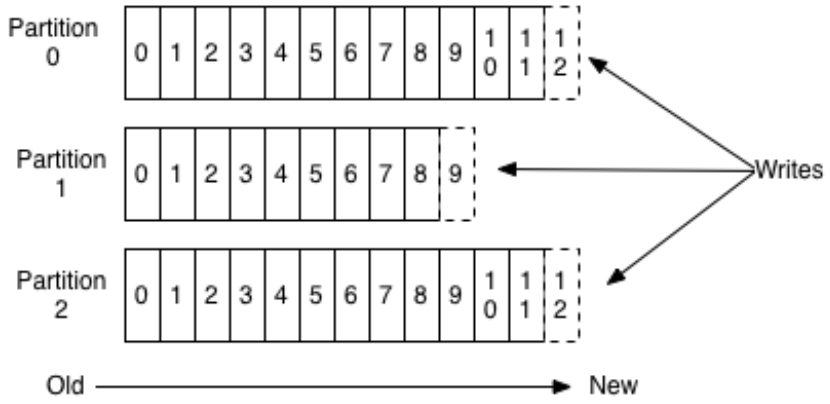


Figure 5.1: The distribution of messages among Kafka brokers

The Kafka cluster consists of 5 Kafka brokers as shown in Fig. 5.2 where we have analyzed the load of the network based on the number of the partitions, the number of the messages passed to the Kafka cluster and also the number of the features captured. The format of the captured data will be in Comma Separated Values (CSV). The number of the features depends on which sensors we are trying to capture data from, where it can vary from 20-200 features.

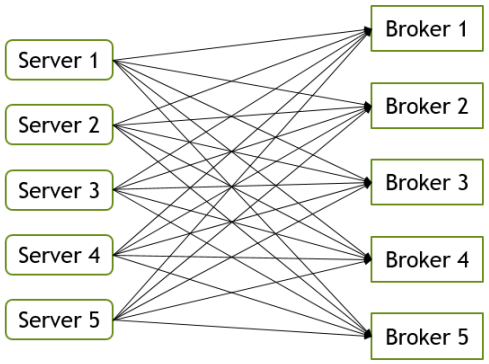


Figure 5.2: The distribution of messages among Kafka brokers

Kafka producers use Round Robin algorithm to distribute messages among Kafka brokers, and our observations indicate that this distribution is unequal among brokers most of the time regardless of configuration parameters. The message distribution of this system has been presented in the Figs. 5.3, 5.4, and 5.5. The unequal message distribution can result in the waste of resources, and limits the message receiving capacity of this system and requires significant improvement.

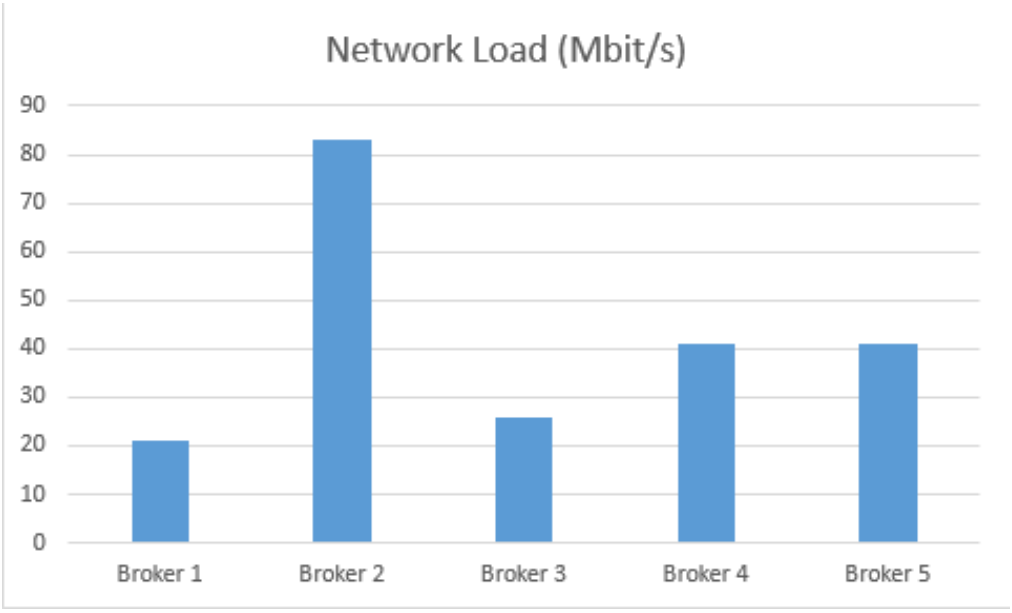


Figure 5.3: The distribution of messages among Kafka brokers with 30k messages per second

