2016-10-18

A Model-Based Sensor Database for Internet of Things

Parul Maheshwari
University of Miami, p.maheshwari@umiami.edu

UNIVERSITY OF MIAMI

A MODEL-BASED SENSOR DATABASE FOR INTERNET OF THINGS

By

Parul Maheshwari

A THESIS

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Master of Science

Coral Gables, Florida

December 2016

UNIVERSITY OF MIAMI

A thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science

A MODEL-BASED SENSOR DATABASE FOR INTERNET OF THINGS

Parul Maheshwari

Approved:

_____	_____
Dilip Sarkar, Ph.D. Associate Professor of Computer Science	Geoff Sutcliffe, Ph.D. Professor of Computer Science
_____	_____
Manohar Murthi, Ph.D. Associate Professor of Electrical and Computer Engineering	Guillermo Prado, Ph.D. Dean of the Graduate School

MAHESHWARI, PARUL (M.S., Computer Science)
A Model-Based Sensor Database For Internet Of Things (December 2016)

Abstract of a thesis at the University of Miami.

Thesis supervised by Professor Dilip Sarkar.
No. of pages in text. (39)

The Internet of Things (IoTs) is becoming ubiquitous in our everyday lives, implying that more technologies will generate data. IoT devices use sensors to monitor various attributes of the environment such as temperature, humidity, light, etc. These sensors produce data periodically and storing this massive data in a database is becoming a huge challenge in the data storage infrastructure. Prior research has proposed compression algorithms and signature techniques to reduce data storage but do not specify how the data patterns are defined. Since similar patterns are exhibited everyday by the environment, this data generates the same information from everyday sensing. Therefore, in this study, we propose a system that stores data models rather than storing raw data points. Instead of storing each data point at a time, we develop and store data models with the corresponding time periods that captures the behavior of the sensor data. This helps in reducing data storage requirements. The data models developed are mathematical polynomial models that fit a sample data set. In addition, we propose a sensor database structure that addresses the issues of data redundancy as well as temporal constraints in the database.

Acknowledgements

I would like to thank the following important people who have supported me during the course of this study.

Firstly, I would like to express my gratitude to my advisor, Dr. Dilip Sarkar, for his unwavering support, guidance and insight throughout this research project. I would also like to thank my committee, Dr. Geoff Sutcliffe and Dr. Manohar Murthi for agreeing to review and provide valuable comments on my research. I also want to thank Department of Computer Science for providing me with graduate assistantship to support my studies, without which this journey would not be possible. I would also like to thank my fellow graduate students for providing their valuable inputs and helping me prepare this thesis. I am extremely grateful to my husband, Dr. Pratik Nyati, for believing in me and standing by my side always. I am also very thankful to my son, Pransh Nyati, for being so supportive - even when I was away from him. Finally, I would like to thank all my family members for helping me survive all the stress throughout the work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Internet of Things (IoTs)

The Internet of Things (IoTs) is the network connecting various objects to the Internet through different information perception devices, so that these individually addressed physical objects are able to exchange information with each other, and hence accomplish the target of recognition, monitoring, tracking, and management. In a typical IoT application architecture, the perception layer of the IoT comprises of a large number of Wireless Sensor Network (WSN). These WSNs are made up of sensors that sense the environment and act as a bridge between the machines and the physical world [17]. There are many technologies involved in the perception layer, such as sensor network design and performance evaluation, radio frequency identification (RFID) technology, sensing chip design technology, internet access technology, multi-sensor information fusion, and so on [9]. The devices in the perception layer of the IoT includes cameras, RFID readers, etc. In

WSNs, sensors are used to measure some properties of the environment in which they are installed, such as pressure, radiation, temperature, humidity, light, and many more, providing raw data for information systems.

There are various types of sensors and each sensor performs different functions. For example, logistic and agricultural applications have temperature, humidity and light sensors, whereas medical devices have sensors that measure heart rate, blood pressure, and other body functions. Therefore, the data obtained from different sensors is heterogeneous [5]. Also, the sensor data are redundant as the environment that the sensors are monitoring can be stable for short time periods and hence, same data is produced by the sensors [6]. Sensors collect data over time and provide real-time data [2]. The real-time data when processed can bring profit to IoT. Thus, when an unexpected event occurs, a timely exception detection can be helpful to make decisions to reduce and avoid the loss of data. Moreover, as the production cost of sensors is very low, large number of sensors are deployed in the environment to get complete information about the surroundings. Also, the sensors are gathering data at frequent intervals resulting in large volumes of data generation. Therefore, sensors develop huge amount of heterogeneous, continuously streaming and geographically-dispersed real-time data. This results in the issue of communication overhead and database will be heavily loaded, growing very fast and performance will drop if all of this data is stored. Hence, there is a need to find a way to efficiently store this massive volume of data.

In addition to the data generated by the IoT objects, there is metadata that defines and describes these objects, such as object identification, location, services

provided, etc. Also, IoT data, unlike the traditional Internet data, possess the time and space attributes that represent the dynamic state changes of an objectâĂŹs location over time. Therefore, communication, storage and process will play a major role in designing the data management solutions for IoT.

IoT data storage mainly have three schemes: local [14], distributed [4], and centralized [11]. In the local scheme of data storage, each sensor has its own local database unit. Thus, the sensor nodes in the WSN runs its own database management system. In the distributed scheme, data is stored in some sensor nodes in WSN and this is done using distributed technologies. Intermediate tools are used to provide data access. In the centralized scheme, the data of the network is gathered by a node, then sent to a data center where all the data of the network is stored. Since the storage capacity and battery power of the sensors is very limited, the local and distributed form of data storage is not suitable for IoT systems as IoTs produce huge data. As a result, the centralized form of data storage is more suitable for IoT applications.

In addition, the issue that most of the data generated from the sensors is redundant does not provide any extra information. This data is still being stored as we do not have any system yet that can detect the data and avoid redundancy. If we are able to develop such a system then it will provide great solutions for future databases.

