**SAKARYA UNIVERSITY**
**INSTITUTE OF SCIENCE AND TECHNOLOGY**

# REAL TIME STREAM PROCESSING FOR INTERNET OF THINGS

**M.Sc. THESIS**

**Hina JAMIL**

| | | |
|---|---|---|
| **Department** | **:** | **COMPUTER AND INFORMATION ENGINEERING** |
| **Supervisor** | **:** | **Prof. Dr. Celal ÇEKEN** |

**June 2018**

SAKARYA UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY

# REAL TIME STREAM PROCESSING FOR INTERNET OF THINGS

M.Sc. THESIS

Hina JAMIL

| | | |
|---|---|---|
| Department | : | COMPUTER AND INFORMATION ENGINEERING |

This thesis has been accepted unanimously / with majority of votes by the examination committee on 28.06.2018.

| | | |
|---|---|---|
| [illegible] | [illegible] | [illegible] |
| Prof.Dr. | Assoc.Prof.Dr. | Assist.Prof.Dr. |
| Celal ÇEKEN | Ali ÇALHAN | Seçkin ARI |
| Head of Jury | Jury Member | Jury Member |

## DECLARATION

I, hereby declare that this project neither as a whole nor as a part there of has been copied out from any source. It is further declared that I have developed this project and the accompanied report entirely on the basis of my personal efforts made under the sincere guidance of my supervisor. No portion of the work presented in this report has been submitted in the support of any other University or Institute of learning, if found we shall stand responsible.

Hina Jamil
29.05.2018

## ACKNOWLEDGEMENT

First of all, I want to thank YTB for giving me scholarship to study in Turkey. Through this scholarship, I pursue my Master’s Degree at Sakarya University. Additionally, I want to thank all the teachers at the department of Computer and Information Technology, as well as the administrative staff for their support throughout the Masters degree.

I would like to thank Dr Celal Çeken for supervising me during this thesis. Through his guidance, I was able to complete my thesis. Thanks a lot.

## TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

## LIST OF SYMBOLS AND ABBREVIATIONS

| | | |
|---|---|---|
| IoT | : | Internet of Things |
| RFID | : | Radio-Frequency Identification |
| BLE | : | Bluetooth low energy |
| M2M | : | Machine to Machine |
| AC | : | Air Conditioner |
| AMPQ | : | Advanced Message Queuing Protocol |
| DBMS | : | Database Management System |
| DSMS | : | Data Stream Management System |
| FIFO | : | First In First Out |
| CEP | : | Complex Event Processing |
| SQL | : | Structured Query Language |
| API | : | Application Programming Interface |
| RDD | : | Resilient Distributed Dataset |
| CQL | : | Cassandra Query Language |

# SUMMARY

Keywords: Iot, Big data, Real time stream processing, Apache Spark

With the increase in popularity of IoT among enterprises, the research and development in the field of monitoring and analysing IoT data has been increased. Iot, being one of the major sources of big data is getting attention from data engineers. The main challenge is real time stream processing of large amount of IoT events. It includes data transfer, storing, processing and analyzing large scale of data in real time. Billions of IoT devices generate huge amount of data that should be analyzed for deriving intelligence in real time.

In this thesis, a unified solution for real time stream processing for IoT is proposed. In the proposed method, sample IoT events of weather station data are generated using Apache Kafka and published to a topic. This data is consumed by Apache Spark consumer which converted it into RDDs. Using Spark SQL, data frames are generated, on which different queries are applied to analyse the data. Data is saved to Cassandra and Zeppelin notebook is used to visualize the data. Logistic Regression algorithm is applied on a data set to make predictions in real time using machine learning library in Spark. In the end, the whole method is speed up by altering different metrics and reducing delay. Results show that this method provides a complete solution to process large IoT data sets in real time.

# NESNELERİN İNTERNETİ İÇİN GERÇEK ZAMANLI AKIŞ İŞLEME

## ÖZET

Anahtar kelimeler: Nesnelerin İnterneti, Gerçek Zamanlı Akış İşleme, Apache Spark

Nesnelerin İnterneti 'nin işletmeler arasında popülerliğinin artmasıyla, izleme ve analiz IoT verilerinin araştırılması ve geliştirilmesi artmıştır. Büyük veri kaynaklarından biri olan Nesnelerin interneti, veri mühendislerinden dikkat çekiyor. Asıl zorluk, büyük miktarda IoT olayının gerçek zamanlı akış işlemesidir. Veri transferini, büyük ölçekli verileri gerçek zamanlı olarak depolamayı, işlemeyi ve analiz etmeyi içerir. Milyarlarca IoT cihazı, istihbaratı gerçek zamanda elde etmek için analiz edilmesi gereken çok miktarda veri üretir.

Bu tezde, IoT için gerçek zamanlı akış işlemek için birleştirilmiş bir çözüm önerilmiştir. Önerilen yöntemde, hava istasyonu verilerinin IoT olayları Apache Kafka kullanılarak üretilir ve bir konuya yayınlanır. Bu veriler Apache Spark tüketicisi tarafından tüketilmekte ve RDD'ye dönüştürülmektedir. Spark SQL'i kullanarak, verileri analiz etmek için farklı sorguların uygulandığı veri çerçeveleri oluşturulur. Veriler Cassandra'ya kaydedilir ve Zeppelin notebook verileri görselleştirmek için kullanılır. Spark'deki makine öğrenme kütüphanesini kullanarak gerçek zamanlı tahminler yapmak için bir veri kümesine Lojistik Regresyon algoritması uygulanır. Sonunda, tüm ölçüm farklı metrikleri değiştirerek ve gecikmeyi azaltarak hızlanır. Sonuçlar, bu yöntemin gerçek zamanlı olarak büyük IoT veri kümelerini işlemek için eksiksiz bir çözüm sunduğunu göstermektedir.

# CHAPTER 1. INTRODUCTION

Today, different platforms are generating huge amount of data per milliseconds which leads to a separate term referred as big data. Organizations are using big data in decision making by analyzing and storing this massive amount of data. Normally, big data is referred as 3V which means the data that is huge in volume generating with high velocity and has varieties like text, image, video etc [1]. Entrepreneurs and businesses have realized the importance of big data and are using the human resources and technologies to get benefits from it. The main key to derive value from big data is to analyze it. Analysis can be done on historical data known as batch processing or real data know as real time processing. Social media platforms such as Twitter and Facebook, online shopping sites like ebay, cloud platforms, and banking sectors etc are the sources of big data. Along with them IoT (Internet of Things) is one of the major sources of big data.

