Scientific Research Publishing

Advances in Internet of Things, 2016, 6, 65-91
http://www.scirp.org/journal/ait
ISSN Online: 2161-6825
ISSN Print: 2161-6817

Design and Implementation of the Sense Egypt Platform for Real-Time Analysis of IoT Data Streams

A. S. Rozik, A. S. Tolba, M. A. El-Dosuky

Faculty of Computers and Information, Mansoura University, Mansoura, Egypt
Email: jdev.cs2011@gmail.com, ast@astolba.com, meldosuky@gmail.com

How to cite this paper: Rozik, A.S., Tolba, A.S. and El-Dosuky, M.A. (2016) Design and Implementation of the Sense Egypt Platform for Real-Time Analysis of IoT Data Streams. *Advances in Internet of Things*, **6**, 65-91.
http://dx.doi.org/10.4236/ait.2016.64005

Received: July 26, 2016
Accepted: September 16, 2016
Published: September 20, 2016

Abstract

Nowadays, we experience an abundance of Internet of Things middleware solutions that make the sensors and the actuators are able to connect to the Internet. These solutions, referred to as platforms to gain a widespread adoption, have to meet the expectations of different players in the IoT ecosystem, including devices [1]. Low cost devices are easily able to connect wirelessly to the Internet, from handhelds to coffee machines, also known as Internet of Things (IoT). This research describes the methodology and the development process of creating an IoT platform. This paper also presents the architecture and implementation for the IoT platform. The goal of this research is to develop an analytics engine which can gather sensor data from different devices and provide the ability to gain meaningful information from IoT data and act on it using machine learning algorithms. The proposed system is introducing the use of a messaging system to improve the overall system performance as well as provide easy scalability.

Keywords

Internet of Things, IoT Platforms, IoT Big Data Analytics, MQTT (Message Queue Telemetry Transport)

1. Introduction

With the enormous improvement in technology nowadays, there are billions of devices that are producing data continuously. Examples of such devices are temperature sensors, motion detectors, humidity sensors or even the luminosity sensor in a smart phone. Due to the vast amount of sensors that exist, the volume of data that get produced every second, will make it difficult to organize in a good and easy way. There are

many attempts to create platforms that allow users to register their sensors, actuators and visualize the data produced from theses sensors/actuators, such as Xively [2] and Thing speak [3]; each of them focuses on different features. This is the main reason behind the creation of the IoT Platform in this paper, in an effort to easily view, handle and interact with IoT data streams. In the platform system users can enroll their devices and transfer data streams to the IOT platform for data visualization and further processing. In addition, the system has the capability to analyze sensors data and according to analysis results, the system can send commands to other actuators registered in the system. In addition the analytics engine is able to also predict the data generated by sensors. The IoT Platform also supports the creation of triggers attached to streams. A trigger [4] is a mechanism that notifies the user by sound alerts, SMS and Email or send commands to other actuators when specific criteria are met or specific thresholds are reached. The Internet of Things IoT [5] is a concept that has become more popular lately; the main principle of IoT is to connect any device to the internet. The concept of internet of things (IoT) leads to big and great ideas, such as the smart city [6]. Suppose that you have just landed in this city airport then you head out to the airport to get a ride to the hotel. A sensor attached to the back of your ear asks if you need a ride for one person or more and you confirm your needs. At the curb [7], a vehicle stops in front of you and a green light on the door indicates it is yours. When you get in the car, there is no driver. In fact, there is no driver's seat or steering wheel. Instead, four seats face each other around a small central worktable. The car knows where you are going because of your hotel reservation information in your phone. As you approach the hotel, your room location appears on your phone. You are checked in automatically when you walk in the hotel entrance. Your phone unlocks your room. This is an example of how the Internet of Things will change our lives to the better. Innovative technologies working together seamlessly over the internet will transform how we interact with almost everything. There is exponential growth in the number of unique, sensor-rich and cyber-enabled devices processing data and communicating with other devices and computers around the world [7].

The IoT is estimated to consist of almost 50 billion devices by 2020 [8]. A smart city is a city where sensors are placed all around it and they would monitor, for example, air pollution, amount of traffic or how full a garbage can is. The information could then be used to make smart decisions, for example in the case of traffic monitors it could redirect traffic to lower the risk of traffic jams or with the garbage sensor one could make garbage collection more efficient by only collecting where it is needed. All of this devices will be too much and headache to be monitored by humans, so we need to build systems to handle data and then either provide a good overview of the data, suggest what to do (so a human can easily take the needed decisions) or automatically do it by the analytics engine that should send alerts and also send commands to the actuators based on the analysis results. Such a system would also need to have meta data regarding the devices to be able to decide how useful the data given from the device is. Ultimately what is envisioned here would be a platform that could handle 50 billion devices, gather

data by polling or pushing, have a good visual overview of the data but also be able to create virtual streams, have triggers on these streams and be able to analyze the data stream by making forecasts [9]. All these things were to be done in a user friendly way where the users could seamlessly add their own sensors and actuators to the system. The remainder of the paper is organized as follows. Section 2 describes the related work. Section 3 describes the proposed architecture for IoT platform. Section 4 demonstrates the proposed implementation for IoT platform. Section 5 describes the results & discussions. Section 6 concludes the paper. Section 7 describes the future work.

2. Related Work

In this section, we present two of the most used IoT platforms as shown in Table 1 that provide for smart objects or things (devices) the internet connectivity and the ability to analyze the data generated from these devices [1]:

Reference architectures for integrating sensors with cloud services and IoT platforms have been discussed in the literature [10]-[12]. These works propose the general architecture that can be used to connect sensors to IoT platforms and the potential issues. In this paper and in our work, we propose a framework that can be used to send the data generated from sensors to the IoT platform using MQTT protocol which is one of the best protocols most fitted to be used in sensors and low power devices as well as showing how to analyze the data and extract meaningful information within a generic framework. We also discuss how to transfer data and process it in a scalable way, topics that are not fully addressed in the previous papers. In addition we are showing the full details to build an IoT platform from scratch using open source technologies and we explored the machine algorithms that can be used to build the real-time analytics layer more efficiently than the previous techniques used in the previous papers that depend on thresholds checking.

Table 1. IoT platforms overview.

Platform	Protocol	Capabilities	Architecture	Open Source
Thing Speak [3]	HTTP	• Visualization • Data Analytics • Store Data • Integrate with Matlab	Centralized/Cloud Based	Yes
Xively [2]	HTTP	• Visualization • Data Analytics • Integrate with Sales Force	Cloud Based	Yes
Sense Egypt (proposed in this paper)	MQTT	• Visualization • Data Analytics • Send SMS/Email Alerts • Send Commands to Actuators based on Analytics results • Store Raw and analyzed Sensors Data	Cloud Based	Yes

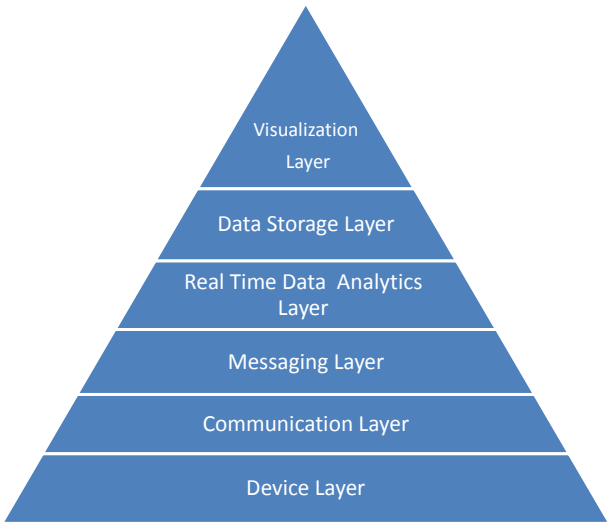

Figure 1. Architectural overview of Sense Egypt.

3. Proposed Platform Architecture

We adopt the architecture of a typical IoT system proposed in [10] as shown in **Figure 1** consists of the following layers:

1) Devices layer
2) Communication layer
3) Messaging layer
4) Real-time Data analytics layer
5) Data Storage layer
6) Visualization layer

Each layer in Sense Layer proposed architecture as shown in **Figure 1**, is briefly described as follows:

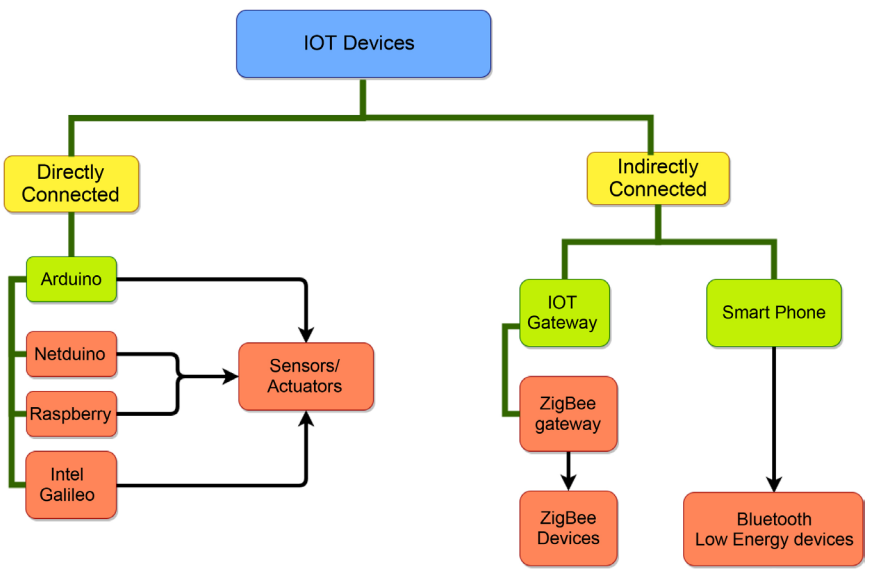

Figure 2. IoT devices types.

3.1. Devices Layer

We consider a device as a set of sensors and actuators. The below diagram shows the IoT devices and their connection to internet.

The diagram shown in **Figure 2** shows the different classes of devices [10] as follows:

1) Devices that don't have operating system like Netduino and Arduino.

2) Devices that may run operating system like linux or another suitable operating system. These devices may be used as a gateway for sensors and small devices, e.g. if a wearable sensor connects to a smart phone via Bluetooth or Raspberry Pi, which then enable sensors to connect to the internet.

