A Framework for Integrating Wireless Sensors and Cloud Computing

by

Mohammad Jassas

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Applied Science

in

Electrical and Computer Engineering
University of Ontario Institute of Technology

Oshawa, Ontario, Canada
© Mohammad Jassas, 2015

Abstract

A Framework for Integrating Wireless Sensors and Cloud Computing

Mohammad Jassas
University of Ontario Institute of Technology, 2015

Advisor:
Professor Qusay H. Mahmoud

Wireless sensors generate a large volume of data that require a highly scalable framework that enables storage, processing, and analysis. Cloud computing technology can provide unlimited storage in addition to a flexible processing infrastructure, allowing for the management and analysis of vast amounts of sensor data. This thesis presents a framework for integrating wireless sensors and cloud computing. This framework can provide scalability and high availability for applications that use wireless sensors. Moreover, this cloud-based framework is designed to immediately make decisions based on real-time sensor and historical data, and a list of sensor and user policies are defined by the system administrator. In order to evaluate the framework performance after applying scalability and availability techniques, a load testing environment was built in the cloud to simulate a large number of virtual users. This environment was created in order to examine the quality of the services as provided by Windows Azure. The results have shown that the use of scalability techniques can significantly increase availability and performance. Finally, we present an eHealth smart system as a case study for collecting real-time data that can be used for real-time monitoring and data analysis.

Dedication

To my parents and siblings.
They mean a lot to me!

Acknowledgements

I would like to thank my family members, as without the continuous encouragement of my parents, and constant motivation and loving support of my wife, it would have been difficult to fulfill this work.

I would like to thank and express my sincerest gratitude to my supervisor, Dr. Qusay Mahmoud, for his continuous support and motivation throughout the research work.

I would also like to thank my exam committee members, Dr. Masoud Makrehchi, Dr. Ying Zhu, and Dr. Martin Agelin-Chaab, for their consideration and time in revising my thesis and their helpful recommendations.

I convey special acknowledgement to Abdullah Qasem, a good friend, for his boundless help and encouragement. He is a friend in the truest sense and I wish him the very best in his future endeavours. I would also like to acknowledge another friend, Abdulaziz Almehmadi, for his assistance with the configuration of the Raspberry Pi.

Last but not least, the scholarship and support from Umm Al-Qura University in the Kingdom of Saudi Arabia is greatly acknowledged.

Table of Contents

Acronyms and Abbreviations

WSNs	Wireless Sensor Networks
IoT	Internet of Things
KNN	K-Nearest Neighbour
SQL	Structured Query Language
NoSQL	Not Only SQL
VM	Virtual Machine
DR	Data Repository
DPU	Data Processing Unit
RS	Request Subscriber
DBMS	Database Management System
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
VHDs	Virtual Hard Disks
CDA	Clinical Document Architecture
WSP	Web 2.0 Service Platform

List of Figures

List of Tables

Chapter 1

Introduction

Many applications have used Wireless Sensor Networks (WSNs) in their components. Although these applications are widely implemented and are an important trend in many industrial, governmental, commercial, and environmental applications, WSNs face several issues due to weaknesses in their communication, such as security and privacy, reliability, scalability, and mobility. WSNs also face many challenges with resources, such as limited battery and storage data capacity, availability of bandwidth, and data processing capabilities. Another important factor to be considered is the lack of efficient management of the vast amounts of sensor data. High-performance, powerful, scalable computing, and an unlimited storage infrastructure are required for real-time storage and data management and analysis. Therefore, using cloud computing technology can be an effective solution to overcome the limitations of wireless sensors in terms of storage, memory, scalability, and availability. Cloud computing technology has several advantages, including flexibility, high levels of automation, scalability, availability, the provision of fast service, and unlimited storage capacity.

In designing and implementing a framework, consideration must be given as to whether such a framework can provide scalability and high availability to wireless sensor applications. In addition, this framework should be able to manage and analyze a large amount of sensor data in order to extract interesting patterns.

This thesis presents the design and implementation of a framework that integrates wireless sensors and cloud computing. The proposed framework can be applied to the many applications that use wireless sensors in their infrastructure. The presentation of this framework focuses on building a prototype that provides scalability, high availability, and security, and enhances data management and analysis for wireless sensor applications.