IoT is an emerging field in the recent years. It is a platform that has potential to turn the world into a smart world. IoT is an ecosystem that includes physical devices such as sensors connected to internet. It allows the monitoring of physical devices over internet. It is a collection of objects or sensors that can sense the surroundings, communicate with other devices using RFID, BLE or Z-wave etc and give response to situations without human intervention. These objects can be located and controlled from anywhere anytime. Apparently M2M technology will reach exponentially to 3.3 billion by 2021 [2]. IoT is playing major role in medical field, security, energy efficiency, military and agriculture applications etc. Some of the major applications of IoT are health monitoring, smart cities, traffic monitoring, wearable devices, weather monitoring, and a lot more. AC of the room can be turned on for cooling before entering the home. Mobile phones can trigger a signal if someone enters the home while you are outside. Users can find available parking on their mobiles, using traffic sensors. A child at school can be monitored using wearable devices

Along with great benefits it offers, there are numbers of challenges in the field of IoT. One of them is real time stream processing of large amount of IoT events. In my thesis, I have provided a method for real time stream processing of IoT. It will create direct streams of IoT events and then process it in real time. Sample weather station data is ued as a use case. Logistic Regression machine learning algorithm has been applied on a data set to make decisions in real time. I have used Apache Kafka, Apache Spark, Cassandra and zeppelin frameworks. Results show that this model provides the complete solution to IoT data processing in real time with large amount of data.

Contribution of this study can be summarized as follows.

- Billions of IoT devices generate huge amount of data that should be analyzed for deriving intelligence in real time.
- Processing of all large amounts of IoT data in real time is a big challenge.
- A unified solution for processing streams of IoT data in real time is proposed.
- It includes data transfer, storing, processing and analyzing data.
- Logistic Regression algorithm is applied in real time to make predictions.
- The results show that the proposed system can process large data sets in real time.

So far chapter 1 includes introduction and background of the topic addressed. In chapter 2 related works is presented. Chapter 3 includes description of technologies used. Chapter 4 contains implementation of proposed model. Chapter 5 includes results and analysis. In chapter 6, a conclusion has been described.

# CHAPTER 2. LITERATURE REVIEW

In [3], a method to perform real time streaming for iot data is presented so that the dynamic integration into the web using AMPQ can be supported. The performance of smart city can be improved by delivering of large amount of data using this method. The proposed system is evaluated based on data size and average exchanged message time using summarized and raw sensor data.

In [4], a new prototype NEPTUNE is proposed to achieve high throughput during real time stream processing. The framework improves the CPU utilization by the use of thread-pools and batch processing. Memory pressure is controlled by the efficient reuse of objects. The performance is compared with apache storm and a higher processing rate is achieved as compared to storm. In [5] a system is proposed to enable data streams with adaptive processing, aggregation and federation, adding semantic annotations to data in order to process large streams of iot data. It also addresses the issues and solutions for ensuring reliable processing and smart adaption in smart city system.

In [6], a survey of existing systems related to real time stream processing is presented and related issues are addressed. Then a new framework called information flow of things (IFoT) is suggested that perform real time stream processing in a distributed manner among Iot devices. In [7], a middleware for IoT devices has been proposed to process the data stream locally using resources of IoT devices in real time. It divides the tasks into subtasks and each sub task is executed on multiple IoT devices in a distributed manner. Real time analysis of data stream is performed. Raspberry Pie has been used to implement prototype of system.

Some large companies like IBM Watson IoT, Microsoft Azure IoT Suite, AWS IoT and Google Cloud IoT have provided cloud based frameworks for IoT [8]. Some others like Kaa and macchina.io are open source and public solutions for IoT. In [9] researchers have presented a framework that provides solution to high data rate, different types of data and high volume of data for IoT by storing the data coming from different IoT resources. The structured data is being stored using both traditional DBMS and noSQL databases. Unstructured data is stored in a file system using name conventions and timestamp related to connected device. However, there can be high latency due to usage of file system which can be avoided.

As explained in [10], FIFO queues are represented in the form of graphs to show the flow of data stream. It consists of operators that perform INPUT/OUTPUT operations on those FIFO queues. Operators are actually transformers that transform the streams into another stream. The transformer must contain the necessary function to carry out transform function like initialization, activation and output rate etc. In these methods, Data Stream Management System (DSMS) are used, that deals directly with data streams. On each data arrival, DSMS updates its records unlike DBMS that updates the data when query is applied.

In [11], a new concept of using “cognition” and “proximity” is proposed by the researchers to use the most relevant device in order to solve the heterogeneity problem of connected smart devices. For this, the IoT devices are considered as virtual devices by introducing the concept of virtual sensors. A smart city application has been used as a case study that further encompasses the wide range of IoT applications. This framework helps in managing interested services and application besides controlling the communication between smart objects. Cognition technique is used to establish an interaction between virtual devices and real devices by using optimization techniques. Whereas, proximity determines the selection of smart object, that will be used to entertain particular IoT application.

Another technique that performs condition based detection of events, determine by data streams [10], [12] is called Complex Event Processing (CEP) in which

Publisher/subscriber systems have been used to employ characteristics based event processing including interest selection and attributes. However such systems become complex when there are multiple sensors. Sensor networks, information and business management, network monitoring and control systems are some of the areas where CEP has been used extensively. Java, R and Python are the languages supported by CEP however, SQL based CEP are mostly used.  There are many platforms available for stream processing; some of them are SAP Sybase Event Stream Processor, IBM Infosphere Stream [13], Yahoo!S4, SQL Stream Blaze, Microsoft Stream Insight, TIBCO Business Events, Amazon AWS IoT, Apache Storm [14] and Apache Spark [15].

Machine learning has become an important feature in IoT platforms to perform analysis and obtain decisions in real time. Many of the platforms mentioned before, supports real time or on-line machine learning methods. For example, spark has its own built in Machine Library (ML) or Mlib and AWS has Amazon Machine Learning to apply machine learning algorithms on streams of data for analysis in real time.

# CHAPTER 3. TECHNOLOGIES USED

In this section, there is a thorough description of the technologies or platforms used in the project. These are Apache Kafka, Apache Spark, Cassandra and Zeppelin.

## 3.1. Apache Kafka

Apache Kafka [16], a distributed open source streaming platform was developed at LinkedIn by Apache Software Foundation in 2011. Initially it was a messaging queue system, which later evolves into a complete streaming platform. It is written in Scala and Java. It provides low latency, fault-tolerant, high throughput publish and subscribe pipelines to handle data in real time. This makes it highly popular among enterprises that deal in stream processing. Kafka Connects provides connection with external systems as well.

### 3.1.1. Kafka Core APIs

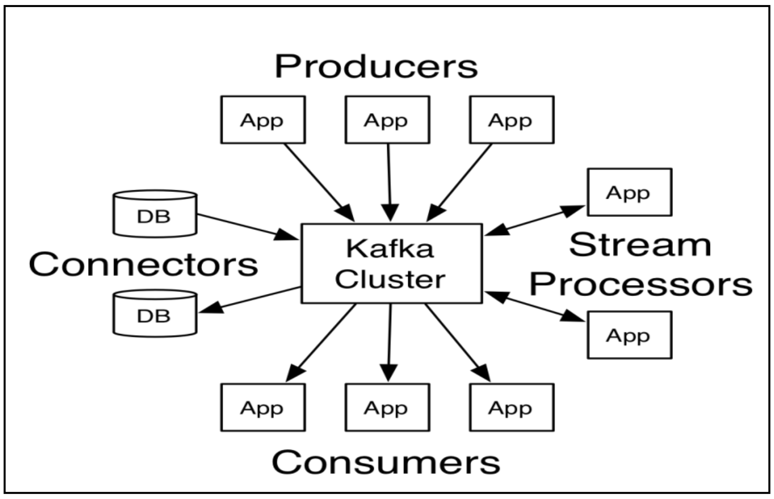


Figure 3.1. Kafka APIs [16]

In Figure 3.1. Kafka's four APIs has been shown. These are as follows.