Therefore, we propose a model-based data storage design solution for IoT applications that can resolve the problem of huge-scale and complex IoT data.

1.2 Overview of the thesis

The rest of this thesis is organized as follows. Chapter 2 discusses the overview of an IoT system that includes sensor network and IoT data, and provides details of the system components. Chapter 3 provides a brief review of the related work. Chapter 4 presents the details of the algorithm that develops data models. These data models are mathematical polynomial models that are generated from sensor raw data. Hence, the sensor data can be represented by these data models. Chapter 5 provides the IoT hierarchical structure that contains IoT objects, sensors in IoT devices, and the models for sensor readings. It further describes the IoT database architecture and few example queries for the IoT database. Chapter 6 provides a detailed description of the experiments conducted on the available sensor data set and the different data models generated by varying the input parameters of the algorithm. Finally, Chapter 7 provides the summary of the thesis and outlines potential new direction in the future.

Chapter 2

Background Information

2.1 System Overview

The integration of physical devices into data networks has progressed a lot in the recent years and it is setting a new pattern in the world of IoT. The data collected from various devices in sensor networks is in the form of physical environment measurements which are communicated to other end-user devices via the Internet. The data communication between the sensor network and the Internet can be done using a gateway node. A gateway node has the power to convert disparate formats of the raw data to standardized formats, thus further reduction in transmission depends on this processing done at the gateway node. To achieve the goal of reducing raw data in an efficient manner, a model-based scheme can be well-established that can predict the data and answer queries of the user just by selecting the correct model for the data instead of accessing a database that stores all the processed data.

2.2 System Components

Figure 2.1 shows the overview and the main components of an IoT system. In the following subsections, we briefly describe function of each component.

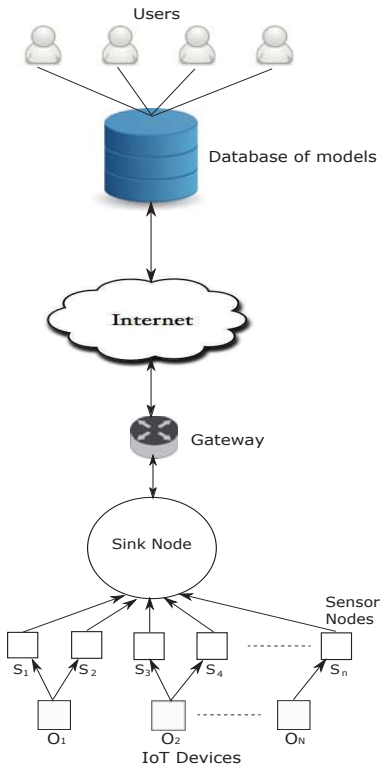

Figure 2.1: Block diagram of a typical IoT database

2.2.1 Sensor Network

A sensor network is a group of different sensors that are embedded in various IoT devices to monitor the physical or environment conditions and has the ability to communicate with each other to form a network. Sensor nodes play an important role in the network. They can be static nodes or mobile nodes. After sensing the environment, the sensor nodes generates the data packet and sends it to the sink

node via the sensor network. Each sensor node has a micro-controller with one or more sensors. The micro-controller is equipped with limited memory and limited computing power.

2.2.2 Sink Node or Aggregator Node

A sink node gathers data from all sensor nodes connected to it. It is also known as an aggregator node, because it collects readings from multiple sensor nodes. The main functions of an aggregator node are gathering data from the connected sensor nodes, pre-processing and aggregating the raw data to produce an estimated reading, and communicating the processed data to the base station. Thus, aggregator is used to summarize and merge operations in real-time to squeeze the massive volume of data to be stored and transmitted [10]. In comparison to a micro-controller, an aggregator has bigger memory and higher computing capability.

2.2.3 Gateway

A gateway is a node that connects two networks. In our IoT system, a sink node is connected to the Internet through a gateway. It is able to transform and standardize the data. It gathers data in disparate formats generated from multiple sensor nodes and converts them to a standard format to be used in the next stage of data processing. Another role of gateway is protocol transformation. It contains multiple communication protocols to accept outgoing data sent by the sink nodes, forward the data to the Internet and communicate the incoming data from

the Internet to the sensor network.

2.2.4 The Internet

The Internet is a medium of connecting a network to another network using the Internet protocol suite (TCP/IP). It connects a computer to any other computer globally forming a network. When two computers are linked over the Internet, they can communicate with each other and send and receive variety of information. For our application, data from sensors in the IoTs are communicated via the Internet.

2.2.5 Database System

A database is a coherent collection of related data. The data is stored in an organized manner to make it easily accessible and manageable. The database users can search for, retrieve and update the data as needed in the database. The queries from users are sent to the database which in return sends the appropriate selected data. In our approach, the database consists of records of different types of IoT devices and their sensors that measure and gather the raw information from the environment they are deployed in. From these raw data points generated periodically, we build and store data models over a range of time period.

2.3 Problem Statement

As IoT devices are increasingly becoming a part of our daily lives, interaction between these devices is producing more and more data. In wireless sensor networks, sensors are generating data periodically and hence the data is growing rapidly. Generating the data is easy but the issue is to manage and store this large volume of data for desired application.

While databases are becoming more efficient and data storage capability is expanding, there will come a stage when data overwhelms data storage. With information communication technologies producing data at an accelerated rate, it is inevitable that the storage space will not be enough to store all this data.

Fortunately, data generated by IoT sensors is redundant and hence does not provide any extra information. Each of the sensor data points are still being stored because currently, we do not have any system that can model such data and store the models instead of the raw data. For example, while monitoring the temperature of any environment, the sensors gather temperature readings as raw data points at regular time interval. The deviation in temperature readings between adjacent time intervals is generally insignificant during large parts of the day and is only affected by few external factors such as the geographic locations and seasons. If we are able to avoid storing each of these temperature readings for adjacent time intervals and develop and store prediction models to minimize the redundancy, we could store less amount of data ensuring minimal loss of information and hence reducing the size of the database.