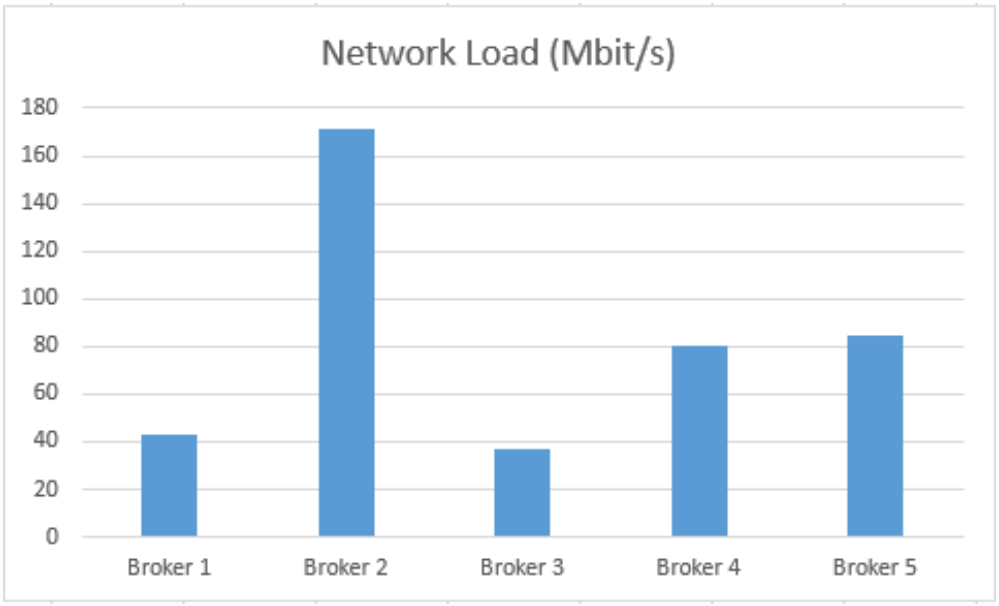


Figure 5.4: The distribution of messages among Kafka brokers with 60k messages per second

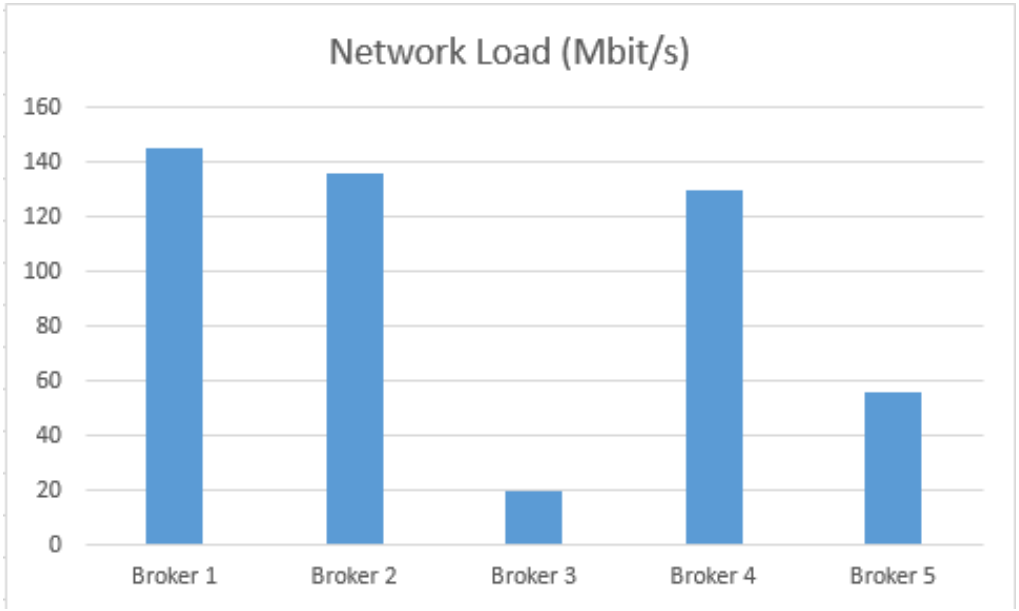


Figure 5.5: The distribution of messages among Kafka brokers with 90k messages per second