3.2. Communication and Connectivity Layer

This layer provides the connectivity of the devices and IoT gateway to the rest of IoT platform pipeline. The gateway is the interface between sensors and the rest of the IoT pipeline. The role of IoT gateway is to abstract and encapsulate the sensors platform, aggregate data from sensors and then sending sensors data to the rest of IoT pipeline.

There are different communication models between IoT devices, IoT gateway and the Internet:

1) Direct WIFI or Ethernet connectivity via UDP or TCP/IP.

2) Connectivity through IoT Gateway.

There are different protocols for communication between IoT devices, IoT gateway and the internet. The most well-known three potential protocols are:

- HTTP/HTTPS (and RESTFUL approaches on those) [13]
- Universal Plug and Play (UPnP) [14]
- Constrained application protocol (COAP) [15]
- MQTT (MQTT official website) [16]
- Extensible Messaging and Presence Protocol (XMPP) [17]

1) Hypertext Transfer Protocol (HTTP):

HTTP has become much more than navigation between pages on the Internet. Today, it is also used in Internet of Things, among other things. So much is done on the Internet today, using the HTTP protocol, because it is easily accessible and easy to relate

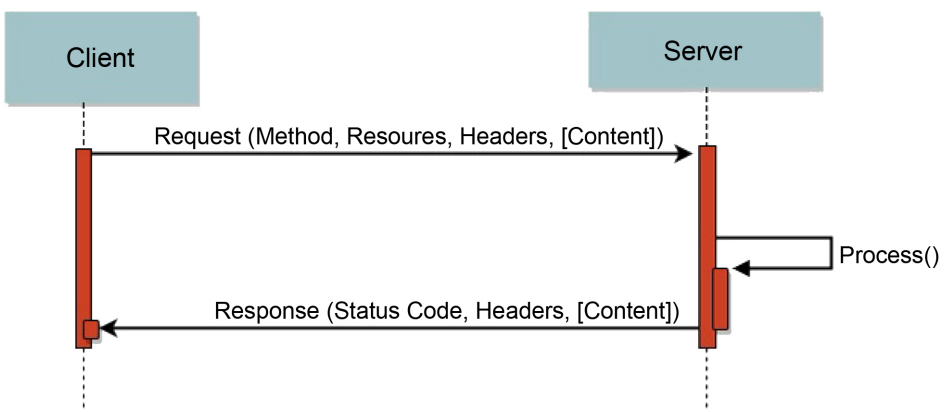

Figure 3. HTTP request/response pattern.

to. HTTP is a stateless request/response protocol where clients request information from a server and the server responds to these requests accordingly as shown in **Figure 3**. A request is basically consists of a method, a resource, some headers, and some optional content. A response is a three-digit status code, some headers and some optional content. The following diagram shows the HTTP request/response pattern [13]:

2) Universal Plug and Play Protocol (UPnP)

UPnP is a protocol or an architecture that uses multiple protocols, helps devices in ad hoc IP networks to discover each other, detects services hosted by each device and reports events. Ad hoc networks are networks with no predefined topology or configuration. Devices can find themselves and adapt themselves to the surrounding environment. UPnP is used by almost all network-enabled consumer electronics products used in your home or office, and as such, it is a vital part of Digital Living Network Alliance (DLNA). UPnP is largely based on an HTTP application where both clients and servers are participants. This HTTP is, however, extended so that it can be used over TCP as well as UDP, where both use unicast addressing (HTTPU) and multicast addressing (HTTPMU) [14].

3) Constrained Application Protocol (CoAP):

CoAP is a very light weight protocol based on HTTP but the main difference between CoAP and HTTPU is that CoAP replaces the text headers used in HTTPU with more compact binary headers, and furthermore, it reduces the number of options available in the header. This makes it much easier to encode and parse CoAP messages. CoAP also reduces the set of methods that can be used; it allows you to have four methods: GET, POST, PUT, and DELETE. Also, in CoAP, method calls can be made using confirmable and non confirmable message services. When you receive a confirmable message, the receiver always returns an acknowledgement. The sender can, in turn, resend messages if an acknowledgement is not returned within the given time period. The response code has also been reduced to make implementation simpler.

4) Message Queue Telemetry Transport (MQTT):

The MQTT protocol is based on the publish/subscribe pattern, as opposed to the request/response in the previous protocols. The publish/subscribe pattern has three types of actors:

- Publisher (MQTT Client): The role of the publisher is to connect to the message broker and publish the content.
- Subscriber (MQTT client): They connect to the same message broker and subscribe to content that they are interested in.
- Message broker: This makes sure that the published content is related to interested subscribers.

Content is identified by topic. When publishing content, the publisher can choose whether the content should be retained by the server or not. If retained, each subscriber will receive the latest published value directly when subscribing. Furthermore, topics are ordered into a tree structure of topics, much like a file system.

5) Extensible Messaging and Presence Protocol (XMPP)

The XMPP [15] open protocol, standardized by Internet Engineering Task Force (IETF) as are HTTP and CoAP. It uses message brokers to bypass firewall barriers. But apart from the publish/subscribe pattern, it also supports other communication patterns, such as point-to-point request/response and asynchronous messaging, that allow you to have a richer communication experience. XMPP was originally designed for use in instant messaging applications (or chat).

3.3. Messaging Layer

Data generated by the many sensors and devices of an IoT system typically needs to be delivered to the storage and analytics systems using (HTTP - UPnP - COAP - MQTT - XMPP) protocols as discussed in the previous section.

It is an important layer of the architecture because that it aggregates and brokers communications. It is a very important layer for the following reasons:

1) It supports MQ Telemetry Transport broker and HTTP Server in order to connect IOT devices to the Internet.

2) It's ability to mediate and route communications between different devices in system that may be connected via IoT gateway.

3.4. Real-Time Data Analytics Layer

Big Data generated by IoT devices is categorized into volume, velocity, and variety of the data [18]. Big Data analytics systems like Hadoop handles only the Volume and Varity but in real time big data analytics systems like Apache Storm and Apache Spark need to handle the velocity of the data in addition to volume and Varity as well.

There are more than one step for real-time analytics systems to be able to handle the data velocity, volume and Varity as follows:

1) Real-time data analytics system should collect the data produced by IoT devices coming in at a rate of thousands and millions of event/second [18].

2) Real-time analytics system should support parallel processing for collected data.

3) The real time system should be a low latency – and fault tolerant distributed system [18].

Objectives of Real-time Data analytics

1) Process data produced by IoT devices in real time or near real-time.

2) Extract meaningful information from data produced by IoT devices by performing event correlation using CEP (Complex Event Processing).

3) Provide predictive analytics for data produced by IoT devices.

4) Take actions based on results of analysis like sending SMS and Email alerts or sending commands to actuators registered in system.

3.5. Data Storage Layer

The data produced by IoT devices needed to be stored at each processing phase like the raw data produced IoT devices, pre-processed data, and analytics results. Storing data makes it possible to perform additional analytics later using the tool of your choice.

3.6. Visualization Layer

Visualization is critical for IoT application as this allows interaction of the user with the environment. This layer presents the raw data produced by sensors and the analysis to the end users of the platform.

4. Implementation of Sense Egypt Platform

Figure 4 shows the components of the real-time data analytics system that based on MQTT protocol. The data produced by IoT devices can be collected via MQTT broker and then can be processed in real time using Apache Storm, and then analytics results can be stored in a database. In between, the MQTT Broker and Messaging system (Apache Kafka) are used for storing/buffering the messages.

The implementation and components of the IoT platform is depending on the communication protocol between the IoT devices, IoT gateway and the internet so if HTTP is used as a communication protocol between IoT devices and the internet then the IoT devices will act as HTTP clients and the IoT analytics platform will act as HTTP server. So IoT devices (HTTP client) will emit their data to IoT analytics platform (HTTP Server). In Sense Egypt IoT platform (Proposed in this paper) we selected the MQTT as a communication protocol because as described in the communication and connectivity layer in the proposed architecture section, MQTT compares with other protocols like (HTTP, CoAP) is designed mainly for devices and is lightweight on the wire, this enables low cost device communication. MQTT is able to keep the bandwidth at an absolute minimum and it can deal with unreliable networks without the need for complex error handling and a huge effort in implementation. It was designed for keeping an steady line to your devices at a minimal cost to support real push notifications and real time communication. So if we need to connect IoT devices to Sense Egypt IoT platform for real time analysis of their data then the IoT devices will act as MQTT clients that

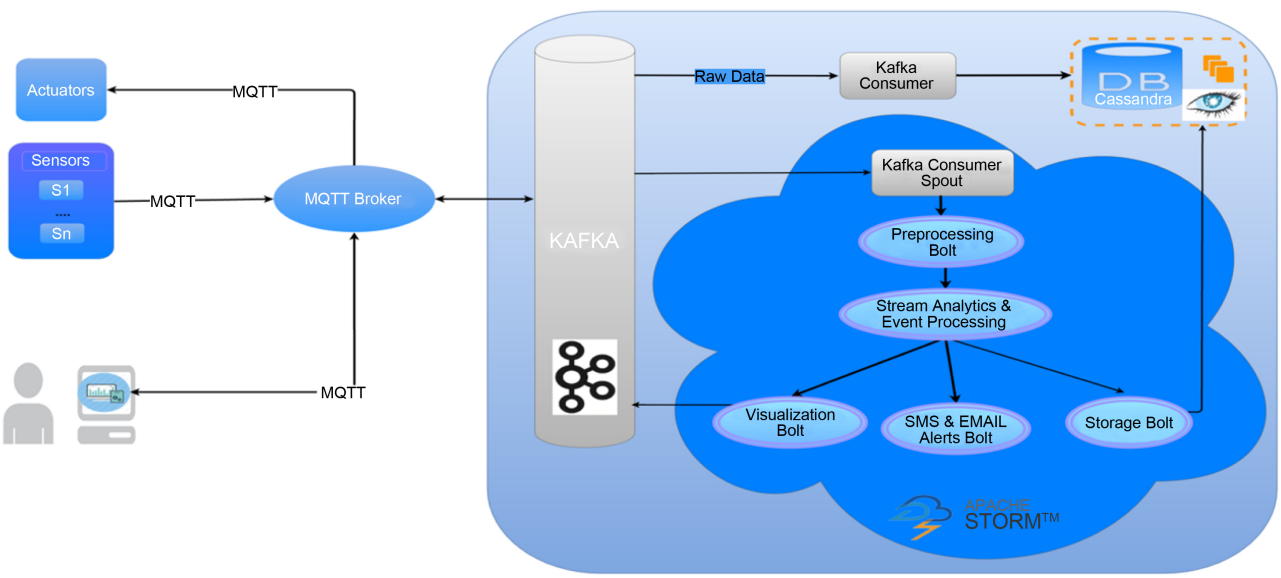

Figure 4. Structure and components of the Sense Egypt IoT platform.

publish their data periodically to the MQTT broker which will forward and send the data received from IoT devices to the rest of IoT pipeline for real time analysis and processing. After that MQTT Broker send the data to the Apache Storm framework that is a real time analytics engine and it's role is to analyze the data generated from IoT devices in real time and extract the meaningful information which will help in taking decisions. After the real time analysis is completed then we visualize the results and take actions accordingly such as send SMS and Email alerts as a notifications to the owners of IoT devices or sending commands to actuators registered in Sense Egypt IoT platform. The raw data gathered from IoT devices and the analyzed data are stored in apache Cassandra DataBase.