- Producer API

  It publishes the record to a topic.
- Consumer API

  It subscribes to a topic and consumes records from that topic to process.
- Streams API

  It allows applications to consume and produce input and output streams from the topics.
- Connector API

  It allows connecting to external systems such as database.

### 3.1.2. Kafka Architecture

A typical Kafka framework consists of producers, a server in Kafka cluster called broker and a consumer as shown in Figure 3.2.

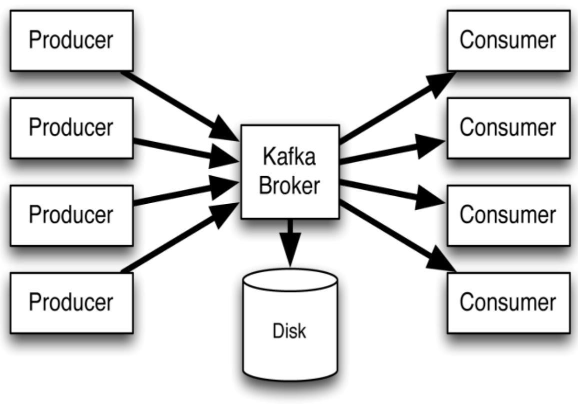


Figure 3.2. Kafka Architecture [17]

Kafka cluster consist of brokers which are actually the servers. The topics are partitioned among different nodes of a cluster. Different processes called producers publishes the data to a topic. Data is distributed to the different partitions of a topic. The messages are recognized by their offsets in the partition. The data from the topic is read by the processes called consumers for further processing.

## 3.2. Apache Spark

Apache Spark [15] is an open source big data processing and analytic engine based on cloud computing. It was initially developed by Berkeley's Lab, but later taken by Apache Software Foundation. It has become a popular big data tool among different kinds of industries e.g. Yahoo, Netflix, and eBay etc. It supports Scala, Java, R and Python languages. It is 100 times faster than Hadoop which uses mapReduce model.

Spark introduces the concept of Resilient Distributed Dataset (RDD). RDDs are distributed among cluster to perform parallel processing. This is the core concept in Spark's fast processing. Different data formats can be converted into RDDs including Json strings, text files and SQL databases. Spark splits the application into multiple tasks that are distributed among different processes called executors. Number of executors may vary depending upon the requirement.

### 3.2.1. Spark Architecture

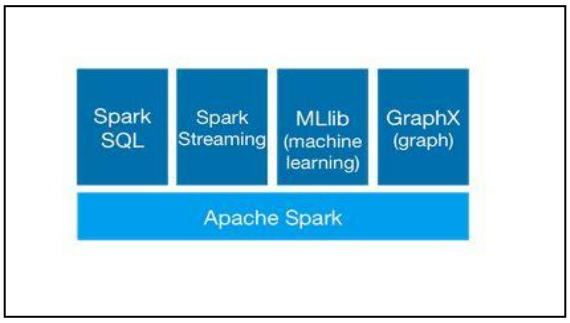


Figure 3.3. Spark Architecture [15]

Figure 3.3 shows that Spark ecosystem consists of following parts.

- Spark SQL

  Spark SQL introduced the concept of data frames that provides SQL language support for quering data. It support semi structured and structured data.

- Spark Streaming

  It allows analyzing and processing both batch data and real time streams of data.
- Mlib

  Machine learning library allows using machine learning algorithms on data for predictions and making decisions.
- GraphX

  It allows to build, transfer and process graph data.
- Spark Core

  All above functions are built on the top of spark core. It provides in memory processing to support wide range of application.

## 3.3. Apache Cassandra

Apache Cassandra [18] is an open source database system that has its own query language called Cassandra Query Language (CQL). It supports fault tolerance, high availability and scalability that make it a perfect choice for critical data. Many large companies including Netflix, Reddit, GitHub and eBay etc are using Cassandra for their large data sets. It is a cluster based platform in which the data is distributed among different nodes of cluster. Every node in the cluster is identical and there is no master node. Replication of data is supported among multiple nodes for fault tolerance. New machines can be added to a cluster making it more scalable. Data is organized in tables as a key-value pair. The primary key of the table is the partition key, which has a value as columns. The tables can be added, deleted and changed at run time.

## 3.4. Apache Zeppelin

Zeppelin [19] is an online multipurpose notebook that enables user friendly data analysis by representing data into graphs, charts and tables. It is an analytical tool that supports data visualization. Zeppelin has an interactive interface that eases the task of developing, managing, processing, sharing and visualizing data. There is a plug-in called Zeppelin Interpreter that uses a specific data processor tool or a language. It supports different interpreters including Spark, Cassandra, python, Flink etc. It also provides the facility to download, print or send the report of your insights. Zeppelin supports multi users on a cluster and can be run in personal mode as well.

# CHAPTER 4. PROPOSED MODEL

The proposed model is applied on a use case of “Weather Station” data. Sample IoT data of 1000 weather stations is generated with Kafka producer and sent to Spark consumer. Spark performs transferring, processing and querying on IoT data using spark SQL and spark streaming. It also applies machine learning algorithms on a dataset to make decisions using Spark ML.  Spark saves the data in Cassandra database. With CQL, queries are applied on the data. The data from Cassandra table is accessed in Zeppelin which provides visual representation and analysis. The whole method is speed up using more than one broker in Kafka and increased number of work nodes in Spark. Figure 4.1 shows the proposed model.