As stated before the messages will be received with public interface of the Kafka cluster, and it needs to be replicated before being passed to consumers. The reason for replication is to make system fault-tolerant. Replicating the message on few brokers will make our system

immune to failed broker nodes. Although replication makes the system resistant to failures, it also puts a heavy load of network traffic on the intra-network of the Kafka brokers. In Fig. 5.6 and 5.7 the total network load of these brokers with different replication factors has been presented.

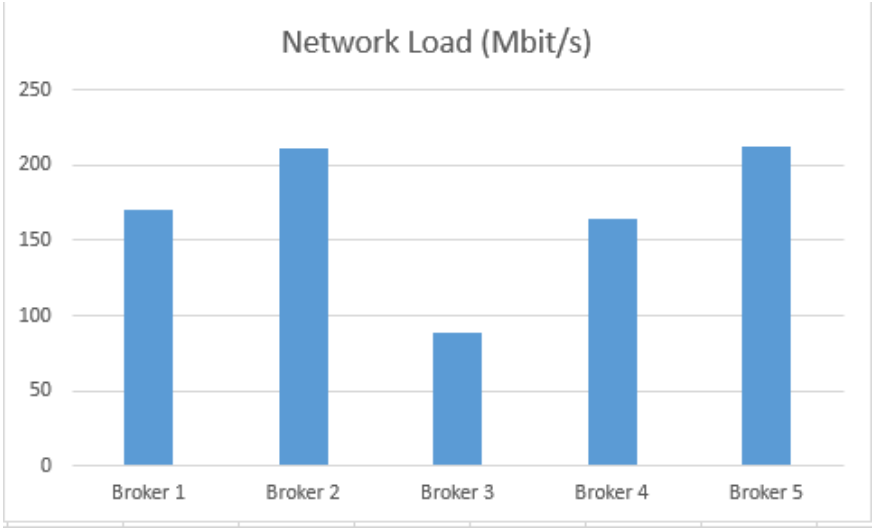


Figure 5.6: The replication of messages among Kafka brokers with replication factor of 2

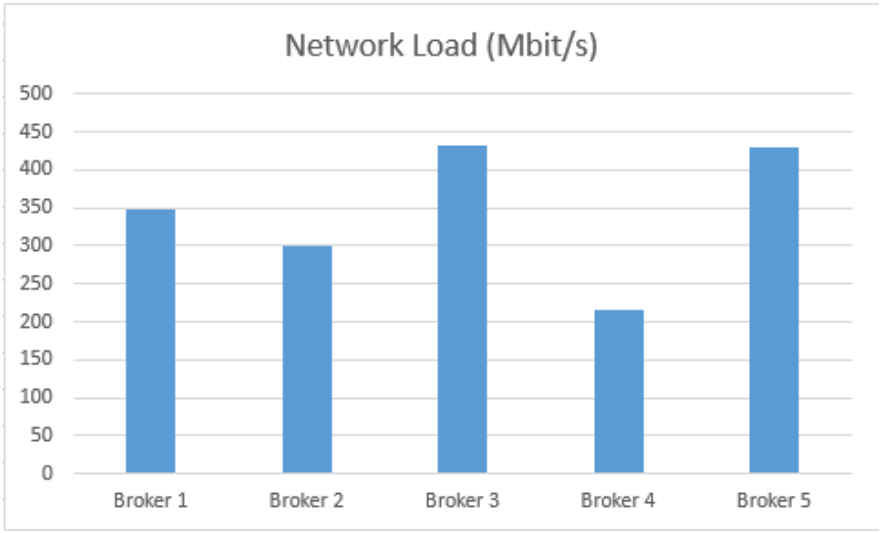


Figure 5.7: The replication of messages among Kafka brokers with replication factor of 3