This chapter presents our motivation behind understanding why integrating wireless sensors with the cloud is an effective solution. This chapter is organized as follows: Section 1.1 presents the motivation of this research, while Section 1.2 details the research statements which instigated this study, and Section 1.3 specifies the thesis contributions. Finally, the organization of this thesis is presented in Section 1.4.

1.1. Motivation

Many modern applications, such as smart home monitoring, the smart city, and eHealth systems have used Wireless Sensor Networks (WSNs) in their infrastructures. For example, eHealth smart systems have been used to give healthcare environments more vitality and to provide very fast service for patients. In recent years, however, these sensors have generated a vast amount of data. Consequently, WSNs have faced several challenges and limitations in dealing with the large scale of data in terms of processing, memory, storage, and scalability. In order to solve these challenges and limitations, our motivation is to develop a framework for integrating wireless sensors with cloud computing. This framework can be applied to offer unlimited data storage, high scalability, availability, and security.

Even though cloud computing has many advantages in terms of storage capacity, the ability to easily share information, and access to lightning-fast processing power, cloud computing has faced several challenges, such as security, availability, and performance[1][2][3]. Thus, this study also presents modern techniques that are applied recently to overcome the cloud computing challenges.

Even though many researchers have provided and developed several frameworks for the integration between WSNs and cloud computing, there are several new technologies and techniques that have emerged in terms of storage, scalability, availability, and security. Thus, our objective is to develop a framework that provides a new approach for the applications that use wireless sensors in their infrastructure. This approach focuses on the design and implementation of a framework that uses new features of cloud computing to overcome the lack of the embedded systems and sensors.

1.2. Research Statement

In order to design and implement a framework for integrating wireless sensors with the cloud, existing frameworks will be studied to identify areas of weakness. Following this, a proposed framework will be designed and implemented using modern techniques of cloud computing. Even though this study focuses on providing scalability and high availability to applications that use wireless sensors in their infrastructures, data management, analysis, security, and privacy are taken into consideration in this framework. The framework will be evaluated in terms of scalability and availability. In this thesis, we also study the challenges of integrating wireless sensors and cloud computing. In addition, we address new techniques that have been used to increase the scalability and availability of cloud technology, while ensuring that all sensor data have been securely stored. Moreover, applying the auto scaling technique to the cloud-based system is extremely valuable when the number of users is increased or decreased, as this technique helps to ensure that the proposed framework is reliable and cost efficient. For example, when the system is experiencing high load, the number of virtual servers employed will be automatically increased. Alternatively, when the system is experiencing low load, the number of virtual servers employed will be automatically decreased. The automatic increase or decrease of server usage contributes to a highly reliable and cost efficient platform.

1.3. Thesis Contributions

The main contributions of this thesis are as follows:

- **Integrating wireless sensors with the cloud** - We present a framework for the integration of wireless sensors and cloud computing. This framework provides wireless sensor technology with high scalability and availability, and promotes more security features. Moreover, storing sensor data in the cloud, which provides unlimited storage capability, can be an efficient solution for applying data analytics techniques. A decision making algorithm is one of the most important features of this framework. This algorithm makes proposed decisions based on real-time sensor and historical sensor data, and a list of sensor and user policies are defined by the system administrator. In terms of security and privacy, we present some existing techniques that can be applied to ensure that sensitive data have been securely transferred and stored.
- **A prototype implementation of the framework using Azure** - Azure is a cloud platform that enables developers to build, publish, manage, scale, and test applications. Thus, the proposed framework was implemented using Windows Azure. The framework has been evaluated in terms of scalability and availability using the JMeter tool to generate a high load by simulating a large number of users to examine the capability of the system after applying the scalability and availability techniques.

- **Data security and privacy:** Securing data is one of the challenges that face cloud technology. Thus, we present some techniques that can be applied to ensure that sensitive data has been securely transferred and stored. Although data security and privacy are considered in this framework, these factors are not the focus of this thesis.
- **Designing and implementing an eHealth System:** As proof of concept, we designed and implemented an eHealth system that can be applied to serve the healthcare community. We have published some of the results at an IEEE conference [4].

1.4. Thesis Outline

This thesis is structured as follows: **Chapter 2** presents an overview of WSNs, cloud computing, and the Internet of Things. We then discuss previous studies that used cloud computing technology to solve the challenges that face WSNs, including limited storage, computing, and low performance. **Chapter 3** describes the design and implementation of the proposed framework, while **Chapter 4** documents the proof of concept implementation of the proposed framework. **Chapter 5** presents the design of experiments for evaluating the performance of the proposed framework as well as results of evaluation. The case study (eHealth Smart System) is the focus of **Chapter 6**. Finally, the conclusion and future work are provided in **Chapter 7**.