The proposed structure of Sense Egyp IoT platform is shown in **Figure 4**, and it's components are described as follows:

4.1. IoT Devices

Every IoT system is consisting of a set of devices and we each device is a set of sensors and actuators. IoT devices such as (Netduino, Arduino, Intel Galileo and Raspberry Pi) will act as MQTT client. The role of any of the mentioned devices is to read inputs from sensors such as (Temperature sensor, Light sensor, Motion detection sensor) and turn it into an output (actuators such as turning on a motor, turning on an LED).

The process of capturing of IoT Devices Data as shown in Figure 5:

1) IoT device which is a set of sensors and actuators should be registered on Egypt Sense platform portal. IoT device is acting as MQTT client that connects to MQTT broker. MQTT client is responsible for collecting information from a telemetry devices and publishing the readings to the MQTT broker. It can also subscribe to topics, receive messages, and use this information to control the telemetry devices so it can be a publisher and a subscriber to the MQTT broker at the same time. MQTT clients implement the published MQTT v3 protocol [16].

2) Sense Egypt platform is generating a unique topic (MQTT Id) for each sensor and actuator registered in the system.

3) The IoT device sensors should publish events (readings) to MQTT broker using the generated topic from the previous step.

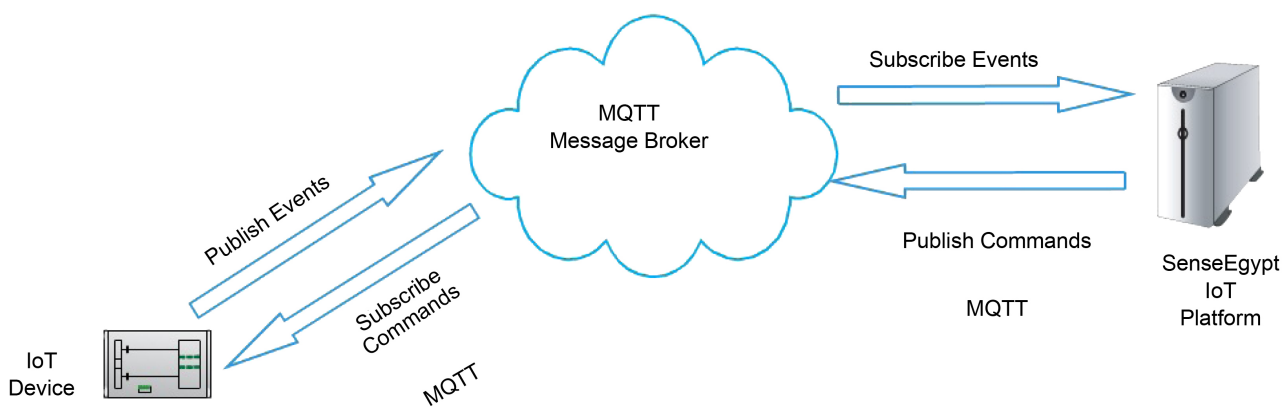

Figure 5. Shows the capturing process of IoT devices data in Sense Egypt IoT platform in real-time using MQTT protocol.

4) The IoT device actuators should subscribe to MQTT broker to receive commands using the topic generated from step no 2.

5) MQTT broker will emit the received sensors data to the IoT platform for advanced analytics as will shown below in the next sections.

MQTT client libraries are available in many different programming languages such as (Java, .Net .PHP, C#, Java Script, Node.Js, C++, C, Arduino) [19]. The owner of IoT devices should first register the sensors and actuators on the Sense Egypt platform to get specific topics to publish sensor readings on specific topic or subscribe to specific topic on the MQTT broker.

Structure of MQTT client application:

1) Create a client object

2) Set the options to connect to an MQTT server

3) Set up callback functions

4) Connect the client to an MQTT server

5) Subscribe to any topics the client needs to receive

6) Repeat until finished:

i) Publish any messages the client needs to

ii) Handle any incoming messages

7) Disconnect the client

8) Free any memory being used by the client

4.2. MQTT Broker (Hive MQ)

MQTT broker provides the ability to connect your devices over the Internet. It provides the capability to deliver messages in real time and it guarantees messages delivery. Connect thousands of devices to your platform, what makes MQTT truly excels is sending instant updates and broadcast push notifications. There are different implementations for message broker for example (Hive MQ, mosquito). Hive MQ [20] is selected in our proposed implementation for the following features:

1) High performance MQTT broker

2) Open Source Plugin System

3) Native Web sockets Support

4) Cluster functionality

5) Embeddable.

4.3. Messaging System (Apache Kafka)

MQTT Broker doesn't provide any buffering mechanism and is not scalable. When a large amount of data is coming in from multiple different sources, then both of these features are necessary. Systems like Apache Kafka should be used as an intermediate messaging system. Using intermediate messaging system between the MQTT broker and the rest of IoT pipeline system can help to improve the overall system performance as well as provide easy scalability. Apache Kafka is a publish/subscribe open source messaging system. A message broker is a programming module which translates mes-

sages from sender messaging protocol to receiver messaging protocol. Kafka is a publish/subscribe messaging system which can handle huge amounts of reads and writes per second from thousands of clients [21]. The major components of any messaging system are the producer of the message, the consumer of the message, and the message broker. The message broker uses message queues internally for asynchronous inter process communication between the message producers and consumers. Any messaging system supports both point to point in addition to the publish/subscribe communications model. In point to point communication model, the messages sent to the queue by the producer and multiple consumers may be registered with this queue, but the sent message to the queue will be consumed by one consumer only. In the publish/subscribe communication model, there are multiple producers that can produce messages for a specific topic, and multiple consumers can be subscribed for this topic. For each subscriber, it will receive the message sent to this topic. In the messaging system, the message broker is the core component which connects message producers and consumers.

4.4. Real Time Stream Analytics Engine (Apache Storm)

The real-time analytics engine is the brain of the IoT platform as the raw data received from IoT devices through the MQTT broker in real time need further processing. MQTT is already supported by Apache Kafka which makes integration effortless. The data received from an MQTT broker will be sent by Apache Kafka to different consumers. For example, one Kafka consumer could be used to send data to Apache Storm for data analysis and the other Kafka consumer could be used to send raw data to a database. The data received from Apache Kafka need further processing, such as adding a time stamp (if not already existed), expecting missing readings, filtering, analysis, predictions etc. We used Apache Storm [22] to achieve these goals. Storm has many features as it is a distributed, fault tolerant, reliable system for data streams processing. Storm topologies consist of two different components (Spouts and Bolts) and each of them is responsible for a specific processing task. Spout component is responsible for emitting the input streams of a storm cluster to other components for processing. The spout sends the data to a Bolt component, which transforms data in some way. A bolt either store the data into database, or sends it to some other bolt.

Storm Cluster as shown in Figure 6 consists of the following components:

- Nimbus node:
  Executes uploaded computations
  Responsible for code distribution across the cluster
  Starting workers across the cluster
  Computations monitoring and workers reallocation as required
- Zookeeper nodes: coordinates between nimbus and supervisor nodes in the Storm cluster
- Supervisor nodes—starting and stopping workers nodes

In Figure 7 the diagram shows storm topology components that consist of spouts and bolts. Spouts are responsible for data stream injection into the topology [18]. Bolts

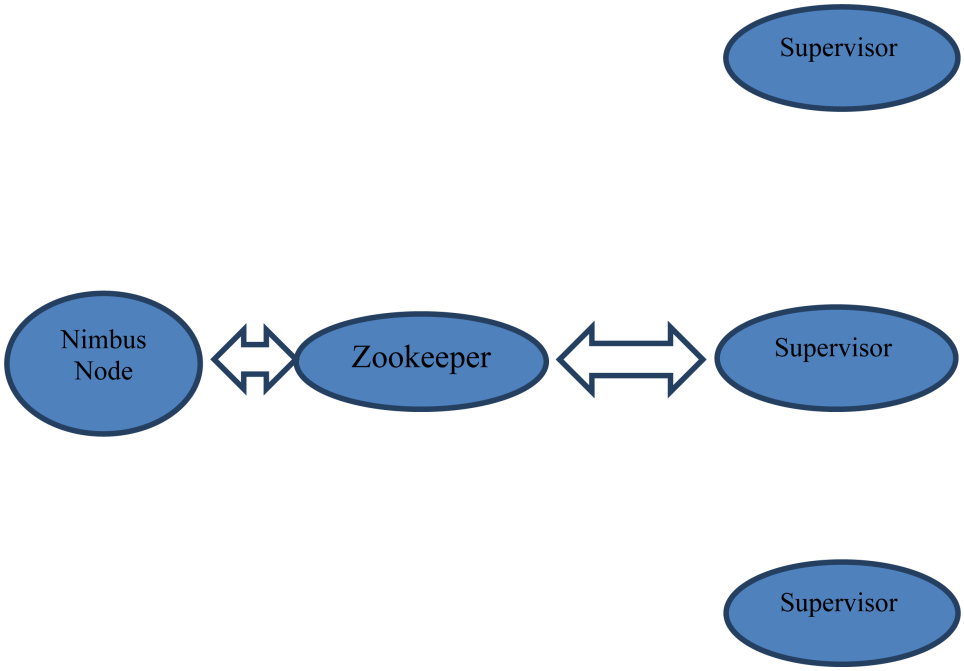

Figure 6. Storm cluster nodes.