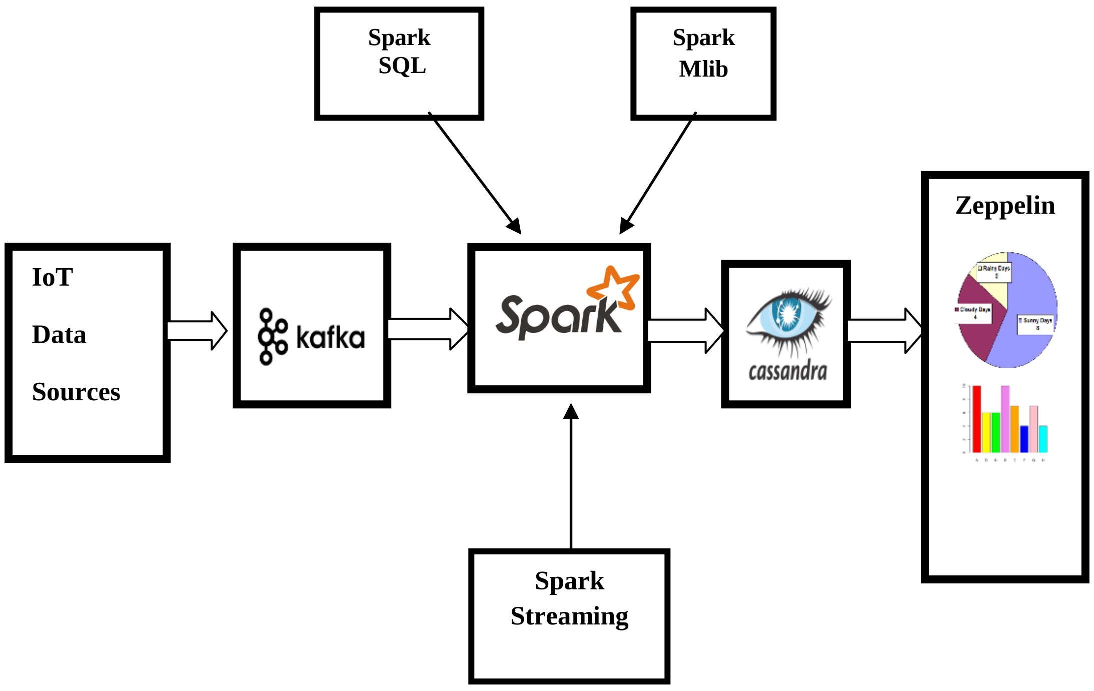


Figure 4.1. Proposed model

## 4.1. Use Case

Weather monitoring is a very useful application of IoT. Weather of different places is sensed by sensors like temperature sensor, rain sensor, humidity sensors etc and sent to online platforms which keep us updated about today's and tomorrow's weather condition. Consider the weather data is arriving from all the cities of a country and we want to find the city with hottest or lowest temperature in real time. The proposed model is applied on a sample IoT data from different weather stations to get these results. To make this example more useful, machine learning algorithm is applied to decide if the weather is suitable for outdoor games or not.

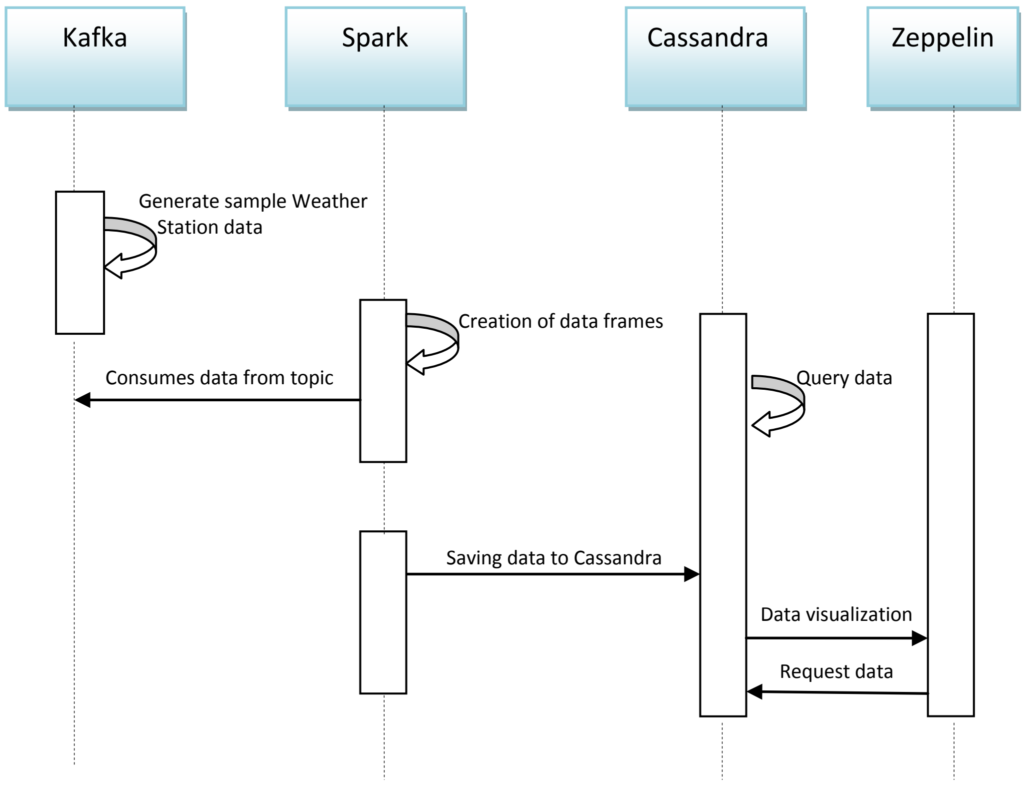


Figure 4.2. Sequence diagram

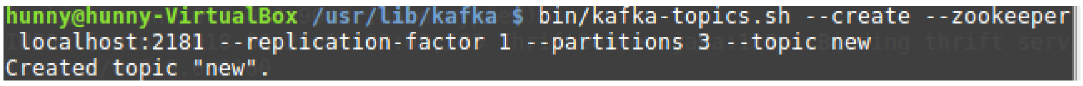
Sequence diagram of a use case is shown in Figure 4.2. Sequence diagram describes the flow of project. It shows the interaction between objects that work together. It is also known as event diagram as it describes the functionality of scenario with time.

## 4.2. Weather Monitoring Application

The project is divided into two modules which are separate Maven projects build in Scala that can run independently. The first module is data Iot Data Producer that will generate Iot data events and send to second module which is data processing. Data processing module will collect process and save the data.

### 4.2.1. IoT Data Producer

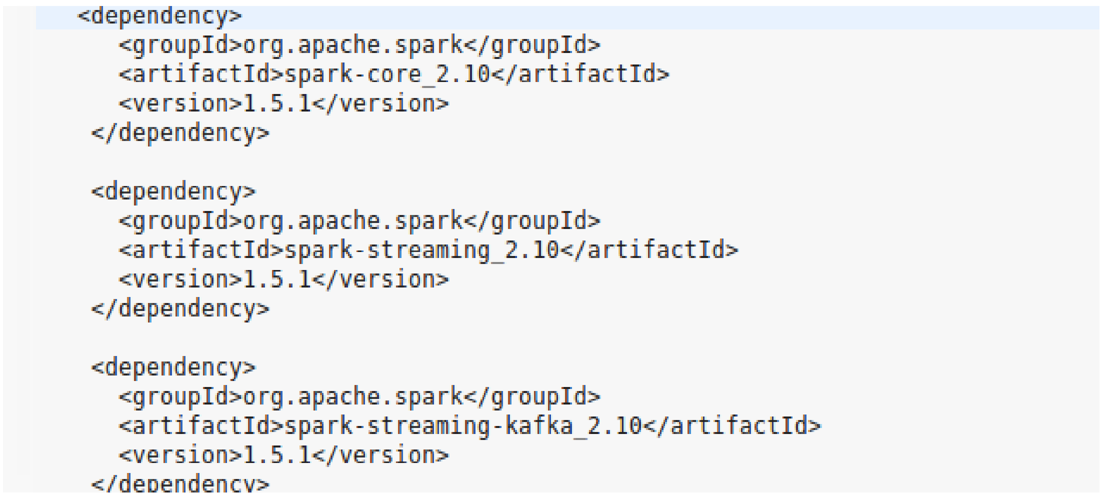
IoT Data Producer is a Kafka producer build in Scala that generates sample IoT data and publish it to a topic. First we need to create a topic on Kafka describing the topic name, zookeeper server address, replication factor that describes the number of servers on which the topic will be replicated and number of partitions as shown in Figure 4.3.



```
hunny@hunny-VirtualBox /usr/lib/kafka $ bin/kafka-topics.sh --create --zookeeper
 localhost:2181 --replication-factor 1 --partitions 3 --topic new
Created topic "new".
```