Chapter 2

Background and Related Work

This chapter presents an overview of the main research areas related to this thesis, namely, cloud computing, wireless sensors, and Internet of Things (IoT). We also review previous studies that have been carried out for integrating wireless sensors and cloud computing.

2.1. Wireless Sensor Networks

Wireless Sensor Networks (WSNs) are widely implemented and becoming an important trend in several industrial, governmental, commercial, and environmental areas. WSN architecture is a self-organizing network that has a number of sensor devices, which connect to other devices by using a wireless communication network. In other words, WSNs are a computer network with a large number of sensor nodes. These sensor nodes interact to monitor a specific area and bring data from surrounding areas. Sensor nodes then transfer sensing data to a master management node [5][6]. A WSN contains sensors that have the ability to cooperatively monitor environmental conditions such as pressure, pollution, temperature, and sound. In recent years, WSNs have also been used in many areas such as healthcare, military targets [7][8], environmental monitoring (such as natural disasters [9]), seismic sensing, dangerous environment exploration [10], and sound. However, WSNs face several challenges and issues due to their lack of communication in areas such as security and privacy, short communication range, reliability, scalability, and mobility. Sensor networks also face many challenges with resources such as limited battery and storage data capacity, availability of bandwidth, and capability to process data [11].

2.2. Cloud Computing

Cloud computing, the future generation's computing model as a utility, has the ability to make cloud-based software more attractive as a service. Developers who have innovative ideas for enhancing Internet services no longer need to set up a large number of physical machines, or require experts and technical support to operate the infrastructures. Cloud computing technology has several advantages such as flexibility, automation, low cost, fast service, and unlimited storage capacity. Cloud computing technology provides users with great flexibility [12][13]. For example, users are able to view their data and modify them from anywhere and at any time by using different kinds of devices (such as computers, smart phones, and laptops) and different kinds of operating system (such as Windows, Mac, and Android). One of the popular cloud computing definitions is that published by the National Institute of Standards and Technology (NIST). *"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models"* [14].

2.2.1. Cloud Delivery and Deployments Models

NIST introduces three cloud delivery models which are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). All three models can be combined to build an architecture of IT resources. Thus, the PaaS environment provides the cloud consumers with a ready-made environment to develop and deploy their applications as SaaS cloud services. Many cloud providers offer PaaS such as Heroku [15] and Google App Engine[16]. Examples of cloud providers that offer IaaS are Amazon Web Services (AWS) [17] and Windows Azure [18].

NIST also presents four cloud deployment models, namely, public cloud, private cloud, community cloud, and hybrid cloud. A public cloud generally offers IT resources and cloud services, and these services are provided by a third party. A private cloud offers IT resources that are used by only one organization. The purpose of the private cloud is to ensure that an organization is isolated from others. A community cloud is generally limited for access by a group of organizations that have common computing concerns. A hybrid cloud is a combination of two or more other cloud deployment models [13].

2.2.2. Scalability in Cloud Computing

One of the main advantages of using cloud computing is that it addresses and supports many scalability issues. When managing a successful application, the ability of the system to scale becomes a critical need. It is important to be able to fit rapid growth in incoming traffic and a large amount of data. This helps to ensure customers, who are using the system, are satisfied. Thus, the most obvious goal of an elastic system is to have the ability to automatically scale-out to meet growing demand. However, the objective is not only to scale out to handle increasing loads, but also to scale in during lower volume periods or reduce the number of incoming requests [19].

Scaling has been applied to many systems to increase computing resources. Two different types of scaling are Vertical Scaling and Horizontal Scaling. The definition of Vertical Scaling, is growing a single machine with more and faster CPUs, more memory, or faster disks. Horizontal Scaling, can be applied by adding additional servers to support the overall application’s computing requirements, and the load balancing technique that has been applied across those resources.

The main role of the load balancer is to distribute incoming requests from clients across available servers. Many cloud providers offer different methods of load balancing. The cloud user can choose which method best fits the application requirements.

How scalability and reliability are interlinked is another important factor that should be considered when the architecture of the system has been designed. Since VMs are added to the system’s infrastructure, new endpoints that are added to the system can fail for any reason. Therefore, all components should be configured in a redundant manner, to ensure 24x7 always-on operations.