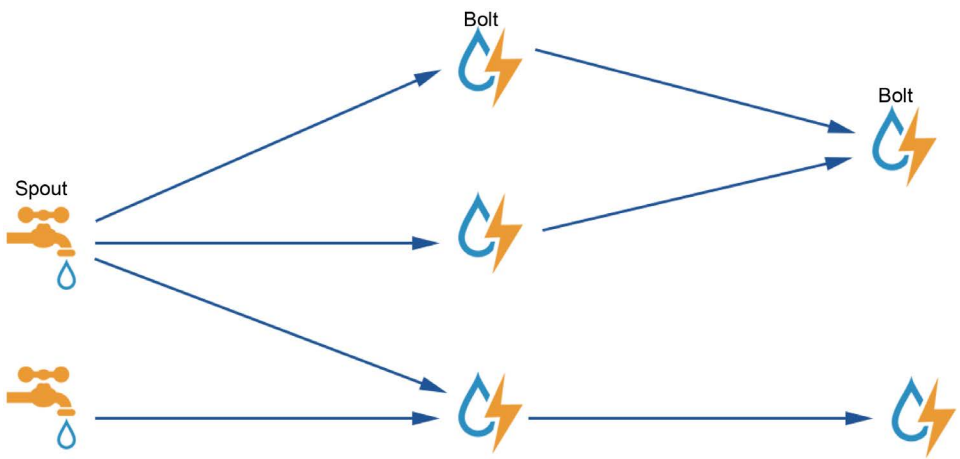

Figure 7. Storm topology components [21].

are responsible for doing some computations and processing over the data received from spouts or received from other bolts.

In Storm Cluster you run topologies. Stream is the core abstraction in Apache Storm framework. A stream consists of unlimited sequence of tuples. Apache Storm has three high level entities which actually run Topologies in Storm cluster [18]:

1) Worker Process

2) Executors

3) Task

A machine in a storm cluster may run one or more worker processes for one or more topologies. Each worker process runs executers for a specific topology and has it's own

Figure 8. Workflow to extract meaningful information from raw sensor data [23].

JVM. One or more executers may run within a single worker process.

Storm based Sensors Data Analytics in SenseEgypt

In the following section, we introduce a general workflow [23] to extract meaningful information from raw sensor data that has been defined by examining several different approaches for information abstraction in the domain of sensor data [24]. The approaches that have been examined follow the workflow as shown in **Figure 8**.

The phases of extracting meaningful information from the raw Data are as follow:

1) Pre-processing

2) Dimensionality reduction

3) Features Extraction

4) Classification

5) Visualization

The components of the data analytics layer as shown in Figure 8 are:

1) Kafka Consumer Spout:

Storm kafka spout [25] role is to fetch messages streams from apache kafka cluster and emit that stream to apache storm bolts for advanced processing. This spout will emit messages to pre-processing bolt.

2) Preprocessing Bolt:

IoT devices generate data in a raw form which is not necessarily suited directly for analytics engine implemented using Apache Storm framework. The generated Data may be missing, requiring an enrichment step, additional preparation or representations of values may need transformation (such as add time stamp for sensors readings) and we achieve this by applying any of the Pre-processing techniques.

Figure 9 is showing the most used pre-processing techniques and they are divided into the following:

i) Mathematical/Statistical Methods

a) Z-Normalization Algorithm

b) Min, Max Algorithms

c) Mean, Median Algorithms

d) Variance and Standard Deviation

e) Correlation and Integration techniques.

ii) Signal Processing Methods

a) Low Pass Filter

b) High Pass Filter

c) Band Pass Filter

The Pre-processing is consisting of the following phases:

i) Data Cleaning Phase:

In real time dynamic environment, a faulty or a missing sensor reading may occur due to bad communication channel or loss of service so we propose the following steps

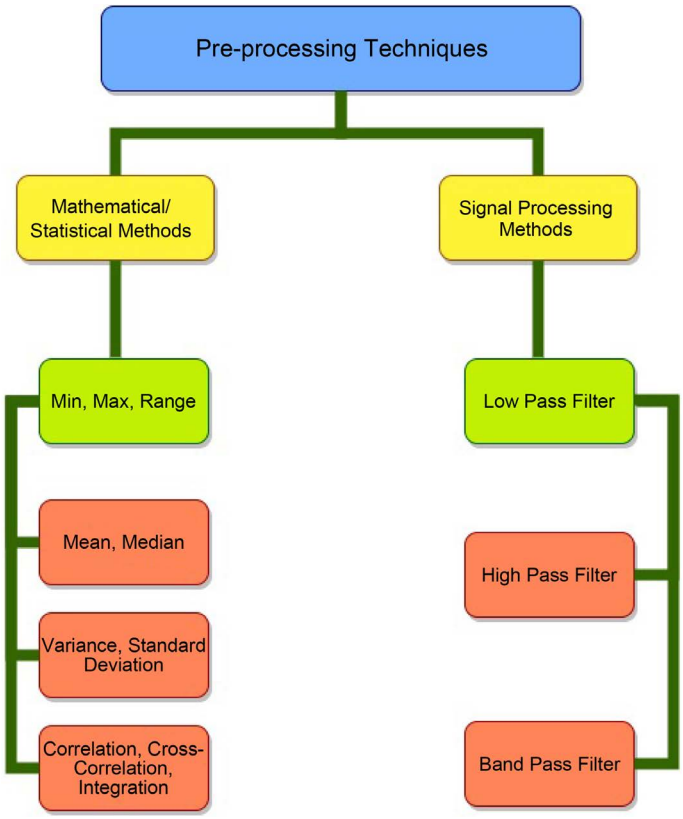

Figure 9. Pre-processing techniques.

for data cleaning phase:

a) Filtering out-of-range value:

To filter the sensor data values that are out of specific range we can use the Bandpass Filter. A Bandpass Filter has two cutoff frequencies, the lower and the upper frequencies and will only pass the signal in between.

b) Filling out missing values:

Missing values can be filled by the mean value of the sensor over some time window, by last recorded value and this can be done using mean/median algorithm

ii) Data Transformation Phase:

The second phase in pre-processing process is the data transformation and it involves transforming the data into the form which is optimum for machine learning process and this can be done using Z-Normalization.

Z-Normalization features:

i) Allows comparison of one time series data with another directly.

ii) Simplifying and enhancing the algorithm complexity.

The transformation formula is shown below:

$$X'i = \left(xi - \mu\right)/\sigma, \text{where } i \in N$$

As shown, the time series mean is subtracted from original values at first, and then the difference is divided by the standard deviation value second.

3) Analytics Bolt:

In the Pre-processing bolt we implemented the pre-processing technique that is the first phase of the proposed workflow to extract a meaningful information from the raw data. In the Analytics bolt we will implement the remaining phases of workflow.

i) Dimensionality Reduction Phase

After the raw data has passed to the pre-processing phase, we will pass the pre-processed data to the dimension reduction phase to reduce the data size. Several variants of aggregation techniques are used in order to reduce the data size without any loss of information. PAA and extended version of PAA called SAX are the most commonly used aggregation techniques in IoT for data reduction [26] as shown in **Figure 10**.

We will use Piecewise Aggregation Approximation (PAA) algorithm for the dimension reduction phase.

Piecewise Aggregation [27] divides the original data of length N into n equally sized windows by taking the mean of each window. This results in a reduction of data size from N to N/n data points. A shorter window length n results in a better reconstruction of the original data, however more data space is needed to store the data and eventually higher energy consumption by higher communication costs.

The pseudo code of the PAA Algorithm

```
Suppose    the following:
"D" is the list of data that containing the raw values that we will reduce it's size,
"L" is the desired output length,
"w" is the number of equally sized windows,
"output" is the result vector of PAA function,
"p" is the pointer    to the data list,
"s" is the segment.
"n" is the iterator over output vector

function    PAA(L, D)
        w:= length(D)/L
        p=0
        n=0
        output[]                                                    // initialize the output vector of length L
        while p < length(D)    do          // iterate over the
            s:= D(p,p+w)              //return segment of the original data of size "w"
output_n    = mean(s)          //calculate the mean of the segment values
            p = p + w                                   //increase the value of the pointer with value of "w"
            n++                                          // increase the value of the output vector iterator
        end while
        return output                  // return the output vector    after    iterating over the data
end function
```

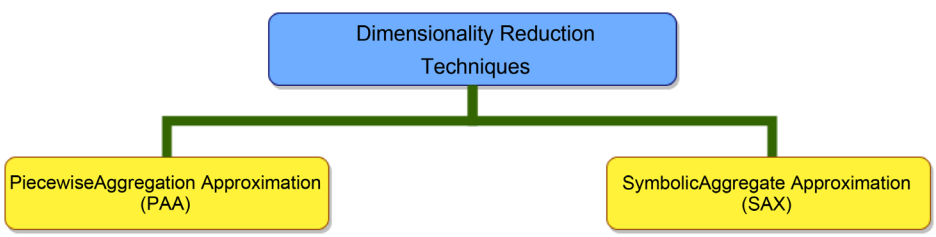

Figure 10. Dimensionality reduction techniques.

ii) Features Extraction Phase

Feature extraction is another technique used widely for data reduction where the number of features of data are large and mostly correlated to each other. Feature extraction enables to extract most relevant and uncorrelated features in order to perform optimum analysis [28].

There are many techniques developed for Feature Extraction as shown in **Figure 11**, but PCA is one of the most famous feature extraction technique which form new uncorrelated features based on the statistical properties in order to represent the same data with less dimensions.

The pseudo code of the PCA Algorithm

Input: $x_1, \ldots, x_n$ d length vector, k
Output: Transform matrix R
X ⇐ n × d data matrix with xi in each row;

$$x' = \frac{1}{n}\sum_{i=1}^{n} x_i$$

X ⇐ subtract x' from each row x_i in X;

$$COV \Leftarrow \frac{1}{n-1} X^T + X$$ Compute eigenvalue $e_1, \ldots, e_d$ of COV, and sort them;

Compute matrix V which satisfy $V^{-1} \times COV \times V = D$, D is the diagonal matrix of eigenvalue of COV;
R ⇐ the first k column of V

iii) Data Classifications Phase

After the raw data has passed to the dimensionality reduction and the features of the data produced by IoT devices have been extracted [23] to detect the outliers [29]. In analysis of the time-series the similarity can be computed by comparing the observed values and can be computed also using meta information such as time or type. After features are extracted from sensors data we need to classify these features and there are many techniques [30] developed for this purpose as show in the figure below.