### 5.3 Online Analysis of Data Streams

After the ingestion of data into the cluster, the next step is moving data into processing unit (Apache Spark) for performing the actual analysis. Worker nodes will do this step of the data analysis. Apache Spark allows its users to specify how many streams should be ingested into worker nodes simultaneously as well as how many threads should work on ingestion of these streams. Increasing the number of ingested streams will make more data available for worker nodes for the final step of the program which is data processing. On the other hand, if we overwhelm the processing unit with the excessive amount of data, this would result in lower performance since worker nodes will not be able to process ingested data on time.

As mentioned earlier, the final step is data processing of data streams. In this section, we are presenting the report of the performance analysis of our system under different analysis scenarios. For the purpose of performance analysis of the online analytic component of the system, we have designed several data analysis pipelines, and we are observing how many streams can we process in real-time in each step of the process.

The important parameters that should be noted while doing data processing are the number of data partitions which specifies the level of parallelism in data processing as well as the number of virtual cores assigned to a worker node. Data partitioning allows worker node threads to do the processing simultaneously, but increasing this number can also result in overheads that can decrease the performance of the processing task, so it is essential to choose a correct number of partitions for every data analysis task.

To summarize the relevant parameters affecting the performance of online data analysis, we should determine the number of data streams and also the number of threads assigned to this task. Also the number of virtual cores assigned to worker nodes and the number of partitions that the data is divided into are important.

This research reports the performance of real-time data processing on simulated user be-

havior data streams on our Hadoop cluster, and how changing above-mentioned parameters can affect the number of data streams that can be processed in real time. For example, Fig. 5.8 shows how many messages can be read and counted by cluster in 200 seconds. As shown, number of the streams in each worker node is an important factor and by increasing this number we can achieve higher data input in our cluster.

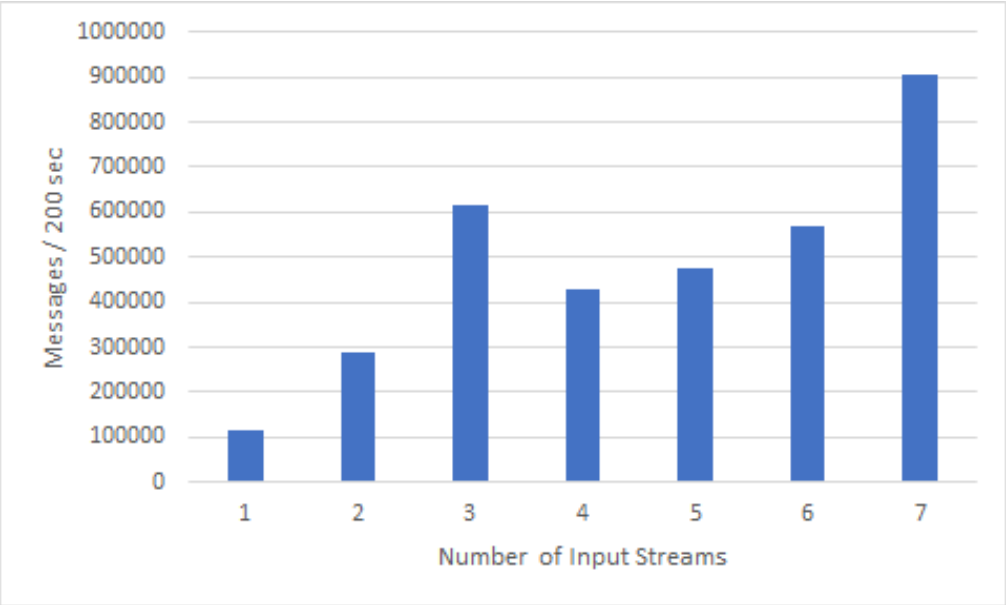


Figure 5.8: Number of the messages that cluster processes with different number of data streams in each worker node in 200 seconds

## 5.4 Offline Analysis of Stored Data

In the process of training the classification models, we observed that the split rate of selected data for training data set does not affect the training time of the machine learning algorithms. This means that the only parameters affecting the training time of classification models for particular dataset and infrastructure are software settings and the resources that are being used. We have discovered that the analysis won't complete if we don't provide enough RAM for the execution, and on the other hand adding excessive RAM will not increase the performance.