Scalability and availability are key factors that make cloud computing more attractive to organizations and individual users. Developers have full control for adding and removing instances as they need. Thus, if one of the instances fails, the load balancer reroutes the workload to the remaining running instances. Load Balancing has many features that include distribution of requests across multiple instances, and continuous checking of the healthy instances which have been registered with Load Balancing. The most popular load balancer technique is Round Robin, which distributes incoming requests one at a time to an application server for processing, allowing requests to be serviced in a certain period of time [20][21].

2.3. Security and Privacy in the Cloud

Security and privacy are the most significant challenges in the cloud computing environment. Cloud computing is a multiple range environment in which numerous computing resources are shared. These shared resources, both hardware and storage data areas, are at risk from insider or outsider attacks. Cloud computing faces several challenges in terms of security and privacy [2][22]. Researchers have discussed various approaches to address these challenges, and they present their solutions for building a more secure cloud environment. Many methods have been introduced to ensure data security and privacy in the cloud, including creating secure channels to transfer data from local servers to cloud servers, using cryptographic processes to encrypt data before storage, and applying authentication and authorization processes before storing or retrieving sensitive data [23].

2.4. Internet of Things

Widespread sensing enabled by WSNs has been used in many areas of the live environment. As a result, this offers a high ability to sense and understand environmental indicators. The fast growth of these devices and sensors in a communicating network has brought into being the Internet of Things (IoT). Actuators and sensors combine with the environment, and the information is shared across multiple platforms. In the IoT model, a large number of the objects/things that are around us are going to be on the network in one form or another. Smart connectivity with new modern networks and context-aware computing are important components of IoT [24][25].

The Internet of Things demands the following: (1) Users understand how they can use their appliances; (2) Pervasive communication networks and software architectures process the contextual information to where it is relevant; (3) The use of analytics tools to discover interesting patterns and smart behavior from collected data. When these three fundamental processes have been applied, context-aware computation and smart connectivity can be accomplished. Ashton in 1999 presented the term Internet of Things [26]. However, the definition of IoT has been more widely used in different kinds of applications such as transportation and healthcare [2]. Weiser, the father of ubiquitous computing, presents the definition of the smart environment as *“ the physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network”* [27].

2.5. Related Work

This section presents some related work which focuses on the design and implementation of a framework for integrating WSN and cloud computing. Some of the work presents techniques that can apply to increase the security and privacy in the cloud.

Fortino et al. [28] present a system designed for the monitoring and management of body sensor data, called Body Cloud, using cloud computing technology. The authors’ approach relies on the integration of cloud computing and WSNs which can provide scalable, powerful data storage, and the enhancement of processing infrastructure to execute the analysis of body sensor data. This study presents the monitoring and management of sensor infrastructure, especially data flows, based on cloud computing infrastructure. One of the important applications is monitoring a patient condition using medical distribution sensors. A network of body sensors generates a vast amount of data that should be processed and stored. Cloud computing technology can solve the limitations of WSNs in terms of computing, storage, and scalability. Body Cloud architecture covers some techniques that are related to data management. However, the security perspective for storing the sensor data in encrypted format is not discussed in their approach.

Rao et al. [29] describe how cloud computing and the IoT work together to overcome the challenges of Big Data. This study discusses sensing service technology in the cloud computing environment when using some applications in contexts such as environmental monitoring and agriculture. This work presents an architecture model in the cloud, and the main advantage of which is providing sensing as a service in the cloud. WSNs enables services and applications to react with the real physical environment. Cloud computing services, includes Big Data analytics and high computing resources, can be used to store, process, and analyse the sensor data to improve availability and scalability.

Perumal et al. [30] presents a study of the integration between WSNs and cloud computing for e-healthcare applications. In this application, sensor data are captured and will be sent to the cloud environment. A variety of end users, such as researchers, clinics, and patients can have full access to their data in the cloud. This study presents an architecture model that can provide a cost efficient solution for life care agencies and automating hospitals, allowing then to manage real-time data from different types of sensors, provide strong authentication mechanisms, and support privacy. However, there are some significant points that should be taken into consideration in future studies such as using a strong encryption mechanism for storing data in the cloud.

Sudarsan et al. [31] introduce a new approach for life care communities, which focuses on a secured Wireless Sensor Networks integrated cloud computing for life care. The system, which has been designed to monitor human health, has many features that allow it to share data in the cloud among different types of users, including doctors, care-givers, pharmacies, and clinics. Thus, at low cost, the users can obtain a high service of healthcare. This study uses different technologies with new approaches, including cloud computing security, and WSN.