There are many techniques developed for IoT data classification as shown in **Figure 12**, but Support vector machine (SVM) is one of the most widely used classification algorithm.

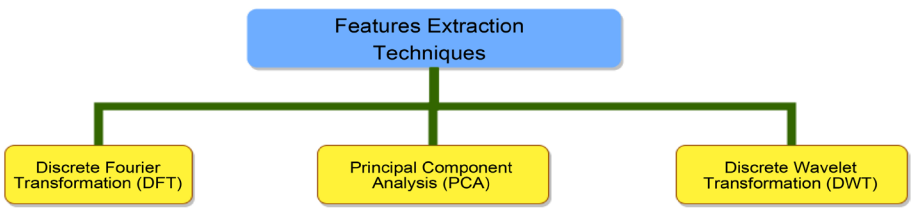

Figure 11. Features extraction techniques.

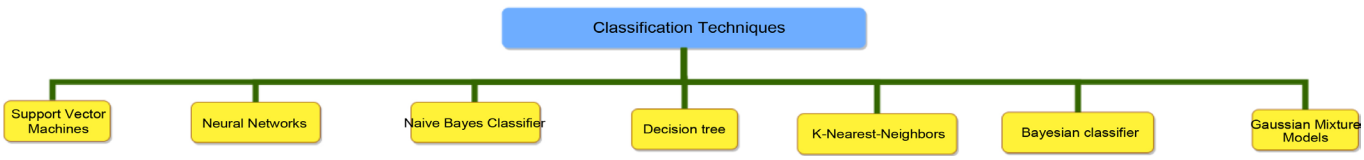

Figure 12. Data classifications techniques.

The two main advantages which gives SVM an edge on others are:

a) Its ability to generate nonlinear decision boundaries using kernel methods.

b) It gives a large margin boundary classifier.

We can conclude the following workflow for IoT Data Analytics Process the selected algorithms for each phase as shown in **Figure 13**:

All the above algorithms and techniques can be implemented using Apache Mahout Library [31].

iv) Storage Bolt:

Storage Bolt is used to interact with various databases. To store the generated raw data from IoT devices and the data generated from the preprocessing bolt, Apache Cassandra DB, Apache Couch DB or Mongo DB are good alternatives.

v) Alerts Bolt:

When any of the matching rules and thresholds are met then the appropriate action handlers in the alerts bolt can be executed such as sending SMS or Email to users. For example, if we have temperature sensor and we set rules and thresholds for this sensor such as minimum threshold is 10 and the maximum threshold is 60 and we selected to send SMS if any of these matching rules are met, and if the sensor reading is below the minimum threshold or above the maximum threshold then SMS should be sent to a certain mobile phone number.

vi) Visualization Bolt:

This bolt just send analytics results to Apache Kafka which sends it to the MQTT broker so that results are visualized to the system users using a dashboard and then act on it.

4.5. Storage Layer: Apache Cassandra

Apache Cassandra™ [32] is a scalable open source NoSQL database. World is moving to big data low cost, highly scalable, and reliable solutions that can store endless amounts of data. Well, we are handling real-time analytics, and everything we need should be accurate, fail-safe, and lightning fast. Therefore, Cassandra is the best choice because:

- It has the fastest writes amongst its peers such as HBase and so on.
- No single point of failure.
- Read and write requests can be handled without impacting each other's performance.
- Handles search queries comprising millions of transactions and lightning-fast speeds.
- Fail-safe and highly available with replication factors in place.

4.6. Visualization Layer Node.JS

For the visualization of sensor data, we have developed a simple dashboard to display

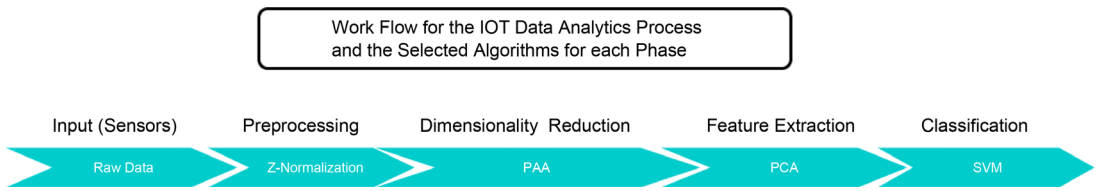

Figure 13. Data abstraction process workflow [24].

charts of raw and analyzed data received from sensors through a Message Broker (Hive MQ) and to display the analyzed data. The dashboard is a simple Node.JS [33] web application in addition using front end technologies such as jQuery, and D3 visualization libraries. The web app gets the data from the historical storage and subscribes to sensors real-time updates MQTT over Web sockets as shown in Figure 14.

Real-Time Analysis of Sensors Data:

After discussing all the components and the structure of Sense Egypt platform now we are concluding how the real-time analysis of sensors data is done.

Sense Egypt IoT platform handling the real time analysis of the sensors data as shown in Figure 15 as follow:

1) The IoT devices acting as MQTT clients to the MQTT broker so that the data generated from sensors will be published periodically to the MQTT broker.

2) The MQTT broker (Hive Mq) receive sensors data and send them to the messaging system (Apache Kafka) for buffering.

3) The data received from an MQTT broker will be sent by Apache Kafka to different consumers. For example, one Kafka consumer could be used to send data to Apache Storm for data analysis and the and the other Kafka consumer could be used to send raw data to a database.

4) Apache Kafka send the data received to Apache Storm through kafka consumer spout. The data received from Apache Kafka need further processing, such as adding a time stamp (if not already existed), expecting missing readings, filtering, analysis,

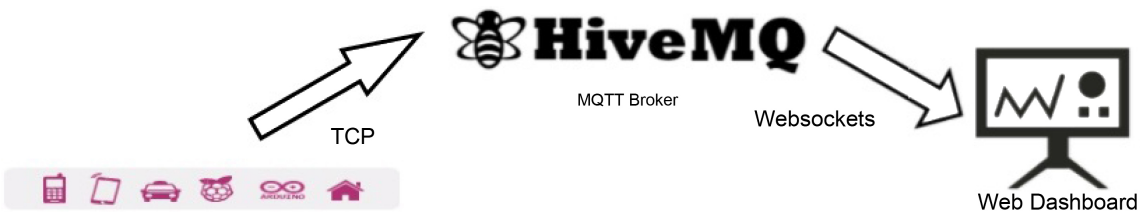

Figure 14. Shows the visualization layer components. It shows how the web dashboard gets the data from MQTT broker (Hive MQ) over HTML web sockets.

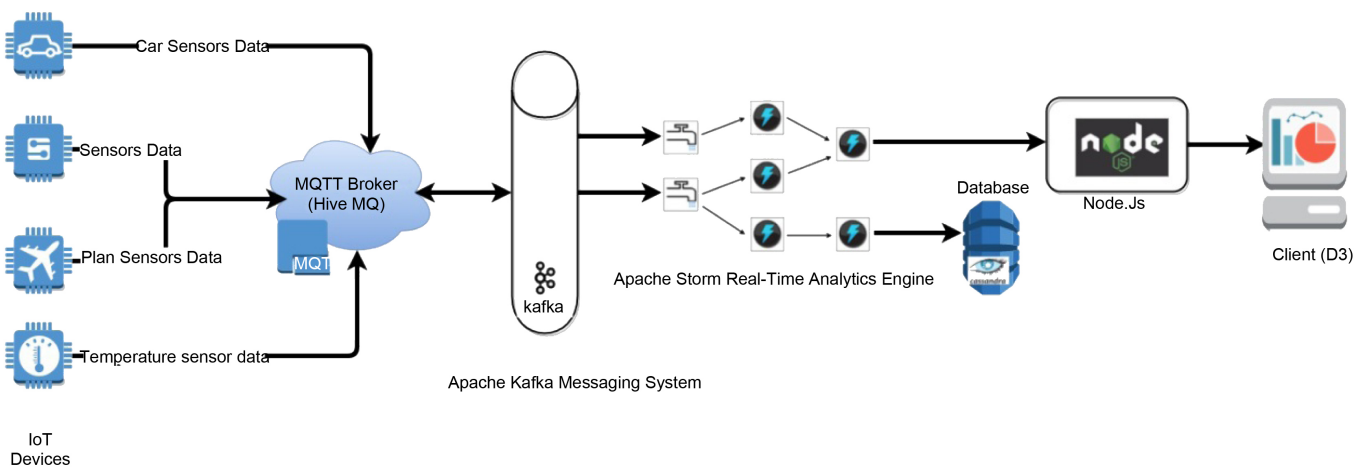

Figure 15. Shows how Sense Egypt IoT platform perform the real-time analytics of sensors data.

predictions etc. We used Apache Storm to achieve these goals.

5) Apache storm is responsible for the real-time analytics of sensors data and the proposed system is using machine learning algorithms for real time analytics of data as shown above. The analyzed data are stored in Apache Cassandra DB through Apache storm storage bolt. Actions also are taken based on analytics results so that SMS/Email alerts can be sent to users and also commands can also be sent to the subscribed actuators(IoT device) through MQTT broker to take an action.

6) The results of real-time analytics of sensors data are visualized and displayed in charts in a simple dashboard developed using D3 library in Node.Js framework.

5. Results & Discussions

5.1. Platform Portal

The main objective of this research is to build a platform for real time analysis of IoT data streams. The Web portal consist of the following pages:

5.1.1. Main Page

IoT Platform Main page from which the user can Sign In, Sign up and open channels page that enables the user to enroll sensors and actuators to the system.

5.1.2. Channels Page

The first step for users to be able to use the portal is to create an account from Sign up page on the platform to be able to add sensors and actuators to their channels. After the user has created his account in the system, he can now be logged on from the Sign on page and then he can add new sensors and actuators to his account. To add a new device to your channel then user should click on the new device button in the channel page then a new form will be displayed and these data should be entered (Device Type, Device Name, Device Description, Mobile No, Device Latitude and Longitude, Minimum and maximum thresholds, Select the triggered actuator if found, Check SMS Box if needed to send SMS message to user using the mobile number entered above if any of the threshold rules is true and Check Email Box if needed to send email messages to users using the email entered during account registration if any of the thresholds rules applies and then click Save button.

After the user Click on the save button in the devices registration page **Figure 16**, The devices table will be displayed and it contains all the devices registered to this user as shown below in **Figure 17**.

Device Registration Sequence Diagram:

The below sequence diagram in **Figure 18**, shows how the registration of devices is done.