In this experiment, we trained the logistic regression model with two LBFGS and SGD optimizers, with over 100 different configurations. The average training time of SGD was 207 seconds and an average time of LBFGS was 106 seconds, as shown in Fig. 5.9, so we can conclude that logistic regression model can be trained 49 percent faster using LBFGS optimizer compared to SGD while being trained with similar cluster configurations.

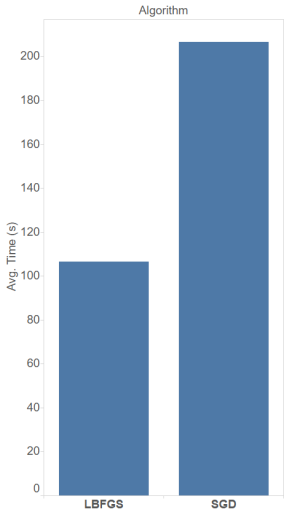


Figure 5.9: Comparison of average training time for logistic regression using SGD and LBFGS

We observed that an increase in the performance of machine learning algorithms as the number of servers increased which is an expected result. It is noteworthy to state, however, that adding additional servers will increase the performance, but decaying performance with a higher number of servers and adding these additional servers may outweigh the performance benefit. The results of this experiment are presented in Fig. 5.10.

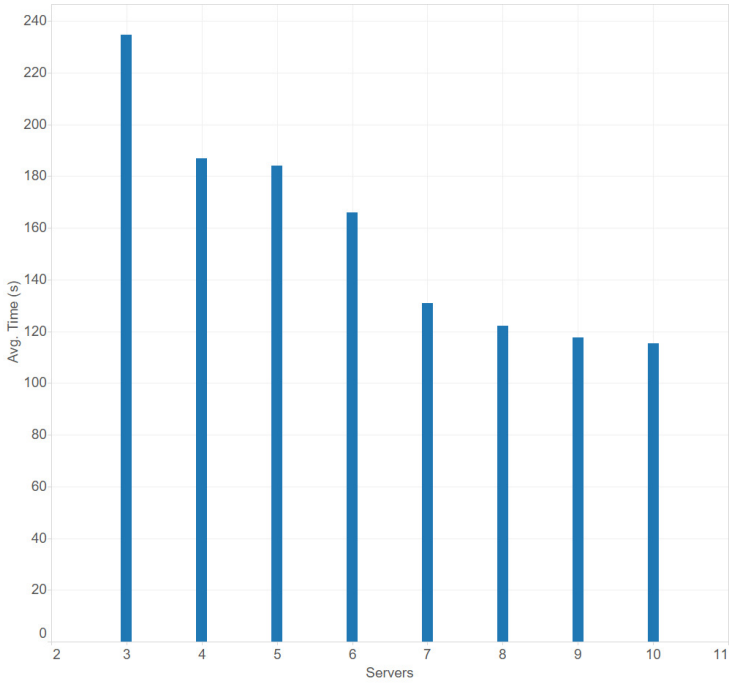


Figure 5.10: Time comparison of classification algorithms on different number of servers

We observed that having more than one virtual core makes a significant difference in the training time and having 2 through 8 virtual cores performs equally well. Furthermore, increasing number of virtual cores used on each server to more than 8 degrades the performance of machine learning algorithm. We conclude that independent of selected algorithm the efficient number of virtual cores is between 2 and 8. In the next step, we compared training time based on the number of servers.