Chung et al. [32] present a new approach for integrating a WSN with cloud computing to monitor the information provided by an agriculture system. The authors propose a solution to monitor a physical environment and to gather important sensor data, such as humidity and temperature. The authors suggest that applying this approach will provide convenient and quicker information for those who use the system, based on WSNs. However, this paper does not mention any technique that is applied to deal with a vast amount of sensor data. In addition, in their approach, the authors do not design their architecture model based on a security perspective. Therefore, the design of this architecture is weak because it does not consider the most important factor, which is the security perspective.

Lounis et al. [33] offer a design and implementation architecture for gathering and accessing large amounts of data generated by WSNs. The main objective of this architecture is to overcome the challenges of dealing with a vast amount of data and to facilitate information sharing among healthcare professionals. The paper focuses on data management in WSNs, specifically sensor data that have been generated from medical sensors.

Khandakar et al. [34] present a framework for integration between WSN and cloud computing and they discuss the security and privacy challenges of cloud computing. The focus of their research was on access control, user management, retrieval of distributed data, and storage. The framework has many components, including Identity and Access Management Unit (IAMU), Data Processing Unit (DPU), Request Subscriber (RS), Pub/Sub Broker, and Data Repository (DR). While they use an encryption message technique in the cloud, they do not mention which cryptography algorithm is applied for the encryption mechanism. In addition, they do not discuss which type of security key mechanism is used. In particularly, whether there is a specific technique to generate a security key for each user or if a static key is used. All these details have a significant impact from a security perspective.

Lan [35] presents a novel framework, Sensor Cloud, which enables the consumer to effortlessly capture, process, access, store, analyze, share, visualize, and seek a large amount of sensor data from many types of applications, using all features of the cloud, such as storage resources and computational IT.

Chang et al. [36] have presented ubiquitous cloud computing middleware for smart home automation. In their approach, the cloud computing platform enables end users to monitor and control various kinds of sensors and devices in the home at any-time and from anywhere. A significantly large amount of data will be processed in the cloud computing environment which has context-aware intelligence processing to control the remote devices.

Rolim et al. [37] focused on developing patients' data gathering technique. This paper presents a novel framework to solve the problems of taking notes manually which is a slow process. Besides, they cause lateness for accessing real-time data and that restricts the capability of clinical monitoring and diagnostics. Thus, authors proposed a system to automate collecting patients' information process using wireless sensor networks which are connected to medical equipment, and then transferring this data to the healthcare provider centers in the cloud to store, process, and analyze patients' data.

2.5.1. Scalability and Availability

One of the most significant features of cloud computing is providing availability. Thus, applying the load balancing technique in the cloud computing environment is an important factor for improving availability and performance. The Load balancing technique has been explained in a white paper as demonstrated by Adler [38]. The paper presents the techniques and tools that have been commonly used for applying load balancing in the cloud computing environment. However, the author states that the load balancing technique still has some challenges in adapting to the many changes in the cloud. Chaczko et al. [21] illustrates the role that load balancing plays in improving availability and performance. For applying load balancing, there are many different types of load balancing algorithms such as Ant Colony and Round Robin.

2.5.2. Data Management and Analysis

Lijun et al. [39] have applied Hadoop technology to develop Health Information Exchange (HIE), which is the next generation of health informatics development in China. Their approach employs a medical information platform that relies on Hadoop, called Medoop. Medoop uses HDFS to efficiently store Clinical Document Architecture (CDA). In addition, they utilize the MapReduce paradigm to organize the attributes of data on the CDA document, based on the frequent business and computed statistic distribution. Their objective in applying the Medoop platform is to build an integrated platform for health information storage by using the components in Hadoop. Utilizing Hadoop to store CDA documents increases the efficiency of storing clinical data.

2.5.3. Context Awareness

Huang [40] presents a new design of the Web 2.0 Service Platform (WSP) for DPWS based smart home sensors and devices in cloud computing environments. This platform automatically records and observes a user's behaviours to adapt home appliances to reset the devices based on the changing need of the residents. Users can obtain more elective mobility and scalability by implementing ontology technology. A mobile user can control and monitor home devices through web 2.0 blog-based interfaces.