Moreover, cost of sending and receiving data from one sensor node to another is much higher than that of gathering data and performing local and in-network data processing. In other words, the cost of communication is greater than the cost of computation. In addition, transmitting and receiving a data packet consumes a considerable amount of energy. Hence, the consumption of network energy should be minimized in order to fully utilize the sensor network. This can be achieved by minimizing the data flow in the network.

With large numbers of sensor nodes comes the problem of scalability, need of large memory and data redundancy. The solution to these issues is generating data models from the raw data points over certain time intervals and designing a model-based database system that stores these models with the respective time periods, hence providing with an efficient way to address these temporal constraints.

Chapter 3

Related Work

3.1 Sensor Data Models

In the aspect of avoiding redundant data generated from the sensors, data models in [12] are created using polynomials while the sensor node is providing new samples. When adding points to the polynomial, the algorithm tries to add as many points to the polynomial by adding degrees to the polynomial to fit the data. If the data does not fit, the polynomial keeps adding degrees until the maximum number of degrees is added. If the value point does not fit within the maximum number of polynomial degrees, the polynomial is stored with the timestamp, then, a new polynomial of degree zero is created to fit the next sample, and the process is restarted. This algorithm is an online segment construction based on live machine learning. Versions for model elements are created if an attribute in the IoT object has changed, but this research has not reused their models for future time intervals. Rather, they create new polynomials for each time interval unlike this

work that tries to predict future trends to reduce sensor traffic.

In [16] a Time Series database service has been created to support pattern searches in the IoT domain. For query processing, TSaaS (Time Series as a Service) partitions the pattern into segments, and searches for a similar pattern using a systematic approach.

The authors in [7] compare representative techniques like piecewise constant approximation, adaptive piecewise constant approximation, slide filters, from categories based on constant, linear, non-linear and correlation models according to data reduction and prediction accuracy. The study concludes that constant and linear models outperform the others in the presence of small variations in the data. As compared to these results, the data models created in this work are based on linear and higher degree polynomials.

The authors in [19] presents an approach that sends and receives information using signatures instead of raw sensor data. The signature is a string representation implying that something is present or absent at a particular sensor. The problem of data communication is overcome because only binary data is transmitted; but how the data patterns are defined is not specific enough. In our application, the sensor readings are understood and analyzed with the proposed model-based method that may learn patterns out of it and further forecast the real time data.

3.2 Database Architecture

Thantriwatte et al. [13] developed a query processing system in WSN based on NoSQL database, but the work done is quite elementary. Instead of targeting the IoTs, it mainly talks about WSN, and the issue of how to store such huge amount of IoT data along with solution to adequately organize and manage this data was not addressed in the article. Also, query optimization performed is not as good as in the relational databases.

Jain et al. [8] presents a method that reduces the amount of data communicated in distributed data streams environment by using Kalman Filters. Also, to maintain the location of moving objects in the database is another concern. When a moving object changes its location, updating the server about its new location is again expensive. According to [15], the trajectory of the moving object can be kept track of, and only when the object changes its trajectory, or does not operate according to the trajectory, then the update is sent to the server. In this work, we propose an approach that handles temporal correlations in the data. This can prove to be very effective in sensor networks.

Chapter 4

Sensor Data Models

Sensor data models provide an efficient way to represent data and minimize storage space with the same data utility. The actual data reading gathered by sensors from their environment are raw data points. Instead of storing these raw data points in the database, we can efficiently utilize the storage space by representing groups of similar raw data points in the form of mathematical equations. Therefore, we adequately manage the storage space by storing these mathematical equations (data models) in the database. Hence, in order to retrieve a raw data reading, instead of fetching a raw data point, we retrieve a data model that corresponds to this data point. We calculate the data value using the data model retrieved.

4.1 Generating Data Models

The data models are generated from a set of raw data points, and they are stored in the database against a time interval - this data model is considered an effec-

tive representation of raw data points that were observed at timestamps which lie within this time interval. The mathematical models, M_1, M_2, M_3, ..., M_N, are polynomial equations that are stored in the database in the form of numeric coefficients of the equation with the corresponding time periods $T_1, T_2, T_3, ..., T_N$. Let us denote these coefficients as a_n, a_{n-1}, a_{n-2}, ..., a_0 where n is the degree of the polynomial equation. We find the coefficients of a polynomial that fits the raw data points by minimizing the sum of the squares of the deviations of these data points from the value provided by the model. The models will be stored in the form of their polynomial coefficients as they will be mathematical functions of the form as shown in Equation 4.1, where the data value Y at an input time period x is calculated by providing the coefficients a_n, a_{n-1}, a_{n-2}, ..., a_0 of the mathematical polynomial model of degree n.

$$Y = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n \tag{4.1}$$

Generation of Sensor Data Model (Algorithm 1) describes an algorithm that develops data models from a set of raw sensor data, $sensorData[]$ with a given error threshold, γ. The raw sensor data is the data set generated by the sensors that monitor the environment in which they are installed. The error threshold is the maximum amount of deviation between the raw data point and the acceptable predicted value generated by the algorithm. Each element in sensor data includes the actual data reading, $actualValue$ with the corresponding timestamp, $timestamp$ when the data was observed. The predicted value, $predictedValue$ calculated from