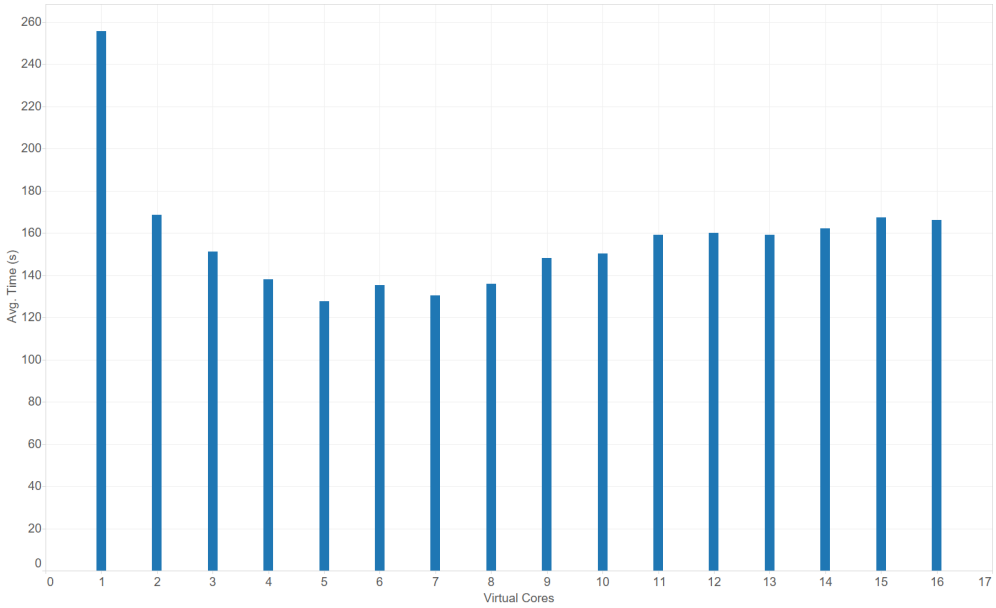


Figure 5.11: Comparison of average training time of classification algorithms per number of virtual cores on each server

As shown in Fig. 5.11, the relationship between the number of cores and the performance of the machine learning algorithms did not follow a similar pattern to that of the number of machines and performance. This interesting issue raises concerns regarding Spark and YARNs ability to manage resources in training of machine learning algorithms.

We further analyzed results using tree ensembles and as was expected random forests capability to adapt with parallel processing; will allow training higher number of subtrees for training predictor model. Fig. 5.12 shows the average training time for this algorithm using 5, 25 and 50 subtrees.

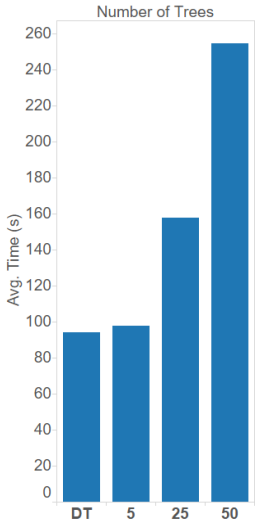


Figure 5.12: Training time comparison of decision tree and random forests with 5, 25, 50 sub trees

We Tested Gradient Boosted Trees with a different number of iterations and as expected the increasing number of iterations significantly increased its training time. Because of the iterative nature of this algorithm, parallelism and cluster computing only marginally helps with its boosting. Fig. 5.13 shows that the training time of gradient boosted tree with even few iterations increases rapidly.

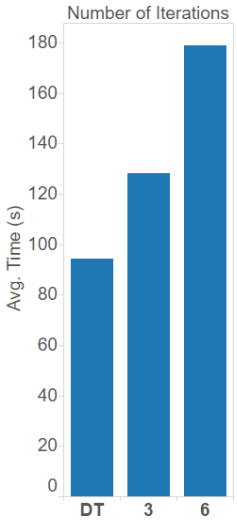


Figure 5.13: Training time Comparison of decision tree and gradient boosted trees with 3 and 6 iterations

# CHAPTER 6: CONCLUSION AND FUTURE WORKS

In this thesis we purpose an architecture to analyze the large streams of data streaming from smartphones. Furthermore, we studied the opportunities that this type of data can create. Our architecture is completely scalable and can process the data streams in real-time. We have presented the optimal server and core configurations needed to best perform machine learning models using Apache Spark on a specific dataset. These optimal configurations will not only save time, but will also reduce cost of infrastructure. The performance of these machine learning algorithms were analyzed in regards to time in seconds. Factors such as number of cores, number of machines, and number of partitions were taken into consideration when analyzing the performance of the machine learning algorithms. It was discovered that as the number of servers increased, the performance increased as well, but performance however decreased, as the number of cores increased. These results infer that more servers and fewer cores are needed for optimal performance of machine learning algorithms. From a business perspective, it is best to use the number of machines that makes a significant impact on performance rather than a meager impact as means to save money. This justifies why we chose a number of servers that was less than the maximum amount of machines that were available. Our methodology aimed to maximize benefits and resources while minimizing expenditures. For future works, researchers can implement out architecture on other distributed processing systems to analyze the accuracy of the results.