Lu et al. [41] propose a novel indoor localization method deploying Wi-Fi access points, called the cluster K-NN algorithm. Furthermore, using this technique does not cost extra money for infrastructures and still provides decent accuracy compared to other localization techniques. According to the offline simulation, the complexity of the cluster k-NN is lower, while the accuracy is up to 98.67 percent.

2.5.4. Security and Privacy in the Cloud

Hwang et al. [42] approach an interesting business model for cloud computing, this model based on the concept of performing the encryption and decryption technique, whereby a cloud provider must ensure that the data have been stored in encrypted format. Moreover, after the computation operations are completed, all data must be deleted. This study discusses many points related to encryption data in the cloud. However, it does not discuss security aspect during uploading and downloading operations in terms of securing the channel between the client and cloud provider.

Lam et al. [43] proposes Tresorium which is a cryptographic file system that is designed for data storage in the cloud. The authors state that the data must be encrypted before uploading to the cloud storage providers. They mention that Tresorium enables the sharing of data among a number of clients using an underlying group key agreement protocol. However, in their solution, the data encryption process is not completed in the cloud. Therefore, users need to have specific software to encrypt their data before the uploading operation.

2.6. Summary

This chapter presented an overview of cloud computing, WSN, and IoT areas. Some previous studies present a framework of integrating WSN and cloud computing while other related work was describe some encryption techniques have been applied in order to increase the security and privacy in the cloud. The proposed framework will be presenting in the next chapter.

Chapter 3

Proposed Framework

This chapter explains the main components of our proposed framework. The most important features of this framework are providing high availability and scalability, and more security. Data management and analysis have been incorporated into this framework. Section 3.1 describes an overview of the proposed framework. The illustration of framework components is presented in Section 3.2, while the description of how sensor data have been collected is offered in Section 3.3. The scalability and availability techniques are presented in Section 3.4; while data management and data analysis is discussed in Section 3.5. Finally, Section 3.6 presents the data security and privacy techniques that apply in the proposed framework.

3.1. Overview

As shown in Figure 3.1, the proposed solution serves different types of applications such as eHealth, smart home, smart city, and other applications that need to use Wireless Sensor Networks (WSNs) in their infrastructure. These sensors are connected to a local server which is responsible for collecting data from sensors and for transmitting the data through wireless communication channels to the appropriate services hosted in the cloud. The system, which is located in the cloud side, has many functions as listed below:

(1) It is responsible for providing a service to store sensor data in an encrypted format;

(2) It provides a platform for analyzing, managing, updating, visualizing real-time data;

(3) The system is able to make appropriate decisions based on real time data and historical data; and

(4) Some security techniques have been applied to ensure that sensitive data is transferred from clients to the cloud server securely, and is stored in encrypted format using advance cryptography algorithm.