Figure 4.3. Topic creation

#### 4.2.1.1 Maven Dependencies



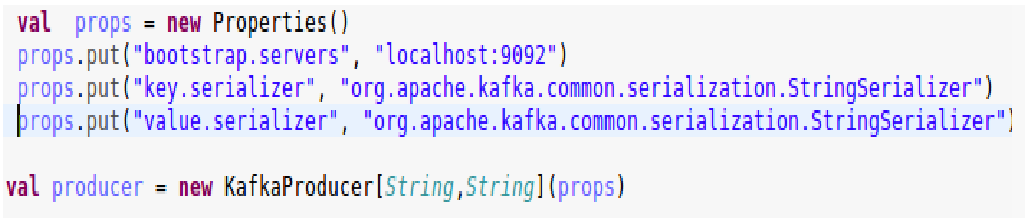
```
     <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.10</artifactId>
        <version>1.5.1</version>
      </dependency>

      <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming_2.10</artifactId>
        <version>1.5.1</version>
      </dependency>

      <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming-kafka_2.10</artifactId>
        <version>1.5.1</version>
      </dependency>
```

Figure 4.4. Maven dependencies

As shown in Figure 4.4., a maven project is created and apache spark, spark-streaming and Kafka dependencies are added to a pom.xml file of the project. Maven helps in building projects by storing libraries in a cache, which are downloaded from repositories.

### 4.2.1.2. Configurations of Kafka

First zookeeper and Kafka servers should start on console in the background. Kafka and zookeeper configuration properties are added.



```
 val  props = new Properties()
 props.put("bootstrap.servers", "localhost:9092")
 props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
 props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer"]

val producer = new KafkaProducer[String,String](props)
```

Figure 4.5. Kafka configuration code

In Figure 4.5. bootstrap.servers provides the list of initial hosts that are the addresses of Kafka servers in a cluster. “Key.serializer” and “value.serializer” define the instructions to convert key and value objects into bytes.

### 4.2.1.3. Generating and Sending IoT data

A new object of Kafka producer is created. Case class is used to define the schema of weather data. It includes the value of station id, temperature, and pressure and rain probability. The object of case class is converted into Json format and data is published to the topic using send() function.

```
var first= new data(id,date,temp,press)

val jsonString = Json.toJson(first).toString

val record = new ProducerRecord(TOPIC, "key", jsonString)

producer.send(record)
```

The data being send in Json format is shown in Figure 4.6.

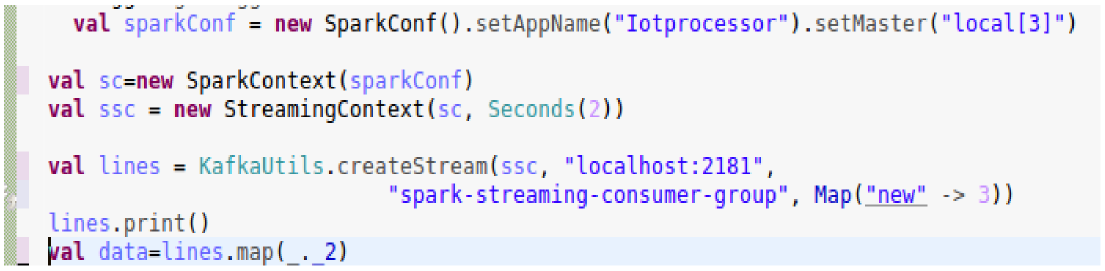
```
{"id":"ZcVoK","dates":"2018-05-16 19:54:28","temp":38,"press":45,"rain_prob":41}
{"id":"WfE8d","dates":"2018-05-16 19:54:30","temp":36,"press":53,"rain_prob":28}
{"id":"wBPae","dates":"2018-05-16 19:54:31","temp":36,"press":61,"rain_prob":23}
{"id":"5coxE","dates":"2018-05-16 19:54:31","temp":57,"press":46,"rain_prob":47}
{"id":"OSBvS","dates":"2018-05-16 19:54:31","temp":26,"press":37,"rain_prob":89}
```

Figure 4.6. IoT data in Json format.

### 4.2.2. Data Processor

In Data Processor section, spark will consume or read the data from the Kafka's topic and generate streams of data using spark streaming. It processes the data, apply queries and save the data in Cassandra table. It also applies machine learning algorithm on data in real time.

#### 4.2.2.1. Configuration and RDD creation:



```
  val sparkConf = new SparkConf().setAppName("Iotprocessor").setMaster("local[3]")

val sc=new SparkContext(sparkConf)
val ssc = new StreamingContext(sc, Seconds(2))

val lines = KafkaUtils.createStream(ssc, "localhost:2181",
                          "spark-streaming-consumer-group", Map("new" -> 3))
lines.print()
val data=lines.map(_._2)
```

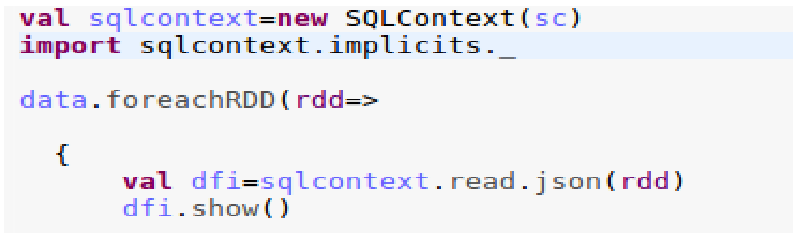
Figure 4.7. Configuration of Spark

According to Figure 4.7., in the configuration code "local [2]" describes that two threads have started on a local machine for this job. Variable "Ssc" is an object of streaming context and "localhost: 2181" is an address of zookeeper. Map function will consume the data from Kafka topic and show the data in key value pair as Rdd. Streams of data will be displayed on console after every 10 seconds. Function "map

(_._2)" will consider the second tuple of key pair value which means it will consider the value part and ignores the key.

### 4.2.2.2. Data Frames

Using SparkSQL the data can be converted in to data frames which are more advance form of RDD. Data frames are easy to process as they are similar to a table in database. SQL queries can also be applied on data frames. Following code shows the creation of data frames.



```
val sqlcontext=new SQLContext(sc)
import sqlcontext.implicits._

data.foreachRDD(rdd=>

  {
       val dfi=sqlcontext.read.json(rdd)
       dfi.show()
```

Figure 4.8. Data frame code

According to Figure 4.8. for using sparkSQL, first need to create an object of SQLcontext. To use all the functions of SQLcontext, sqlcontext.implicits._ is imported. "sqlcontext.read.json(rdd)" will convert the Json Rdd into data frames which are shown in Table 4.1. where data is shown in the form of tables. Each value of Rdd is represented by a column in a Table.

| Dates | Id | Press | Rain_prob | Temp |
|---|---|---|---|---|
| 2018-05-16 21:56:48 | A5v06 | 62 | 64 | 24 |
| 2018-05-16 21:56:48 | Q6Y0f | 30 | 58 | 36 |
| 2018-05-16 21:56:49 | E21gX | 15 | 28 | 1 |
| 2018-05-16 21:56:49 | acTJx | 53 | 30 | 35 |
| 2018-05-16 21:56:49 | 6s00X | 17 | 26 | 21 |
| 2018-05-16 21:56:49 | Ze84J | 51 | 79 | 39 |
| 2018-05-16 21:56:49 | hGdzc | 24 | 69 | 30 |

Table 4.1. Data frames of IoT data

In Table 4.1., the weather station data is shown in the form of table. It includes the date and time of each event, unique id for each station, temperature, air pressure and rain probability values from each station.