Algorithm 1 Generation of Sensor Data Model

```
1: procedure GENERATIONOFSENSORDATAMODEL(maximumDegree, γ)
2:     sensorData[] < timestamp, actualValue >          ▷ Raw sensor data
3:     degree = 1;
4:     low = 0;
5:     high = sensorData.size();          ▷ Number of records in sensorData[]
6:     for all records in sensorData[] do          ▷ Generate acceptable models
7:         executePolyfit(low, high, degree)
8:         loop
9:             if (degree > maximumDegree) then
10:             ▷ Do binary division on all records by calculating the middle index
11:                 mid = low + (high-low)/2;
12:                 executePolyfit(low, mid, degree);
13:                 executePolyfit(mid, high, degree);
14:             end if                          ▷ Get coefficients of the polynomial
15:             p = getPolyfit(low, high, degree);
16:             for sensorData[low] to sensorData[high] do
17:                 loop                          ▷ Calculate predicted value
18:                     predictedValue = polyval(p, timestamp);
19:                     absoluteError = abs(predictedValue - actualValue);
20:                     if (absoluteError > γ) then     ▷ Data model is not acceptable
21:                         executePolyfit(low, high, degree++);
22:                         return;
23:                     end if
24:                 end loop
25:             end for
26:             store data model p in database
27:         end loop
28:     end for
29: end procedure
```

the algorithm is accepted only if its value lies within the error threshold. We generate the acceptable data models in a function named $executePolyfit(low, high, degree)$ which is carried out on $sensorData[]$ starting from its record which is the one at its lowest index, low to the one at its highest index, $high$ with the degree of the polynomial, $degree$. Our algorithm is initialized with the value of low equal to 0 and $high$ equal to the size or number of records in $sensorData[]$ calculated by the function $size()$. The function $getPolyfit(low, high, degree)$ generates the coefficients, p of a polynomial that fits the raw data points by minimizing the sum of the squares of the deviations of these data points from the value provided by the polynomial. Considering the whole sensor data at once, firstly, we try to obtain the polynomials with degree one, i.e., we find coefficients a_0 and a_1 for linear polynomials in such a way that the absolute difference, $absoluteError$ between $actualValue$ and $predictedValue$ is always less than the given error threshold. The predicted value is calculated by the function $polyval(p, timestamp)$ that returns the value of a polynomial of degree $degree$ evaluated at $timestamp$ taking the input argument p as the coefficients of the polynomial returned by the function $getPolyfit(low, high, degree)$. If a linear model is found such that the values predicted by it has an error less than the given error threshold, it implies that the linear model fits the sensor data, and hence we can accept this linear data model with the corresponding time period of the sensor data. As a result, when the user enters a query, we can retrieve the coefficients of this model and calculate the predicted value at any given input timestamp as described further in Chapter 5 and Chapter 6.

However, if the sensor data does not fit with this linear model, then we find the

coefficients for a quadratic data model, i.e., polynomial with degree two. If this two degree polynomial falls under the error threshold, then it means that all the sensor data can be represented with this polynomial and hence we have found the acceptable data model. This process of finding the coefficients for the whole sensor data set continues until we reach the maximum degree of polynomial, *maximumDegree* given as an input parameter for the algorithm.

Considering the sensor data set as a whole, even after reaching the maximum degree, if there is no polynomial found that can be accepted with the given error threshold, then a binary division needs to be conducted on the whole data set. The binary partition divides the data set into two equal halves by calculating the middle index, *mid* of *sensorData*[] and then we find the coefficients for the data set in these two halves. To get the data model, we repeat the process with first finding the linear polynomial that fits with the given error threshold. If it does not fall under the error threshold, then we find the polynomials with higher degrees. This process of finding sensor data models is carried out until all the sensor data can be represented by acceptable polynomial equations. The models guarantee that every value they output never has an error greater than the given error threshold.

Chapter 5

Sensor Database

Nowadays, sensors are everywhere and the data they are producing is growing at a phenomenal rate. The sensors gather information of various phenomenon in their environment on a regular basis. Thus and so, a large number of phenomenon readings are generated every day. And therefore, it is becoming difficult to store, manage and analyze this large volume of data. As a solution to this, we have developed an algorithm for these phenomenon readings that converts raw data points into data models, as discussed in Chapter 4, and in addition, storing this large number of models still requires huge amount of space. To find a better way to store and retrieve the data models and hence the phenomenon readings efficiently supporting the environmental information, we further discuss the structure of our sensor database.

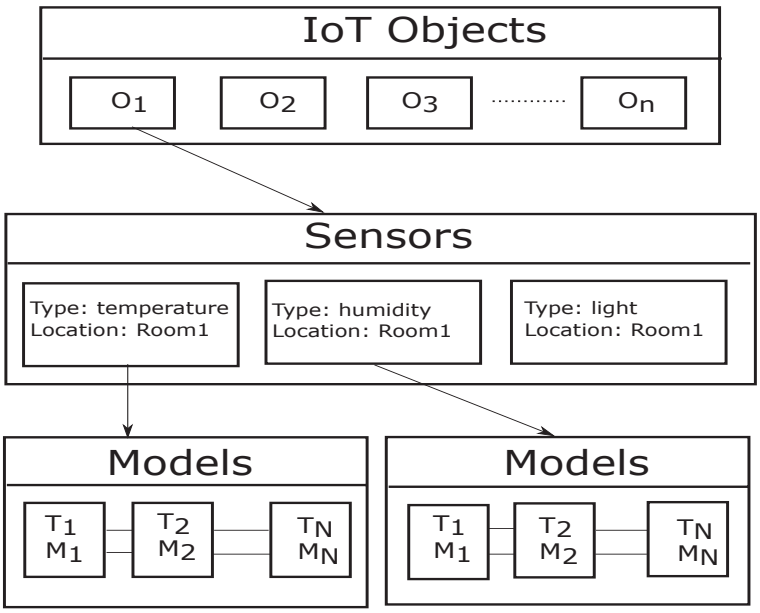

Figure 5.1: IoT hierarchical structure

5.1 IoT Hierarchical Structure

We are moving from a time where there are millions of devices connected to a network today to a time where there will be billions of devices connected to this network. We need to create a hierarchical structure, as shown in Figure 5.1, that makes query processing easier by creating a logical flow between IoT objects.