5.1.3. Dashboard Page

Figure 19 shows the Dashboard page from which the user can select from "Device" drop down list the sensor that needed to be monitored. When sensor is selected then the data generated by this sensor is displayed in a customized chart in addition to the

analyzed data. In addition the location of each sensors is displayed in Google map as shown in the above figure.

Sensors Interaction with IOT platform sequence diagram as shown in Figure 20.

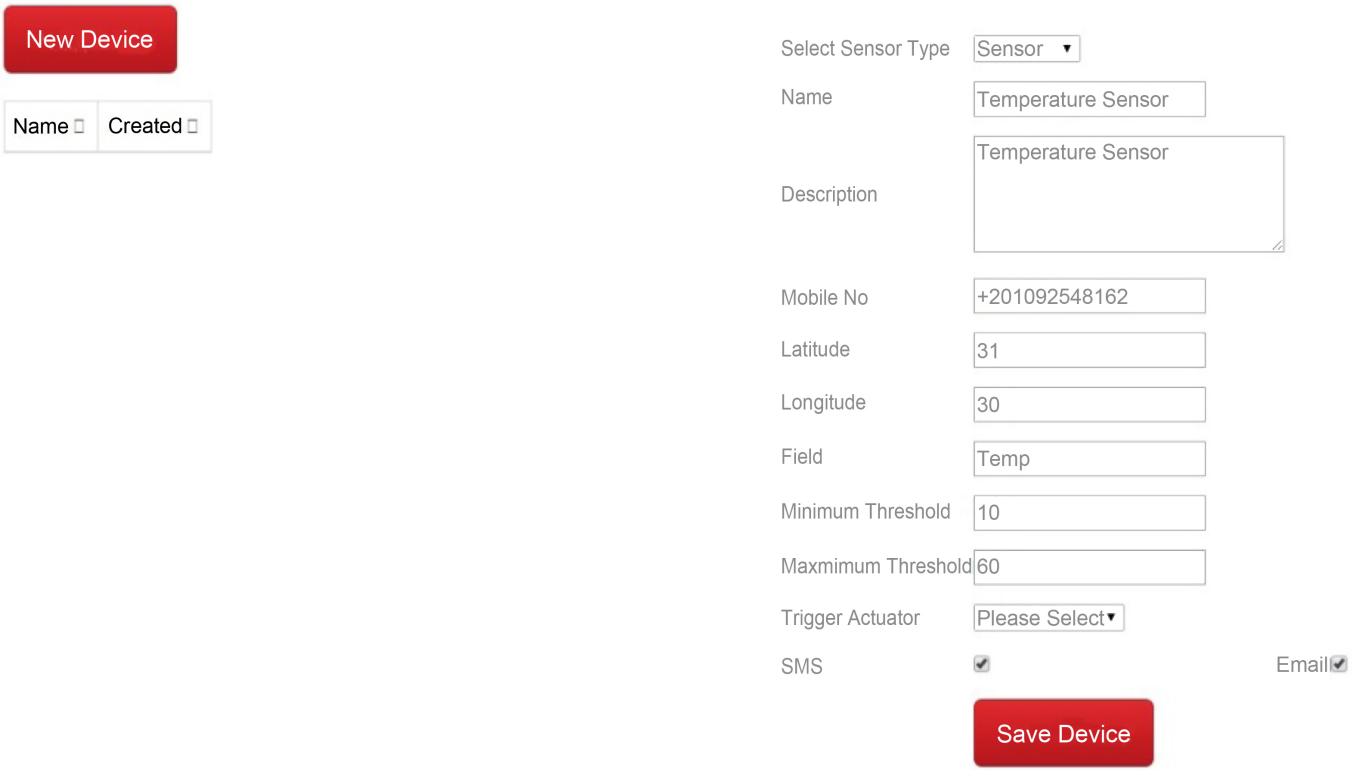

Figure 16. Shows channels page from which we can add a new device (sensor or actuator).

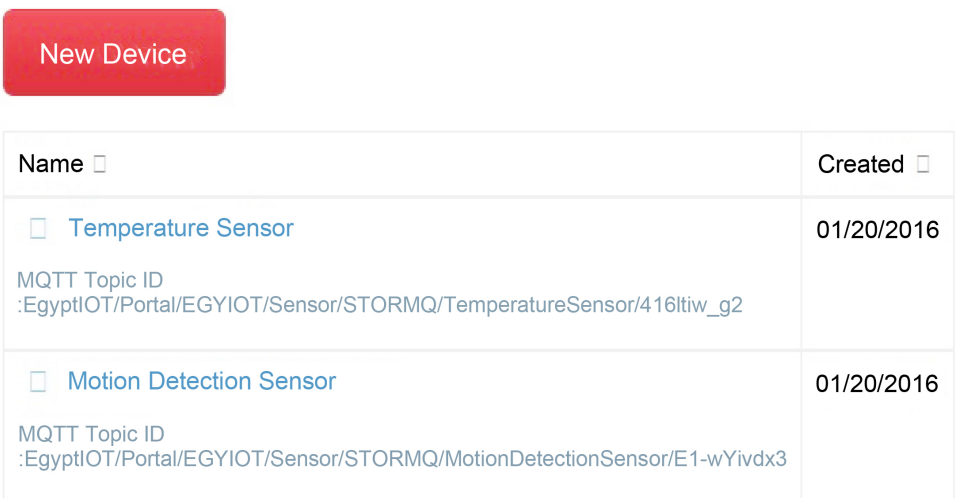

Figure 17. Shows the list of registered sensors and actuators in a table that consists of device name, unique MQTT topic generated by system for the registered device and the creation date. The MQTT topic generated by system should be used by the user to subscribe his devices (sensors and actuators) to the MQTT message broker so any device should send it's data to this MQTT topic.

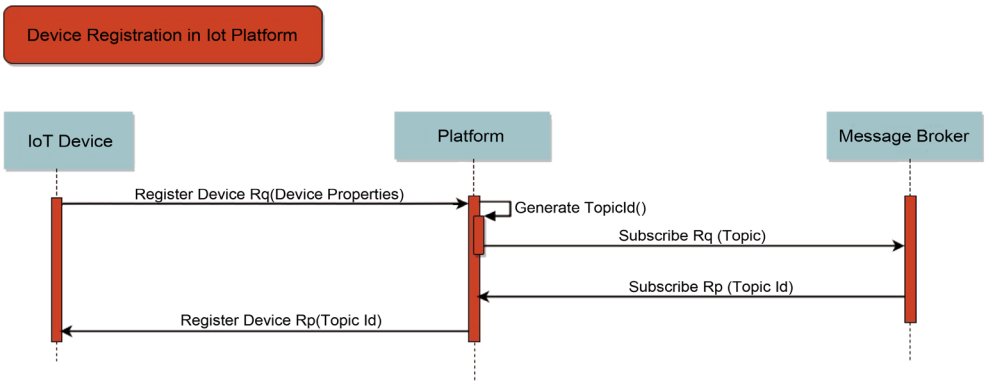

Figure 18. Shows device registration sequence diagram.

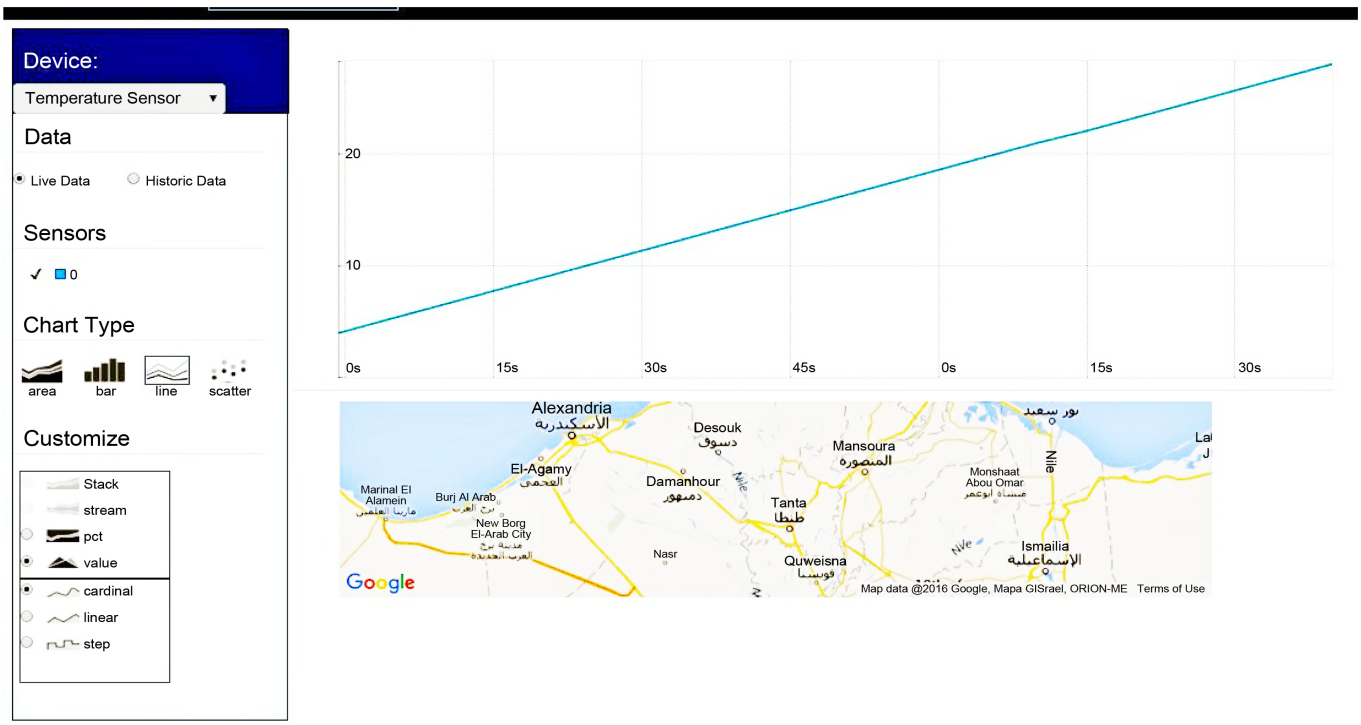

Figure 19. Dashboard and navigation pane.