### 4.2.2.3. Saving to Cassandra

First the table is created in Cassandra database whose structure should be similar to the schema of data frame.

First keyspace is created which defines data replication on nodes in a cluster. “Simple Strategy” means that there is a simple replication for a cluster. “Replication_factor” depends on the number of nodes used. Table is created in Cassandra as follow.

```
CREATE TABLE weather_st (station_id varchar PRIMARY KEY, dates varchar,
temp int, press int, rain_prob int);
```

After creating table, the data will be send and save to Cassandra using Spark-Cassandra connector.

```
dfi.write.format("org.apache.spark.sql.cassandra").options(Map("table"-
>"weather_st","keyspace"->"data")).save()
```

In this way, the data frames will be directly save to Cassandra using Spark-Cassandra connector.

### 4.2.2.4. Machine Learning Algorithm

Machine learning (ML) has become the priority for businesses that helps them in decision making. It has also become a major part in IoT and big data frameworks. Applying ML algorithms on IoT data streams in real time, helps in taking action at the right time. Spark has built in ml library that can be applied directly to data frames. It includes methods of classification and regression, clustering, collaborative filtering etc that can be applied depending on the requirement of application.

As mentioned before, I have considered a scenario in which decision will be made for outdoor sports. If the temperature will be greater than 25 degrees and rain probability will be more than 50%, the condition for outdoor sports is bad. Otherwise it is good. As this scenario is a selection between two conditions, classification algorithm is the most suitable one. Logistic Regression has been applied to a data set which will determine the output. The algorithm is first trained with sample data.

#### 4.2.2.4.1. Logistic Regression

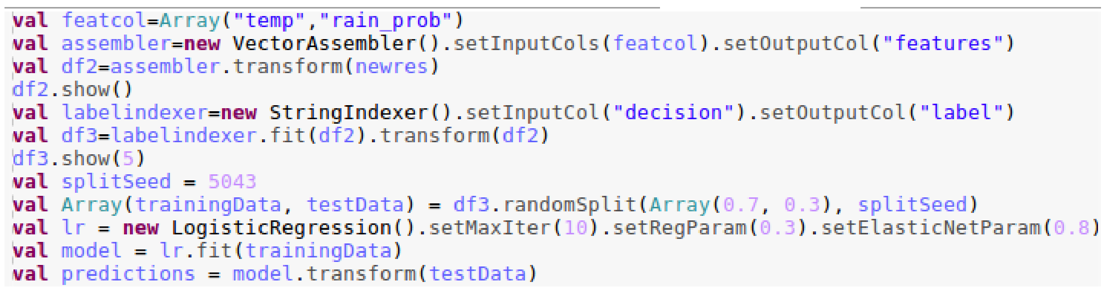
Logistic regression [20] is a classification model designed for a set of binary variable. These binary variables represents two conditions such as sick/healthy, good/bad, pass/fail or alive/dead. The conditions are normally labelled as “0” and “1”. It is a predictive analysis method that describes the relation between one dependent and one or more independent variables. It is used to find the probability of occurrence of an event based on predictors. Logistic function is defined in Equation 3.1.

$$ln\left(\frac{P_i}{1-P_i}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n \tag{3.1}$$

Where P is the probability of event, ln is natural algorithm, $\beta_0$ is the intercept, $\beta_i$ are the regression coefficients multiplied by $x_i$ explanatory variables. After applying exponential on both sides, equation becomes as follows.

$$\left(\frac{P_i}{1-P_i}\right) = e^{b_0 + b_i x_i} \tag{3.2}$$

### 4.2.2.4.1.2. Logistic Regression on data frames



```
val featcol=Array("temp","rain_prob")
val assembler=new VectorAssembler().setInputCols(featcol).setOutputCol("features")
val df2=assembler.transform(newres)
df2.show()
val labelindexer=new StringIndexer().setInputCol("decision").setOutputCol("label")
val df3=labelindexer.fit(df2).transform(df2)
df3.show(5)
val splitSeed = 5043
val Array(trainingData, testData) = df3.randomSplit(Array(0.7, 0.3), splitSeed)
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val model = lr.fit(trainingData)
val predictions = model.transform(testData)
```

Figure 4.9. Logistic regression code

Figure 4.9. shows the code of Logistic regression. In this code, first the columns that will determine the value of prediction is selected which in this case are temperature and rain_probability. Feature vector is created from these columns and input and output column name is selected. Create a label column based on decision column with transform method. These transformations are the pipelines that will give us the required data frame on which the logistic regression model will be applied. Next, split the data, use 70% for training the model with historical data and 30% to test the data without labels. The model is trained using elastic net regularization. Predictions are made using test data. The whole method is shown in Figure 4.10.

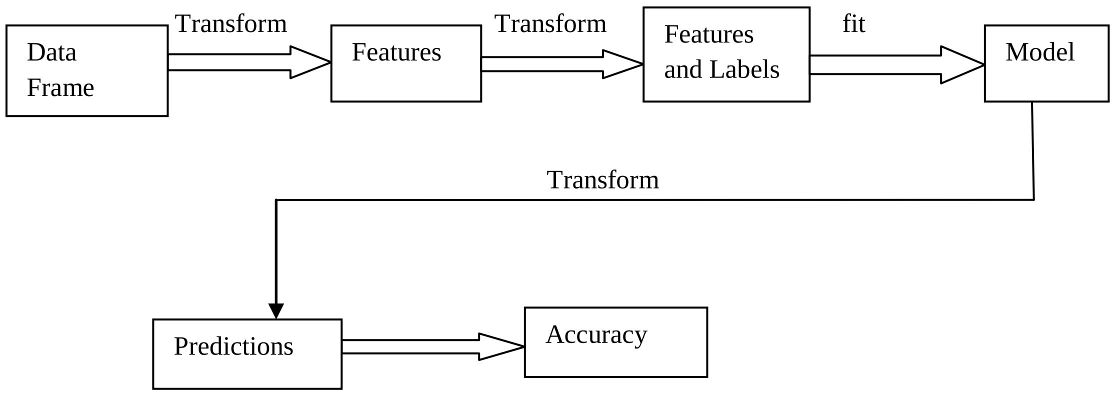


Figure 4.10. Logistic Regression Model

# CHAPTER 5. RESULTS AND COMPARISON

In this section results have been shown that are derived from the proposed model. First of all queries applied to the data frames in real time using SparkSQL. For example, to select and show stations where temperature value is greater than 30. The result is shown below in Table 5.1.

| ID | Temp |
|---|---|
| Q6YOf | 36 |
| acTJx | 35 |
| Ze84J | 39 |
| asRjX | 42 |
| fMZps | 35 |
| kycbd | 31 |
| LiVQw | 41 |

Table 5.1. Spark SQL query

Zeppelin notebook is used to analyse data in visual form. With Cassandra interpreter, the data that is saved from spark to Cassandra is retrieved by applying queries and result is shown in graphical form as shown in Figure 5.1.