5.1.1 IoT Objects

An IoT database is organized by different data coming from IoT devices or objects. Therefore, the user needs to specify what IoT object they want to search for. The IoT objects consists different kinds of information sensing devices such as speedometers, rain gauges, microphones, and many more that have variety of sensors like automotive sensors, environmental sensors, acoustic and sound sensors,

etc [1]. These objects can be connected to the Internet and hence data or information collection, transmission and processing between these devices can be achieved in an intelligent and effective manner [18].

5.1.2 Sensors in IoT Devices

The sensors hierarchy contains all the sensors that are embedded in the IoT devices. For example, a car have different types of sensors such as speedometer, manifold absolute pressure (MAP) sensor, blind spot monitor and many more that measures speed, pressure, blind spot detection, etc. These sensors produce data which can be represented by mathematical models that predicts the data.

5.1.3 Models for Sensor Readings

The models hierarchy contains one of the most important pieces in the database which is the data model. The data models will be in the form of polynomial equations whose coefficients will be represented as the parameters of the data model. The models in the hierarchy will be organized by timestamps in a sorted fashion. This is explained further in section 5.3.

5.2 Database Architecture

Our model-based IoT database is a database management system (DBMS) built for IoT objects and their various sensors. It presents a set of relational database operations that helps in creation of the database and solves complex data queries.

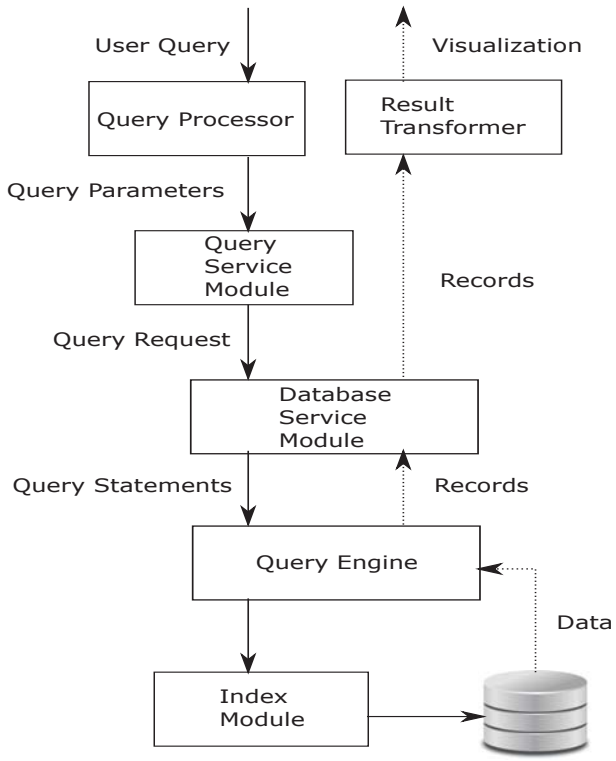

Figure 5.2: IoT database architecture

Figure 5.2 proposes an underlying architecture of an IoT database. When a user enters any query, it is parsed by the query processor. The query service module builds the query according to the query parameters and sends the query request to the database service module. The database service module holds the database logic and sends the query statements to the query engine. The index established in the index module will be utilized to retrieve the requested data. After the required data is obtained from the database, the query engine sends the records to the result transformer that presents the query results in a format easy to use by the user.

5.3 Terminology

For IoT database, we use the standard terminology from the concept of relational database [3]. A relational database is a collection of information related to a particular topic or purpose. It specifies the data types, structures, and constraints of the data to be stored. A database management system is a collection of programs that enables users to create and maintain a database. A relation or a table is a format of rows and columns that displays related information. An attribute is a specific item of information that contains a homogeneous set of values throughout the table. Attributes appear as columns in a table. A record is an individual listing of related information that contains a number of related attributes stored in a table. A record appear as rows in a table. Each attribute has a domain that depicts the data type of the attributes. To avoid duplication of information, a relation has a primary key that uniquely identify each record in the relation. To retrieve a particular collection of information in a database, a query can be send to the database.

Figure 5.3 shows an example of a relation named as *objectModels* that contains four records of sample information collected by some sensors that monitor a particular environment. The column names such as $objectID$, $location$, $timestamp$, and *dataModel* are the attributes of this relation where $timestamp$ is the primary key stating that each record will be uniquely identified by timestamp. In other words, no two records can have the same timestamp. Hence, by knowing the timestamp of a particular object, we can retrieve its objectID, location, and data model. Moreover, the definition of a sample record is shown in Equation 5.1, where $objectID \in$

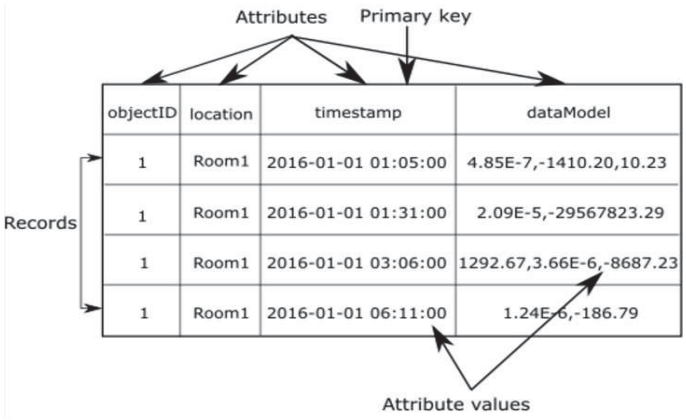

Figure 5.3: Relation objectModels containing sample information

Integer is the ID of an IoT object; $location \in String$ is the location where the object was installed while monitoring; $timestamp \in Timestamp$ is the date and time of the format $yyyy - mm - dd\ hh : MM : ss$ where $yyyy$, mm, dd, hh, MM and ss represent the year, month, date, hour, minute and second respectively when the data was observed; $dataModel \in String$ represents the coefficients of the polynomial equation of the data model as a comma separated value.

$$SampleRecord = (objectID, location, timestamp, dataModel) \tag{5.1}$$

5.4 Query IoT Model Database

Queries play a major role in the abstraction of data from a database. A relation is defined by using a set of operations. It consists of a set of attributes, their data

types and a set of constraints on the records to be inserted in the relation. The CREATE TABLE operation specifies the layout of a relation. Once a relation is defined, the data records are inserted in it. While inserting, the database verifies that the attribute values satisfies the domain or the data type of the corresponding attribute. The SELECT operation is used to retrieve a set of records from a relation. With the REMOVE operation, records can be deleted in a relation.