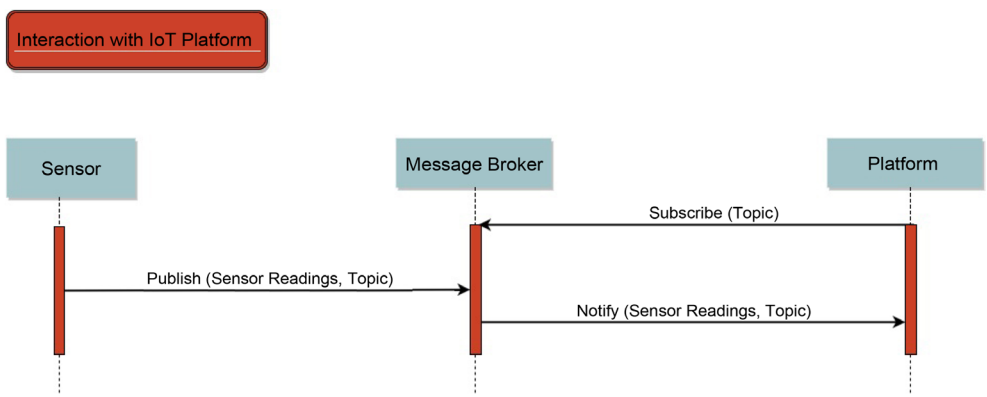

Figure 20. Sequence diagram for devices interaction with IOT platform.

5.2. MQTT Broker (Hive MQ) Performance Evaluation

The MQTT Broker (Hive MQ) is one of the main components in the system that enables sensors and actuators to connect to the rest of IoT pipeline, so we need to evaluate it's performance as follows. All Tests were executed on Amazon Web Services (AWS), a cloud infrastructure provider.

Hive MQ Server Instance Hardware specs as shown below in Table 2: The following EC2 Instance Type was used for the Hive MQ installation:

1) Latency Test:

This test shows the latency of Hive MQ for different Quality of Service levels for different amounts of MQTT clients and high throughput.

Latency is key for IoT systems at high scale where responsiveness and the real time experience are key acceptance factors of end users or downstream systems. The following benchmark shows how Hive MQ performs in an end-to-end scenario with real network round trip for latencies.

QoS 1 Results:

This benchmark tests the end-to-end latency of MQTT messages with QoS 1 guarantees. This means, Hive MQ uses disk persistence for every outgoing MQTT message due to the at least once semantics of QoS 1. No messages were lost in this test since the TCP connection was stable all the time and the QoS 1 guarantees were in place as shown in Figure 21.

Discussion

This test shows that the throughput and latency was stable all the time for the whole measurement time (45 minutes) of every individual test. With an increasing number of clients and messages per second the latency did not increase significantly. Average round trip time was always in the lower one-digit milliseconds. Even with linearly increasing throughput and number of subscriptions all measured latencies remain very low. Every single message was persisted to disk before delivering so the additional latency compared to QoS 0 messages are a result of the additional disk I/O overhead. This benchmark demonstrated that Hive MQ delivers very high QoS 1 message throughput (>15,000 messages per second) with a one-digit latency by average, while complying to the QoS 1 at-least-once guarantees as Table 3 shows.

2) Telemetry Test:

Table 2. Shows the hardware used in the performance evaluation process.

Name	Value
Instance Type	C4.2x large
RAM	15 GiB (~16 GB)
v CPU	8
Physical Processor	Intel Xeon E5-2666 v3
Clock Speed (GHz)	2.9

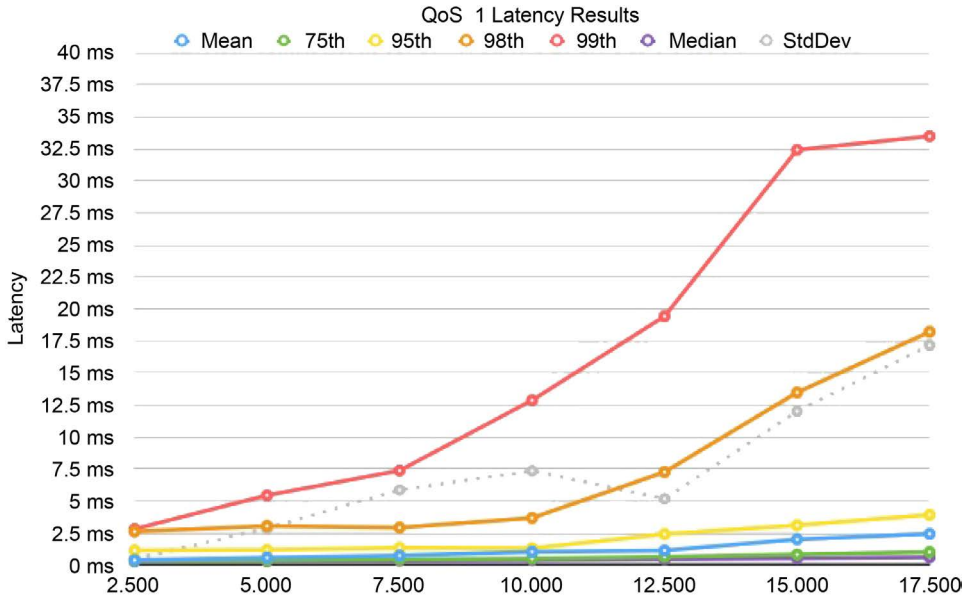

Figure 21. Number of clients/messages per second.

Table 3. QoS 1 latency results.

	2.500	5.000	7.500	10.000	12.500	15.000
Mean	0.415047152	0.604632525	0.767690498	1.062635311	1.163355852	2.032116374
75th	0.358394	0.389408	0.461854	0.525717875	0.675208875	0.857987
95th	1.17566645	1.2139728	1.374251375	1.3377911	2.4430323	3.134619025
98th	2.64368922	3.07982922	2.95271823	3.6996623	7.28728491	13.49425474
99th	2.8415169	5.4699688	7.403215115	12.88660823	19.44312396	32.44365795
Median	0.3032895	0.328127	0.3717255	0.40715475	0.481312	0.55862825
Std Dev	0.506549184	2.889852539	5.889325998	7.383906539	5.198095769	12.02608314

MQTT brokers are often deployed in environments where it's key to collect data from a huge amount of devices while only a few subscribers process the data published by the devices. A typical use case are telemetry scenarios where the MQTT broker needs to process a very high incoming MQTT message rate. The following benchmarks focus on the throughput of Hive MQ in such a scenario. In order to understand the runtime behavior of Hive MQ in a telemetry scenario, all relevant runtime statistics like CPU usage, and RAM and used bandwidth are measured. So this benchmark is focused on resource consumption of Hive MQ while delivering constant message throughput.

QoS 1 Results

This benchmark tests the resource consumption of Hive MQ with incoming QoS 1 messages. As discussed in the Benchmark Setup section, the subscribing clients subscribe with QoS 0. The following measurements were executed during the test executions: Average CPU utilization as shown in Figure 22, used total memory as shown in Figure 23,

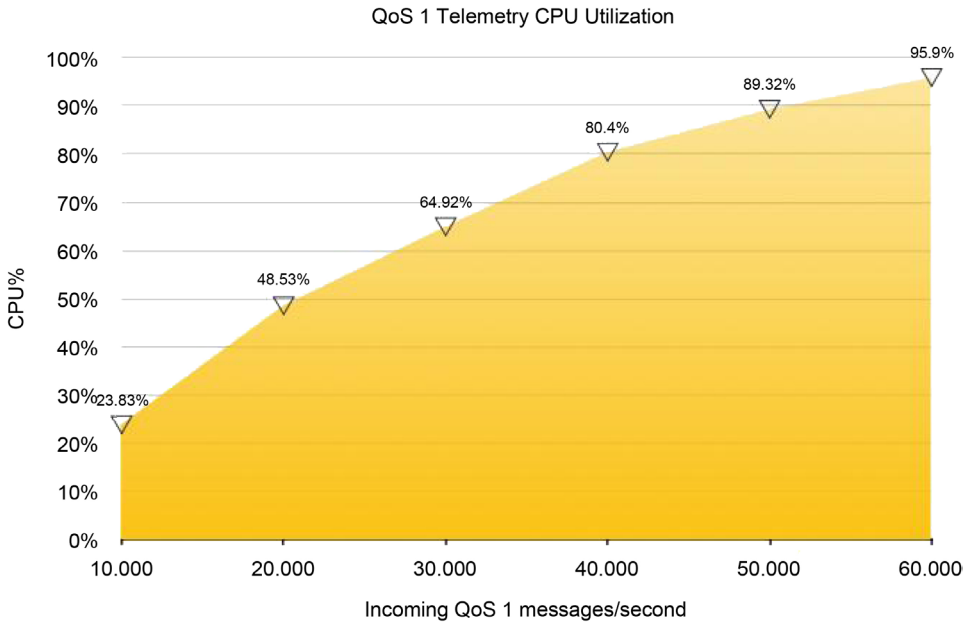

Figure 22. CPU utilization.

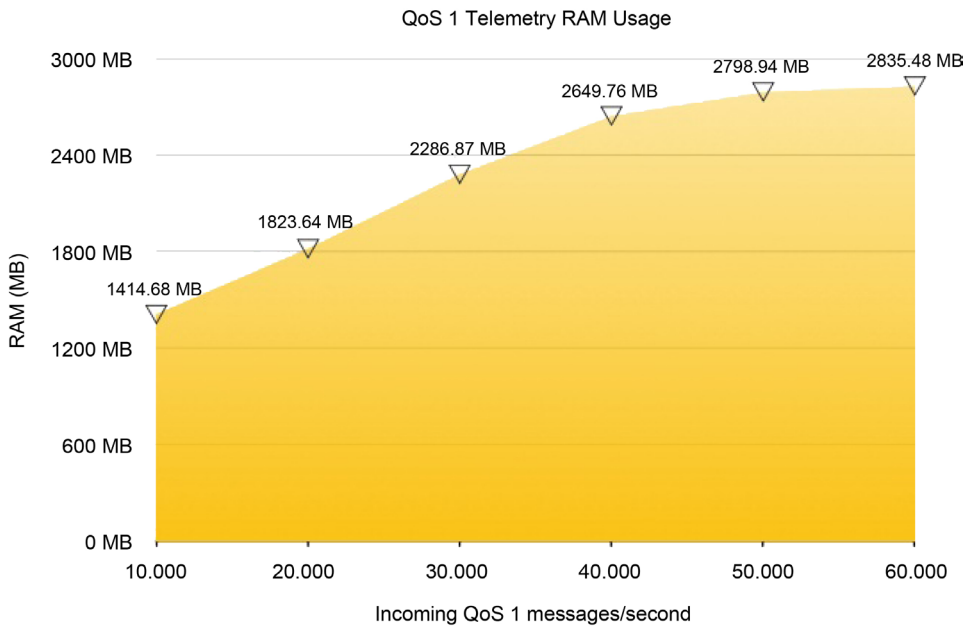

Figure 23. RAM usage.

incoming and outgoing traffic per minute as shown in Figure 24. No messages were lost in this test since the TCP connection was stable all the time.