## REFERENCES


[1] Hsinchun Society for Information Management (U.S.), Roger H. L. University of Minnesota. Management Information Systems Research Center., Veda C. Society for Management Information Systems (U.S.), and Association for Information Systems. *MIS quarterly : management information systems.*, volume vol.36. Society for Management Information Systems, 1977.

[2] Muhammad Anshari and Yabit Alas. Smartphones habits, necessities, and big data challenges. *The Journal of High Technology Management Research*, vol.26(no. 2):pp. 177–185, 2015.

[3] Hong Cao and Miao Lin. Mining smartphone data for app usage prediction and recommendations: A survey. *Pervasive and Mobile Computing*, vol.37:pp. 1–22, 2017.

[4] Number of smartphone users worldwide 2014-2020 — Statista, 2017. URL `https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/`.

[5] Sonali Agarwal and Bakshi Rohit Prasad. High speed streaming data analysis of web generated log streams. In *2015 IEEE 10th International Conference on Industrial and Information Systems (ICIIS)*, pages pp. 413–418. IEEE, 2015.

[6] Oscar Miguel-Hurtado, Sarah V. Stevenage, Chris Bevan, and Richard Guest. Predicting sex as a soft-biometrics from device interaction swipe gestures, 2016.

[7] Anett Hoppe, Ana Roxin, and Christophe Nicolle. Dynamic, Behavior-Based User Profiling Using Semantic Web Technologies in a Big Data Context. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages pp. 363–372. Springer, Berlin, Heidelberg, 2013.

[8] Claus Aichinger, Philippe Nitsche, Rainer Stütz, and Marko Harnisch. Using Low-cost Smartphone Sensor Data for Locating Crash Risk Spots in a Road Network. *Transportation Research Procedia*, vol.14:pp. 2015–2024, 2016.

[9] Yisroel Mirsky, Asaf Shabtai, Bracha Shapira, Yuval Elovici, and Lior Rokach. Anomaly detection for smartphone data streams. *Pervasive and Mobile Computing*, vol.35:pp. 83–107, 2017.

[10] Xin Liu, Dehai Zhao, Liang Xu, Weishan Zhang, Jijun Yin, and Xiufeng Chen. A distributed video management cloud platform using hadoop. *IEEE Access*, 3:2637–2643, 2015.

[11] Lekha R. Nair, Sujala D. Shetty, and Siddhanth D. Shetty. Applying spark based machine learning model on streaming big data for health status prediction. *Computers & Electrical Engineering*, 2017.

[12] Demetrio Gomes Mestre, Carlos Eduardo Santos Pires, Dimas Cassimiro Nascimento, Andreza Raquel Monteiro de Queiroz, Veruska Borges Santos, and Tiago Brasileiro Araujo. An efficient spark-based adaptive windowing for entity matching. *Journal of Systems and Software*, vol.128:pp. 1–10, 2017.

[13] Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios Vasilakos. Machine Learning on Big Data: Opportunities and Challenges, 2017.

[14] Rahul C. Deo. Machine learning in medicine. *Circulation*, vol.132(no. 20):pp. 1920–1930, 2015.

[15] Shijun Wang and Ronald M. Summers. Machine learning and radiology. *Medical Image Analysis*, vol.16(no. 5):pp. 933–951, 2012.

[16] P. Samui and T. G. Sitharam. Machine learning modelling for predicting soil liquefaction susceptibility. *Natural Hazards and Earth System Science*, vol. 11(no. 1):pp. 1–9, 2011.

[17] Amir E. Khandani, Adlar J. Kim, and Andrew W. Lo. Consumer credit-risk models via machine-learning algorithms. *Journal of Banking and Finance*, vol.34(no. 11):pp. 2767–2787, 2010.