5.4.1 Query Syntaxes

We need to define a set of query syntaxes for our IoT database.

1. **getByLocation:** It returns all the *SampleRecord* from the database where the value of specified location is equal to the value of the attribute *location* in the *objectModels* relation as discussed further in Chapter 6.

2. **getElement:** It returns all the *SampleRecord* based on a key. This key can be any attribute such as timestamp, location or dataModel depending on the requested data.

3. **getByTimeRange:** It returns all the *SampleRecords* from the database where the requested time range lie between the values of the attribute *timestamp* in the *objectModels* relation as discussed further in Chapter 6.

5.4.2 Query Examples

Let us consider a few query examples that can be carried out on the IoT database example as discussed in Figure 5.3.

Example 1: *query the temperature of the Robocanes room at time 01:25:32 on 2016-01-01.*

In order to obtain the temperature value, firstly, we will retrieve the data model from the database corresponding to the input timestamp. The database contains the data models in the form of coefficients of a polynomial equation in the relation *objectModels*. The attribute values in the relation is denoted by *value*.

for each SampleRecord in objectModels

getByLocation(SampleRecord, location).value = "Robocanes"

AND getElement(((SampleRecord, timestamp).value >= "2016-01-01 01:25:32")

limit 1)

return getElement(SampleRecord, dataModel).value

Suppose, the data model retrieved corresponding to the input timestamp 2016-01-01 01:25:32 is (2.09E-5,-29567823.29). Then, the polynomial equation formed will be

$$Y = (2.09E - 5 * x) - 29567823.29 \tag{5.2}$$

where x is the input timestamp.

Next, we convert the timestamp into epoch time that gives us the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC),

Thursday, 1 January 1970 and input this value as x in the mathematical equation. This will give us the value of the temperature on 2016-01-01 at 01:25:32.

Example 2: *query the location of those rooms where temperature is greater than 22.34 degree Celsius between January 2, 2016 to January 4, 2016.*

Firstly, we retrieve all the records from the database from January 2, 2016 00:00:00 to January 4, 2016 23:59:00. Then we input the corresponding timestamps and the model coefficients resulting in the predicted temperature values for this time range. Then within these records, we will select only those records where the data models predict the value of the temperature greater than 22.34 and return the respective locations.

for each SampleRecord in objectsModels

getByTimeRange(SampleRecord, "2016-01-02 00:00:00","2016-01-04 23:59:00")

AND getElement((SampleRecord, temperature).value > 22.34)

return(getElement(SampleRecord, location).value)

As a result, we will get all the locations where temperature is greater than 22.34 degree Celsius within the time range January 2, 2016 to January 4, 2016.

Chapter 6

Experimental Results

This chapter provides experimental results that we obtained from the implementation of our proposed Generation of Sensor Data Model algorithm that develops data models from a set of raw data points. We also developed a relational database to store these data models and perform user query execution.

6.1 Experimental Settings

For this experiment, we have installed a TM4C1294 Connected LaunchPad Evaluation Kit which is a low-cost development platform ARM Cortex-M4F-based micro-controllers. We have also installed a Texas Instruments Sensor Hub BoosterPack (BOOSTXL-SENSHUB) which is an add-on board that provides a platform for evaluating the use of ARM Cortex-M4F-based TM4C devices in sensor fusion applications. This booster pack features seven different kinds of sensors such as MPU-9150 motion tracking sensor, BMP180 pressure sensor, SHT21 humidity

and ambient temperature sensor, ISL29023 ambient and infrared light sensor, and TMP006 infrared temperature sensor. We have installed these around our offices at the University of Miami. We have used Java and Matlab 9.0 (R2015b) as our programming platform to use the polynomial fitting tools and build the algorithm presented in Chapter 4.

6.2 IoT Model Database

In our relational database, we construct a relation known as objects, as shown in Table 6.1, that contains all the information about an IoT object such as its ID, name and type. This relation is created using a set of queries, as shown in Figure 6.1, where *objects* is the name of the relation in a database. It contains three attributes - *objectID*, *objectName*, and *objectType* where *objectID* is the primary key of this table. These attributes represent the identification numbers, name and type of various IoT objects.

Table 6.1: Relation objects

Attribute Name	**Data Type**
objectID	int
objectName	varchar (45)
objectType	varchar (45)

We construct another relation known as objectModels, as shown in Table 6.2. It includes all the data models of sensor data signal that the corresponding *objectID* has along with the timestamp and location at which data models are obtained. A

```
-- Table `mydb`.`objects`

CREATE TABLE IF NOT EXISTS `mydb`.`Objects` (

 `objectID` INT NOT NULL AUTO_INCREMENT,

 `objectName` VARCHAR(45) NULL,

 `objectType` VARCHAR(45) NULL,

 PRIMARY KEY (`objectID`))
```

Figure 6.1: Set of queries for objects relation

sensor generates one single reading at a time, hence only one relation is created for every sensor. If multiple objects are there then multiple relations will be created to store corresponding data models. The relation objectModels is created using a set of queries, as shown in Figure 6.2, where *objectModels* is the name of a relation in the database named *mydb*. The attributes of this relation are *objectID*, *location*, *timestamp*, and *dataModel* where *timestamp* is the primary key and *objectID* connects this relation to the other relation *objects* by representing it as a foreign key.