Discussion

Increasing the total number of QoS 1 messages per second linearly result in linear bandwidth increase while CPU and RAM usage grow at a predictable level. A notable observation is, that while the bandwidth usage increases linearly with the number of messages/second, the CPU and RAM usage do not increase linearly. Hive MQ delivers

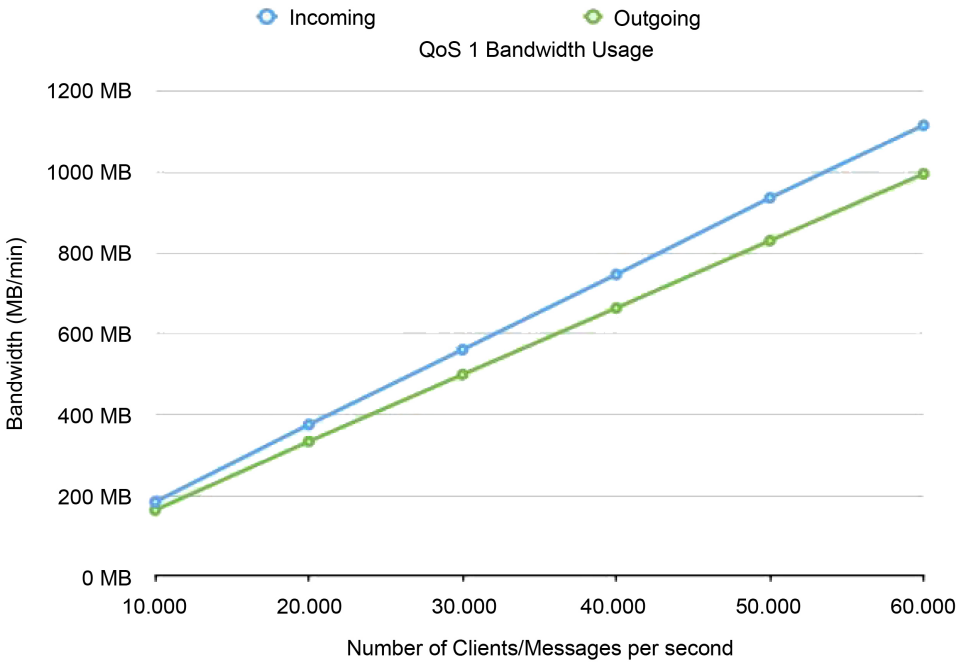

Figure 24. Bandwidth usage.

constant and predictable results until CPU limits of the EC2 instance are reached.

RAM is negligible in this test since 3 GB of RAM usage were never exceeded although the machine was configured to reserve up to 10 GB of RAM for Hive MQ. The limiting factor in this test is clearly CPU and even higher throughput can be expected for machines with more computing power. The multithreaded nature of Hive MQ allows to scale with the number of CPUs.

6. Conclusion

In this paper, we proposed a platform for real time IoT Data analytics using MQTT Protocol to support delivery of large volumes of data. An architecture is presented for IOT data analytics platform and also the implementation for each layer in the proposed architecture. Also we presented the open source technologies that can be used in messaging layer (Apache Kafka, Hive MQ), Analytics (Apache Storm) layer, Storage (Apache Cassandra) layer and Visualization layers (Node.JS Framework) layer. In addition to analytics layer, a workflow to extract meaningful information that is human and/or machine-understandable from raw data generated by sensors and the algorithms that should be applied in each of the work flow stage is also presented. Also a dashboard is implemented to visualize sensors data and commands to actuators registered in platform.

7. Future Work

Future research will focus on extending the platform with new analytics techniques to work with high performance computing and Big Data analytics tools such as Hadoop

[23]. In addition revamp the current system architecture to apply lambda architecture that consists of Batch, Serving and real time layer. Also we will use Symbolic Aggregate Approximation (SAX) Algorithm for Dimensionality Reduction layer as some studies indicate that the SAX algorithm performs better in preserving the features of data and also use the Hidden Markov Model Algorithm for features extraction.

References

[1] Mineraud, J., Mazhelis, O., Su, X. and Tarkoma, S. (2016) A Gap Analysis of Internet-of-Things Platforms. *Computer Communications*, **89-90**, 5-16. http://dx.doi.org/10.1016/j.comcom.2016.03.015

[2] Xively—Public Cloud for the Internet of Things. https://xively.com/

[3] Thing Speak—Internet of Things. https://www.thingspeak.com/

[4] Vandikas, K. and Tsiatsis, V. (2014) Performance Evaluation of an IoT Platform. 8*th International Conference on Next Generation Mobile Apps, Services and Technologies*, 10-12 September 2014, 141-146. http://dx.doi.org/10.1109/NGMAST.2014.66

[5] Ashton, K (2009) That "Internet of Things" Thing: In the Real World Things Matter More than Ideas. *RFID Journal*. http://www.rfidjournal.com/articles/view?4986

[6] Mahizhnan, A. (1999) Smart Cities: The Singapore Case. *Cities*, **16**, 13-18. http://lkyspp.nus.edu.sg/wp-content/uploads/2013/04/pa_Arun_Smart-Cities-The-Singapore-Case_99.pdf http://dx.doi.org/10.1016/S0264-2751(98)00050-X

[7] Mcpherson, D. (2016) Big Data and the Internet of Things. http://blog.edx.org/big-data-and-the-internet-of-things

[8] Evans, D. (2011) The Internet of Things: How the Next Evolution of the Internet Is Changing Everything. http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[9] Arias Fernández, J., *et al.* (2013) IoT-Framework. Product Report, Uppsala University. http://www.it.uu.se/edu/course/homepage/projektDV/ht13/ProductReport.FINAL.pdf

[10] Fremantle, P. (2015) A Reference Architecture for the Internet of Things. http://wso2.com/wso2_resources/wso2_whitepaper_a-reference-architecture-for-the-internet-of-things.pdf

[11] Dash, S.K., Sahoo, J.P., Mohapatra, S. and Pati, S.P. (2012) Sensor-Cloud: Assimilation of Wireless Sensor Network and the Cloud. In: Meghanathan, N., Chaki, N. and Nagamalai, D., Eds., *Advances in Computer Science and Information Technology. Networks and Communications*, Springer, Berlin, 455-464. http://dx.doi.org/10.1007/978-3-642-27299-8_48

[12] Alamri, A., Ansari, W.S., Hassan, M.M., Hossain, M.S., Alelaiwi, A. and Hossain, M.A. (2013) A Survey on Sensor-Cloud: Architecture, Applications, and Approaches. *International Journal of Distributed Sensor Networks*, **9**, Article ID: 917923. http://dx.doi.org/10.1155/2013/917923

[13] Hypertext Transfer Protocol—HTTP/1.1. https://tools.ietf.org/html/rfc2616

[14] Universal Plug and Play (UPnP). https://jan.newmarch.name/internetdevices/upnp/upnp.html

[15] Constrained Application Protocol (CoAP). http://tools.ietf.org/html/draft-ietf-core-coap-18

[16] OASIS Message Queuing Telemetry Transport (MQTT) TC.

https://www.oasis-open.org/committees/mqtt/

[17] Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. http://xmpp.org/rfcs/rfc3921.html

[18] Bhattacharya, D. and Mitra, M. (2013) Analytics on Big Fast Data Using Real Time Stream Data Processing Architecture. EMC Proven Professional Knowledge Sharing.

[19] MQTT Client Libraries. https://github.com/mqtt/mqtt.github.io/wiki/libraries

[20] Hive MQ Official Website. http://www.hivemq.com/mqtt/

[21] Apache Kafka Official Website. http://kafka.apache.org/

[22] Apache Storm Official Website. http://storm.apache.org/

[23] Ganz, F., Puschmann, D., Barnaghi, P. and Carrez, F. (2015) A Practical Evaluation of Information Processing and Abstraction Techniques for the Internet of Things. *IEEE Internet of Things Journal*, **2**, 340-354. http://dx.doi.org/10.1109/JIOT.2015.2411227

[24] Hall, D.L. and Llinas, J. (1997) An Introduction to Multisensor Data Fusion. *Proceedings of the IEEE*, **85**, 6-23. http://dx.doi.org/10.1109/5.554205

[25] Storm-Kafka Spout Code. https://github.com/apache/storm/tree/master/external/storm-kafka-client

[26] Li, X., Lu, R., Liang, X., Shen, X., Chen, J. and Lin, X. (2011) Smart Community: An Internet of Things Application. *IEEE Communications Magazine*, **49**, 68-75. http://dx.doi.org/10.1109/MCOM.2011.6069711

[27] Ganz, F., Barnaghi, P. and Carrez, F. (2013) Information Abstraction for Heterogeneous Real World Internet Data. *IEEE Sensors Journal*, **13**, 3793-3805. http://dx.doi.org/10.1109/JSEN.2013.2271562

[28] Eid, M., Liscano, R. and El Saddik, A. (2007) A Universal Ontology for Sensor Networks Data. *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, Ostuni, 27-29 June 2007, 59-62. http://dx.doi.org/10.1109/cimsa.2007.4362539

[29] Hodge, V.J. and Austin, J. (2004) A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review*, **22**, 85-126. http://dx.doi.org/10.1023/B:AIRE.0000045502.10941.a9

[30] Kolozali, S., Bermudez-Edo, M., Puschmann, D., Ganz, F. and Barnaghi, P. (2014) A Knowledge-Based Approach for Real-Time IoT Data Stream Annotation and Processing. 2014 *IEEE International Conference on Internet of Things* (*iThings*), *and Green Computing and Communications* (*GreenCom*), *IEEE and Cyber, Physical and Social Computing* (*CPSCom*), Taipei, 1-3 September 2014, 215-222. http://dx.doi.org/10.1109/ithings.2014.39

[31] Apache Mahout: Machine Learning Library Official Website. http://mahout.apache.org/

[32] Apache Cassandra Official Website. http://cassandra.apache.org/

[33] Node JS Express Framework Official Website. http://expressjs.com/