[18] Kristin Sainani. Logistic Regression. *PM&R*, vol.6(no. 12):pp. 1–28, 2006.

[19] Wenni Zheng, Pengbo Bo, Yang Liu, and Wenping Wang. Fast B-spline curve fitting by L-BFGS. *Computer Aided Geometric Design*, vol.29(no. 7):pp. 448–462, 2012.

[20] Krzysztof Sopyła and Paweł Drozda. Stochastic gradient descent with Barzilai-Borwein update step for SVM. *Information Sciences*, vol.316(no. C):pp. 218–233, 2015.

[21] Carl Kingsford and Steven L Salzberg. What are decision trees? *Nature biotechnology*, vol.26(no. 9):pp. 1011–3, 2008.

[22] D. Richard Cutler, Thomas C. Edwards, Karen H. Beard, Adele Cutler, Kyle T. Hess, Jacob Gibson, and Joshua J. Lawler. Random forests for classification in ecology. *Ecology*, vol.88(no. 11):pp. 2783–2792, 2007.

[23] Glenn De'ath. Boosted trees for ecological modeling and prediction. *Ecology*, vol.88(no. 1):pp. 243–251, 2007.

[24] P Baldi, P Sadowski, and D Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, vol.5:pp. 4308, 2014.

[25] David Siegal, Jia Guo, and Gagan Agrawal. Smart-MLlib: A High-Performance Machine-Learning Library. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages pp. 336–345. IEEE, 2016.

[26] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, and Ankur Dave. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, 2012.

[27] Justin Kestelyn. Putting Spark to Use: Fast In-Memory Computing for Your Big Data Applications — Cloudera Developer Blog, 2013. URL `http://blog.cloudera.com/blog/2013/11/putting-spark-to-use-fast-in-memory-computing-for-your-big-data-applications/`.

[28] Apache Software Foundation. Apache Spark - Lightning-fast cluster computing, 2015. URL `http://spark.apache.org/`.

[29] Kian-Lee Tan, Qingchao Cai, Beng Chin Ooi, Weng-Fai Wong, Chang Yao, and Hao Zhang. In-memory Databases. *ACM SIGMOD Record*, vol.44(no. 2):pp. 35–40, 2015.

[30] Cheqing Jin, Yangxin Kong, Qiangqiang Kang, Weining Qian, and Aoying Zhou. Benchmarking in-memory database. *Frontiers of Computer Science*, vol.10(no. 6):pp. 1067–1081, 2016.

[31] Hector Garcia-Molina and Kenneth Salem. Main Memory Database Systems: An Overview. *IEEE Transactions on Knowledge and Data Engineering*, vol.4(no. 6):pp. 509–516, 1992.

[32] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. PIM-enabled instructions.

In *Proceedings of the 42nd Annual International Symposium on Computer Architecture - ISCA '15*, pages pp. 336–348, New York, New York, USA, 2015. ACM Press.

[33] Eric Frenkiel. Want to Rise Up in Your Organization? Real-Time Big Data Analytics Is the Wave to Catch - Database Trends and Applications. *Database Trends and Applications*, vol.28(no. 4):pp. 27, 2014.

[34] Alex F R Trajano and Marcial P Fernandez. Two-phase load balancing of In-Memory Key-Value Storages through NFV and SDN. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages pp. 409–414. IEEE, 2015.

[35] Olaf Acker, Florian Gröne, Adrian Blockus, and Carsten Bange. In-memory analytics strategies for real-time CRM. *Journal of Database Marketing & Customer Strategy Management*, vol.18(no. 2):pp. 129–136, 2011.

[36] R Bärenfänger, B Otto, and H Österle. Business value of in-memory technology-Multiple-case study insights. *Industrial Management and Data Systems*, vol.114(no. 9):pp. 1396–1414, 2014.

[37] Lee Garber. Using in-memory analytics to quickly crunch big data. *Computer*, vol.45 (no. 10):pp. 16–18, 2012.

[38] IBM. What is In-memory computing, 2017. URL `https://www.ibm.com/analytics/us/en/technology/data-warehousing/`.