Table 6.2: Relation objectModels

Attribute Name	**Data Type**
objectID	int
location	varchar(50)
timestamp	timestamp
dataModel	varchar(50)

We populate our database with the data models with the corresponding timestamps and locations with an error threshold of 0.10 and maximum degree equal to 2. Currently, our database contains 12 days' worth of data starting from January

```
-- Table `mydb`.`objectModels`
CREATE TABLE IF NOT EXISTS `mydb`.`objectModels` (
 `objectID` INT NOT NULL,
 `location` VARCHAR(50) NULL,
 `timestamp` TIMESTAMP NULL,
 `dataModel` VARCHAR(50) NULL,
 INDEX `objectID_idx` (`ID` ASC),
 PRIMARY KEY (`timestamp`),
 CONSTRAINT `objectID`
 FOREIGN KEY (`objectID`)
 REFERENCES `mydb`.`objects` (`objectID`)
 ON DELETE NO ACTION
 ON UPDATE NO ACTION)
```

Figure 6.2: Set of queries for objectModels relation

1, 2016 00:00:00 to January 12, 2016 23:58:00. This consists of 3448 original data points generated by the SHT21 ambient temperature sensor. The total number of data models produced are 549 which are then stored in the *objectModels* table.

6.3 Results

To generate the models, we used past data from a SHT21 temperature sensor that returns the temperature in degrees Celsius. Then, we applied the Generation of Sensor Data Model algorithm to produce our data models. The algorithm was applied to 12 days' worth of data to capture all the behaviors that the data exhibits throughout the day.

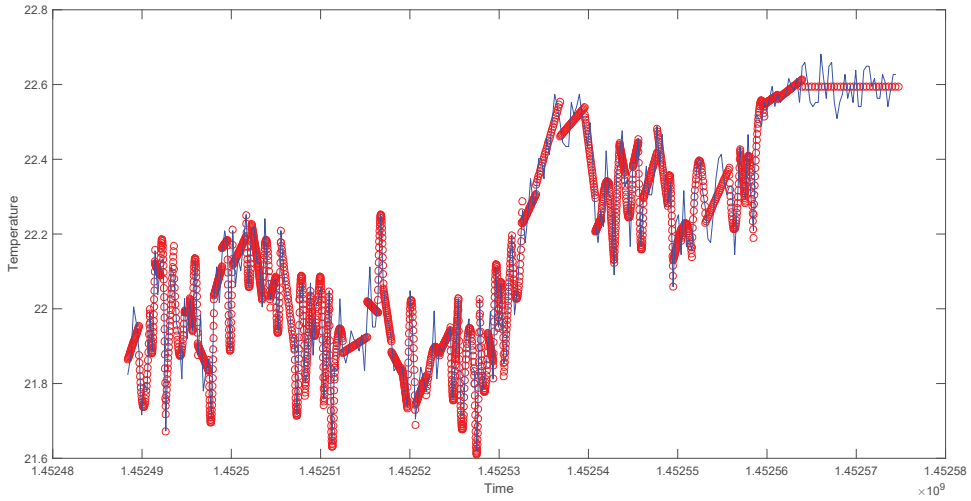

Figure 6.3: Graph showing polynomial models generated from sensor data obtained on January 11, 2016 with error threshold set to 0.10

In Figure 6.3, 78 polynomial models were generated for January 11, 2016 00:00:00 to January 11, 2016 23:59:00. These models were generated using error threshold equal to 0.10 degrees Celsius and maximum polynomial degree equal to 2. The maximum polynomial degree is set to 2 and does not need to be high as all the data points are represented by maximum 3 coefficients reducing and saving the storage space. This sample data set has 288 data points generated daily every five minutes. Therefore, our algorithm produced only 78 models that can predict the same data with a 0.10 degrees Celsius error threshold. The Figure 6.4 depicts a part of the database where the data models are stored in the form of coefficients of the polynomial equations produced.

Table 6.3 shows the number of data models retrieved with error threshold equal to 0.10 degrees Celsius but maximum degree for the polynomials to be obtained is varied from 1 to 5. Thus, we produced 121 linear models with the constraint of

objectID	location	timestamp	dataModel
1	Robocanes	2016-11-01 19:03:00	4.927656221940725E-7,-1431.5396398972812,1.0396959773270846E12
1	Robocanes	2016-11-01 19:13:00	-2.316666667717799E-4,336531.4553526831
1	Robocanes	2016-11-01 19:28:00	-7.409584197647818E-7,2152.5700654101647,-1.56336636538405E12
1	Robocanes	2016-11-01 19:48:00	-4.004958645795447E-7,1163.4880275363887,-8.450177079890381E11
1	Robocanes	2016-11-01 20:13:00	1.6380952371185535E-5,-23771.76388676544
1	Robocanes	2016-11-01 20:59:00	1.8822971559628637E-5,-27318.956594274845

Figure 6.4: Models generated from raw data gathered by sensor

maximum polynomial degree as 1. Moreover, when we increase the value of maximum degree of the polynomial, the number of data models obtained are reduced.

Table 6.3: Number of models generated with error threshold set to 0.10

Highest order of polynomial	**Number of models**
1	121
2	78
3	78
4	77
5	77

Similarly, Table 6.4 shows the number of data models retrieved with error threshold set equal to 0.25. Hence, we get 20 models considering only the linear polynomials allowed. This number decreases as we raise the maximum degree of polynomials.

Example User Query: *Show the temperature of the room named 'Robocanes' at time 03:59:23 on January 11, 2016.*

First, we run the following query on the database to retrieve the data model of

Table 6.4: Number of models generated with error threshold set to 0.25

Highest order of polynomial	Number of models
1	20
2	14
3	13
4	13
5	13

temperature sensor with this corresponding timestamp.