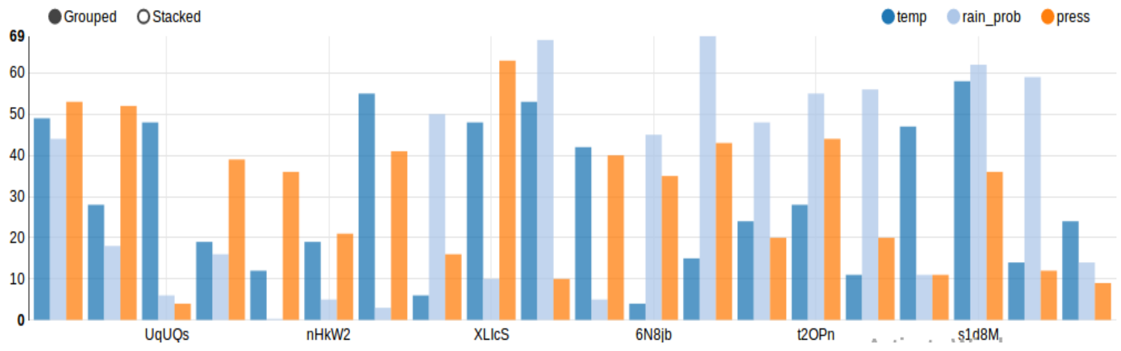


Figure 5.1. Graph of IoT data

In Figure 5.1 the bar chart is shown with the weather station ids along with the values of temperature, pressure and rain_probability represented by dark blue, orange and light blue bars respectively.

## 5.1. Machine Learning Algorithm Results and comparison

By applying Logistic Regression on a data set to find if the weather condition for outdoor sports is good or not, following results have achieved.

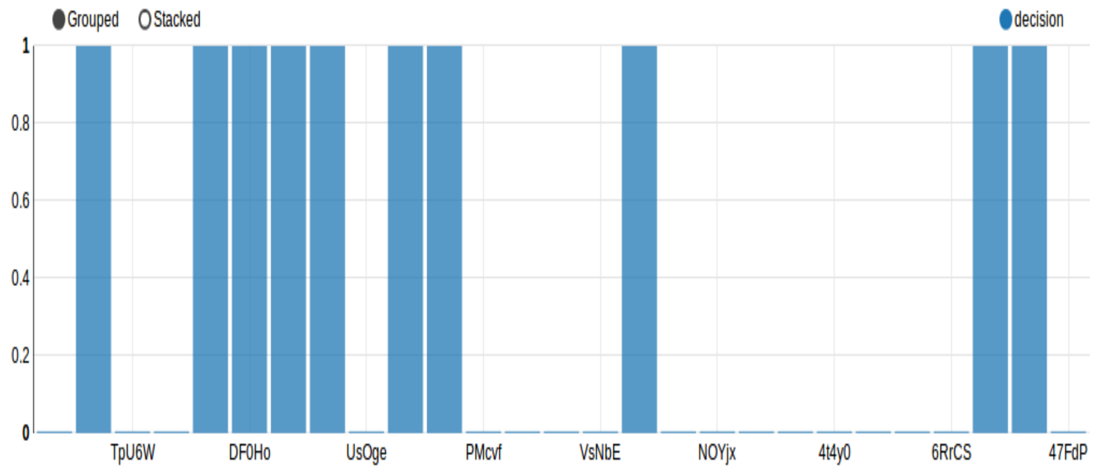


Figure 5.2. Decisions made for stations

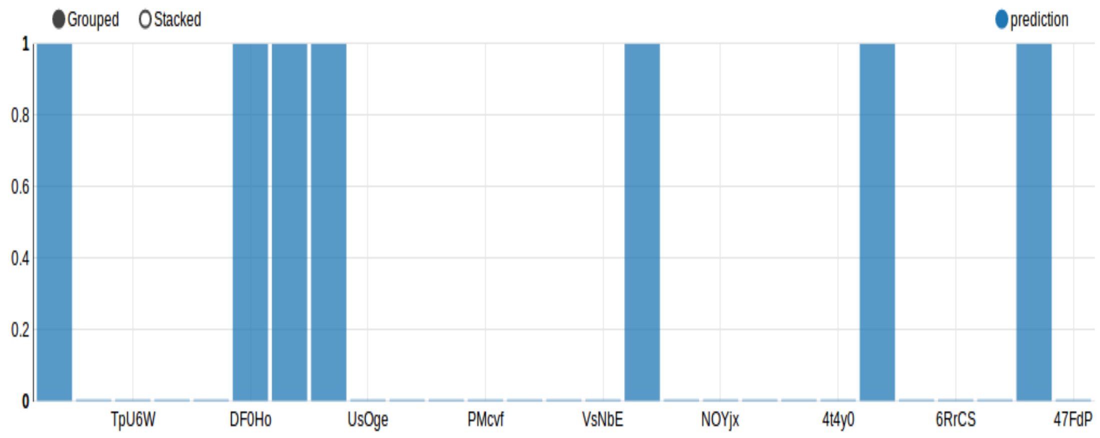


Figure 5.3. Predictions made for stations

Figure 5.2 shows the decisions made for each station that if the weather is suitable for outdoor sports or not. Value ‘1’ shows the weather is good and ‘0’ shows the weather is not good. This decision is based on the condition that if the temperature value is more than 30 degrees and the rain_probability is more than 50%, the weather is not good and decision will be 0. Otherwise decision will be 1. In Figure 5.3., the predictions made on test data using Logistic Regression has been shown. A comparison between above graphs shows the predictions are 74 % similar to decisions. After applying this algorithm on series of data following accuracy values have been achieved.

| **Accuracy** | 0.944 | 94,4% |
|---|---|---|
| **Accuracy** | 0.833 | 83,3% |
| **Accuracy** | 0.792 | 79,2% |
| **Accuracy** | 0.971 | 97,1% |
| **Accuracy** | 0.841 | 84,1% |

Table 5.2. Accuracy of algorithm

Table 5.2. shows the accuracy of algorithm and results. Average value of accuracy is 87.62 which make the algorithm, a suitable choice for this scenario.

## 5.2. Delay Comparison

Initially only single Kafka broker is used with the two threads in spark. The main purpose of this model is to process large amount of data in a minimum time. In order to reduce the delay, the number of Kafka brokers is increased to 3, with the replication factor of topic is 3. In this way, if one broker stops working, the data can still be read and write because the topic is replicated among three brokers. The number of topic partition is also increased to 3. The number of threads in spark is increased to 3 in order to speed up the streaming and processing. A comparison of delay has been made between both scenarios that are shown in figure 5.4. and 5.5.