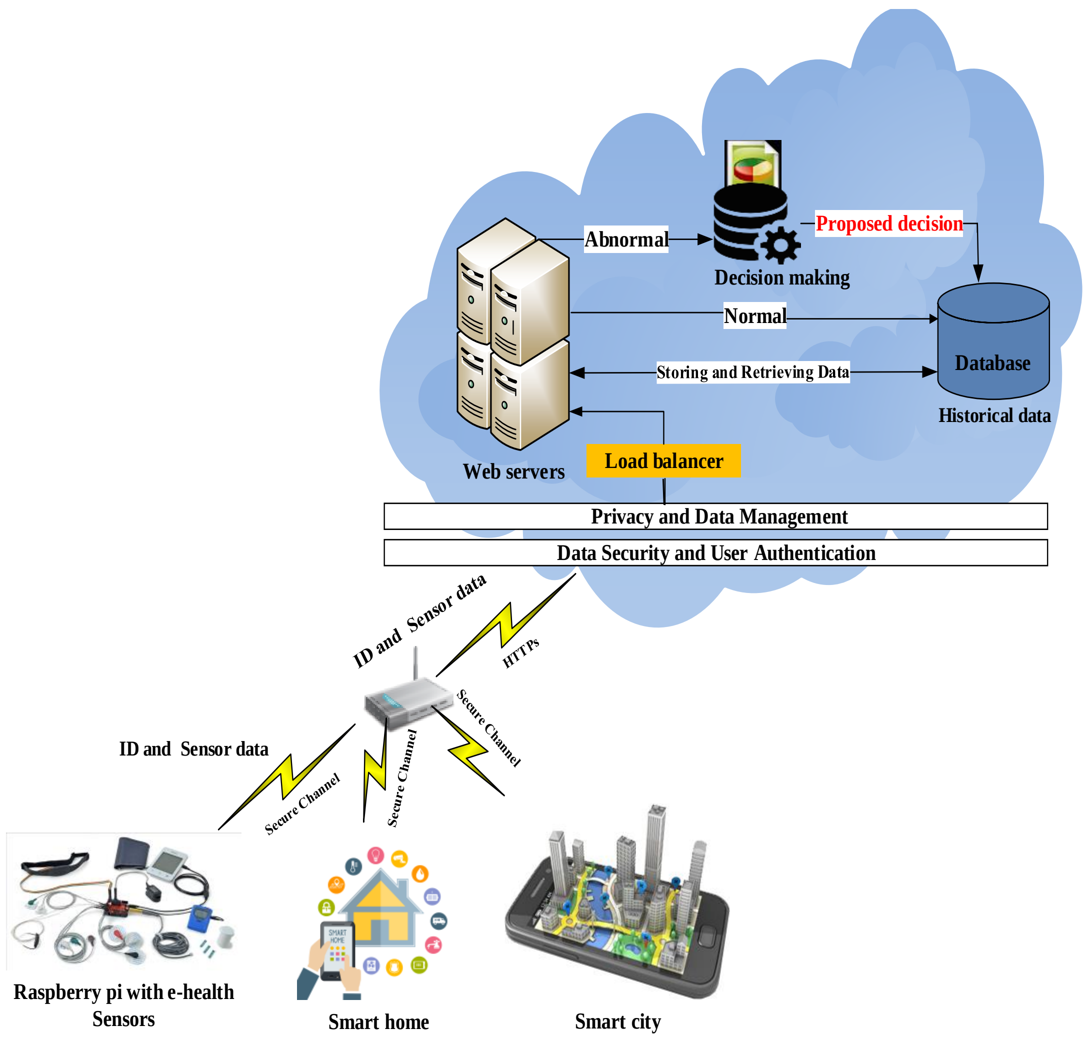


Figure 3.1: The Proposed Framework

3.2. Framework Components and Design Goals

A number of points should be followed in order to explain the concepts and techniques that are designed to build this framework:

3.3. Data Collection

Wireless sensors are responsible for measuring and monitoring environmental conditions. These sensors are often connected to a local server or microcomputer. For example, these sensors are connected to the Raspberry Pi that reads the data from a smart home sensor device and securely transfers this data (device id and sensor data) over the wireless network channel to the cloud. A cloud-based system can be connected to one Raspberry Pi or more, based on the application requirements. An application has been deployed to the cloud to receive and store sensor data in database. Sensor data will be stored in a high scalable available database that is running in the cloud.

3.4. Scalability and Availability

Building a scalable and available framework in the cloud is one of our objectives. Thus, we architect the proposed framework has been designed to ensure high scalability and availability using some cloud services that have been created to help developers build high scalable, available applications running in the cloud environment. The proposed framework can serve a large number of users after applying the scalability techniques. Horizontal scaling has been selected as this type of scaling allows the system to scale out and scale in without down time. Maximizing scalability is a significant aspect that can be applied by increasing the number of virtual machines that can serve a large number of users, and scaling a system based on the business requirements by adding or removing resources from the system's components. In addition, some cloud providers offer some features such as caching and load balancer services that are specifically offered to enhance the scalability of the cloud application.

One of the best features of cloud computing is offering auto-scaling service. Auto-scaling helps cloud users apply some available metrics based on their preference, such as CPU average. For instance, cloud users can set metrics such as adding three virtual machines when the CPU average is higher than 50% for all running VMs.

When we launch a single VM, we are launching a VM running on a physical server at one of data centers. As a result, one of the following events could take down the VM:

- The VM itself could fail, or the hard drive volume could become corrupted;
- The physical server on which the VM resides could fail;
- The data center, which houses the physical machine could fail.

Running the system in one single virtual machine is not an optimal solution because it can fail for any reason. Thus, the proposed framework has used several techniques that are offered by cloud providers to maximize the availability of the system, such as running a system in different geo-locations to ensure that the system is always available. Thus, hosting copies of a web-based system in multiple data centers around the world increases the availability of the system. The data center is called an Availability Zone (AZ). A whole cluster of AZ's is known as a Region. The proposed framework is designed to deploy the system to different AZs for high availability. When a single VM or an entire AZ goes offline, traffic stops routing to the affected VM. This is the idea behind a Multi-AZ setup, and which will covered in more detail later in Chapter 4.