Query: *SELECT dataModel FROM objectModels where location = 'Robocanes' and timestamp >= '2016-01-11 03:59:23' limit 1;*

The corresponding data model retrieved is a linear model with coefficients (a_1, a_0) = $(-1.4333333346432084E-4, 208214.21779025975)$. Hence, the required temperature value is equal to (a_1 * epoch time of *2016-01-11 03:59:23*) + a_0 which gives the resulting temperature value as 22.15 degrees.

6.4 Discussion

As sensors produce large volume of raw data everyday, these data points are transformed into polynomial data models with respect to the timestamps at which the raw data was collected, and then instead of storing each and every data point, these polynomial models are stored in the database. Referring to Tables 6.3 and 6.4, we can reduce the number of data models to be produced by allowing polynomials of higher order. As a result, the amount of storage space the sensor information occupies is reduced from 3448 original data points to 549 data models without losing

any information. Considering a days' worth of data with 288 raw data points and generating data models out of it with maximum degree of polynomials is equal to 2, the number of models obtained are 14. This implies that each model will have maximum 3 coefficients that will represent the data points resulting in 3 * 14 = 42 numbers to store instead of 288 numbers. Therefore, instead of storing each raw data point, we build data models out of it. Hence, we have successfully reduced the data points to a compressed set of data models.

Chapter 7

Conclusion

With the increasing trend of information communication technologies, data is being generated at very high rates. Data is becoming very hard to manage and an efficient way to organize data in databases is an important issue. IoT model databases is becoming an important notion to alleviate data generation by decreasing the space that data consumes while also maintaining the same information.

Data models also provide data with a negligible error that can fit many raw data points from sensors. These models are created by fitting a function to the data points. In this research, we used polynomials with different order, for example, first order, second order, etc to fit the data points. Our algorithm, Generation of Sensor Data Model, finds a polynomial curve whose parameters are the coefficients of the polynomial equations. These parameters now cover many raw data points within a time range. In other words, with data models we can represent enormous amount of data points without having to overfill databases or sacrifice data utility.

Also, an IoT storage management architecture is proposed to meet the needs of

massive IoT data. It not only supports how to reasonably and effectively store big IoT data but is also concerned about how to respond to the queries satisfying the temporal and spatial correlation constraints.

As future work, more robust algorithms can be created to segment data into more accurate models using a set of mathematical functions other than the polynomials of higher degrees such as logarithmic functions. Finding the most probable model efficiently will also help the system save energy.

References

[1] Sensor | Types of Sensor. *http://www.electrical4u.com/sensor-types-of-sensor/*, 2011.

[2] L. Chen and K. D. Kang. A framework for real-time information derivation from big sensor data. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*, pages 1020–1026, Aug 2015.

[3] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[4] O. Diallo, J.J.P.C. Rodrigues, M. Sene, and J. Lloret. Distributed database management techniques for wireless sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 26(2):604–620, Feb 2015.

[5] W. Elmenreich and R. Leidenfrost. Fusion of heterogeneous sensors data. In *Intelligent Solutions in Embedded Systems, 2008 International Workshop on*, pages 1–10, July 2008.

[6] A. Ghaddar, T. Razafindralambo, I. Simplot-Ryl, S. Tawbi, and A. Hijazi. Algorithm for data similarity measurements to reduce data redundancy in wireless sensor networks. In *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*, pages 1–6, June 2010.

[7] Nguyen Quoc Viet Hung, Hoyoung Jeung, and K. Aberer. An evaluation of model-based approaches to sensor data compression. *Knowledge and Data Engineering, IEEE Transactions on*, 25(11):2434–2447, Nov 2013.

[8] Ankur Jain, Edward Y. Chang, and Yuan-Fang Wang. Adaptive stream resource management using kalman filters. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 11–22, New York, NY, USA, 2004. ACM.

[9] Xu Jianlong, Liu Guixiong, and Hong Xiaobin. Internet of things perception layer scenario abstract method research and application. *Advances in information Sciences and Service Sciences(AISS)*, 5(7), 2013.

[10] Chong Luo, Feng Wu, Jun Sun, and Chang Wen Chen. Compressive data gathering for large-scale wireless sensor networks. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, pages 145–156, New York, NY, USA, 2009. ACM.

[11] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, March 2005.

[12] A. Moawad, T. Hartmann, F. Fouquet, G. Nain, J. Klein, and Y. Le Traon. Beyond discrete modeling: A continuous and efficient model for IoT. In *Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on*, pages 90–99, Sept 2015.

[13] T.A.M.C. Thantriwatte and C.I. Keppetiyagama. NoSQL query processing system for wireless ad-hoc and sensor networks. In *Advances in ICT for Emerging Regions (ICTer), 2011 International Conference on*, pages 78–82, Sept 2011.

[14] Nicolas Tsiftes and Adam Dunkels. A database in every sensor. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 316–332, New York, NY, USA, 2011. ACM.

[15] Ouri Wolfson, Sam Chamberlain, Son Dao, and Liqin Jiang. Location management in moving objects databases. In *In WoSBIS*, pages 7–13.

[16] Xiaomin Xu, Sheng Huang, Yaoliang Chen, K. Browny, I. Halilovicy, and Wei Lu. TSAaaS: Time Series analytics as a service on IoT. In *Web Services (ICWS), 2014 IEEE International Conference on*, pages 249–256, June 2014.

[17] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

[18] Lei Yuan and Junsan Zhao. Construction of the system framework of spatial data warehouse in internet of things environments. In *Advanced Computational Intelligence (ICACI), 2012 IEEE Fifth International Conference on*, pages 54–58, Oct 2012.

[19] Mira Yun, Danielle Bragg, Amrinder Arora, and Hyeong-Ah Choi. Battle event detection using sensor networks and distributed query processing. *Computer Communications Workshops, INFOCOM WKSHPS , IEEE Conference on*, pages 750–755, April 2011.