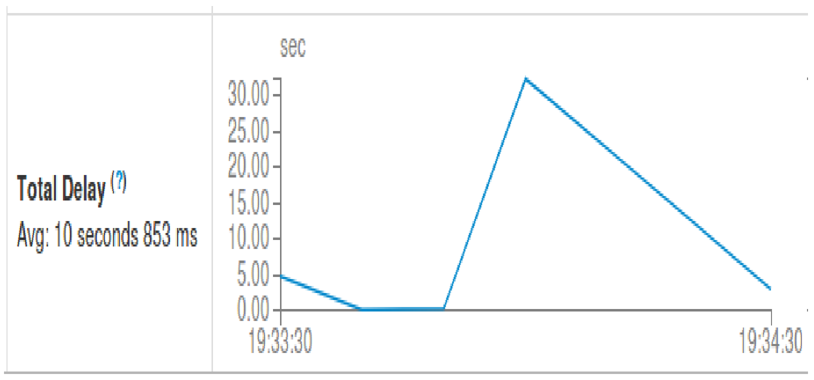


Figure 5.4. Delay with 2 Spark threads

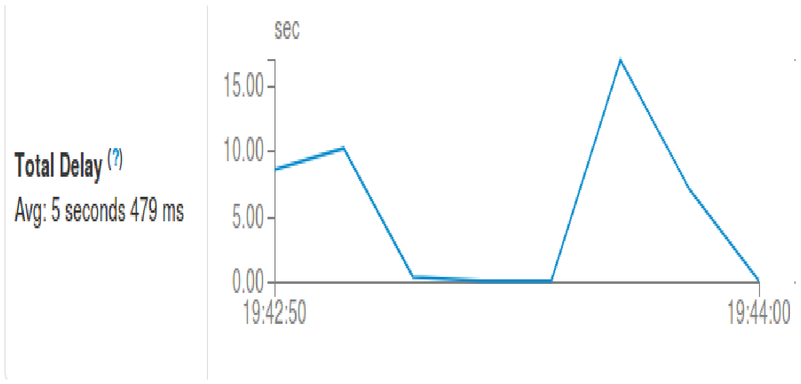


Figure 5.5. Delay with 3 Spark threads

A significant reduction in delay can be observed by comparing Figure 5.4. and Figure 5.5. Average Delay with 1 topic partition and 2 Spark threads is 10 seconds 853 ms while delay with 3 topic partitions and 3 Spark thread is 5 seconds 470 ms.

# CHAPTER 6. CONCLUSION AND FUTURE WORK

In this thesis, a method is proposed for real time stream processing for IoT. In this method, Kafka producer generates sample IoT data and publish it to topic. Spark acting as a consumer, subscribe to that topic and retrieve the data sent from kafka producer. Spark performs transferring, processing and querying on IoT data using spark SQL and spark streaming. Decisions are made in a particular scenario by applying machine learning algorithms on a dataset using Spark ML.  Spark sends the data in Cassandra database. With CQL, queries are applied on the data. The data from Cassandra table is accessed in Zeppelin using Cassandra interpreter. Zeppelin supports data visualization in form of graphs and tables. The whole method is speed up using more than one Kafka broker and by increasing number of threads in Spark. Thus, this model provides the complete solution to IoT data processing in real time with large amount of data.

In future, the same model will be applied using different frameworks and comparing with the proposed one in order to get the fastest solution. The method will be applied to a larger set of data. The application will be made smarter and more prediction based decisions will be made by using machine learning algorithms.

# REFERENCES

[1] Watson, H.J., 2014. Tutorial: Big data analytics: Concepts, technologies, and applications. CAIS, 34, p.65.

[2] Cisco, V.N.I., 2016. Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper. Document ID, 958959758.

[3] Kolozali, S., Bermudez-Edo, M., Puschmann, D., Ganz, F. and Barnaghi, P., 2014, September. A knowledge-based approach for real-time iot data stream annotation and processing. In Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE (pp. 215-222). IEEE.

[4] Buddhika, T. and Pallickara, S., 2016, May. Neptune: Real time stream processing for internet of things and sensing environments. In Parallel and Distributed Processing Symposium, 2016 IEEE International (pp. 1143-1152). IEEE.

[5] Tönjes, R., Barnaghi, P., Ali, M., Mileo, A., Hauswirth, M., Ganz, F., Ganea, S., Kjærgaard, B., Kuemper, D., Nechifor, S. and Puiu, D., 2014. Real time iot stream processing and large-scale data analytics for smart city applications. In poster session, European Conference on Networks and Communications.

[6] Yasumoto, K., Yamaguchi, H. and Shigeno, H., 2016. Survey of real-time processing technologies of iot data streams. Journal of Information Processing, 24(2), pp.195-202.

[7] Nakamura, Y., Suwa, H., Arakawa, Y., Yamaguchi, H. and Yasumoto, K., 2016, June. Design and Implementation of Middleware for IoT Devices toward Real-Time Flow Processing. In Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on (pp. 162-167). IEEE.

[8] Ralhan, P. (2000). Web.njit.edu. Retrieved 2 May 2017, from https://web.njit.edu/~turoff/coursenotes/CIS732/samplepro/prototyping.doc, Access Date: 10.12.2017.

[9] Jiang, L., Da Xu, L., Cai, H., Jiang, Z., Bu, F. and Xu, B., 2014. An IoT-oriented data storage framework in cloud computing platform. IEEE.

[10] Hirzel, M., Soulé, R., Schneider, S., Gedik, B. and Grimm, R., 2014. A catalog of stream processing optimizations. ACM Computing Surveys (CSUR), 46(4), p.46.

[11] Vlacheas, P., Giaffreda, R., Stavroulaki, V., Kelaidonis, D., Foteinos, V., Poulios, G., Demestichas, P., Somov, A., Biswas, A.R. and Moessner, K., 2013. Enabling smart cities through a cognitive management framework for the internet of things. IEEE communications magazine, 51(6), pp.102-111.

[12] Cugola, G. and Margara, A., 2012. Processing flows of information: From data stream to complex event processing. ACM Computing Surveys (CSUR), 44(3), p.15.

[13] Gedik, B. and Andrade, H., 2012. A model- based framework for building extensible, high performance stream processing middleware and programming language for IBM InfoSphere Streams. Software: Practice and Experience, 42(11), pp.1363-1391.

[14] Storm project available from https://storm-project.net/, Access Date: 5.08.2017.

[15] Apache Spark available from https://spark.apache.org/, Access Date: 10.08.2017.

[16] Apache Kafka project available from http://kafka.apache.org/, Access Date: 13.09.2017.

[17] Dunning, T. and Friedman, E., 2016. Streaming architecture: new designs using Apache Kafka and MapR streams. " O'Reilly Media, Inc.".

[18] Apache Cassandra available from http://cassandra.apache.org/, Access Date: 7.3.2018.

[19] Apache Zeppelin available from https://zeppelin.apache.org/, Access Date: 14.4.2018

[20] Logistic regression from https://www.statisticssolutions.com/what-is-logistic-regression/, Access Date: 22.4.2018.

## RESUME

Hina Jamil, was born on 24.08.1992 in Pakistan. She has completed her primary, secondary and College education from Wah Cantt, Pakistan. In 2010, she started her bachelor’s in Telecommunication and Networks from Comsats University and graduated from there in 2014. In 2014, she started working as a research associate in Comsats University.  In 2015, she went to Turkey and completed one year Turkish language course. Currently, she is doing Masters in Computer and Information Engineering from Sakarya University, Turkey.