Since we provide our framework to serve multiple numbers of users, we need to apply load balancing technology to our proposed solution to distribute expected incoming application workload across multiple web servers. This will help to achieve maximum levels of fault tolerance in the application and provide the desired amount of load balancing capability, which is required to distribute application workload. Furthermore, load balancing is responsible for ensuring that only healthy web servers receive requests by detecting unhealthy web servers and changing the new request path towards the remaining healthy web servers.

3.5. Data Management and Analysis

Once wireless sensor applications are integrated to cloud, unlimited storage will be available. In addition, many cloud providers offer different ways to store and manage large amounts of data. Many applications today rely on a Relational database using Database Management System (DBMS) to store receiving data. Relational databases such as Microsoft SQL Server. Managing and analyzing sensor data is an important approach particularly when sensors generate a large amount of data. As a result, using advance analysis tools that can help to extract interesting patterns from a large amount of data is required. In the proposed framework, a system is able to make decisions based on real-time sensor data and historical data. In the next section, we will explain the decision making process.

3.5.1. Decision Making Algorithm

The system is responsible for creating appropriate decisions using a decision-making algorithm based on standard medical practices and historical data. The proposed decisions will be sent to system administrators, who are responsible for making a final decision for approval. Therefore, users will have high quality services because the system provides real-time data gathering, eliminating manual data collection, enabling the monitoring of the large number of devices, and creating appropriate decisions that can help users to create final decisions.

When the cloud application receives sensor data from the client application, the cloud application will check if the sensor data is normal or abnormal, depending on the policies defined in the system. Every sensor and user has a list of policies are defined by the system administrator. For example, in the Smart eHealth System, some patients have chronic diseases, so medical staff must consider some changes in patient heartbeat rates, decreases in Oxygen saturations and increase in body temperature. Thus, the system will make decisions based on these abnormal data and other criteria. Another concern in smart home systems is applying policies technique. This provides the necessary options to create a dynamic policy by using active devices and their attributes. The following examples can be applied in smart home applications to set policies. These examples explain how machine to machine can take place based on policies that have been created by the system administrator.

If Alarm System = "On" and HomeTem > "40.5" = "Set Status is Abnormal" and Making Decision & (send notification to the Emergency Services).

If the data is normal, the application will store this data in the database to feed the historical data. Otherwise, the application will make a decision based on historical medical data and polices. If the user does not have any historical data for the same condition/s, the system will make a decision based on the historical statistical data of users who have similar conditions. The decision making algorithm has been presented in detail in chapter 6.

3.6. Data Security and Privacy

One primary requirement of our study is to secure communication between sensors and the cloud service, and sensitive data must be stored in encrypted format. There are a few different attack vectors that could be utilized by adversaries and those security concerns need to be taken into consideration when designing our framework. Even though it is quite a challenge to address all possible data security concerns, it is important to discuss the possible threat models which need to be considered in such an implementation. For example, in some cases, home owners are able to remotely control intelligent appliances such as furnaces, washing machines or even a coffee machine with the use of a smart phone. While these types of advancements provide huge comfort to our hectic lives, they also pose a certain danger. For example, if any enemy is able to remotely control your furnace or enter into your home, this poses a threat to your loved ones as well as to your home. Therefore, it is important to provide proper security measures to mitigate these types of risks.

3.6.1. Encryption Mechanism

Data, by default is presented in a readable format called plaintext. As a result, when sensitive data is transmitted over a network as plaintext, it is vulnerable to unauthorized and potentially malicious access. To protect sensitive data from malicious users, applying a strong encrypted algorithm is required. In the proposed framework, we apply two security techniques, encryption and hashing, to ensure that the framework provides more security to these applications that use wireless sensors in their infrastructure. We used an encryption mechanism to secure the channel between clients and cloud server. Also, storing sensitive data in the cloud using, an encryption key will be implemented [44]. More details on the encryption mechanism is mentioned in Chapter 4.

3.6.2. Hashing Mechanism

The hashing mechanism is used when a one way form of data protection is required. When hashing mechanism has been applied to a message, it is locked. We used hashing to store users passwords in cloud storage.

3.7. Summary

In this chapter, we presented the proposed solution which can be applied to implement a high available and scalable framework for the integrating between WSN and cloud. In addition, the framework has been designed to provide scalability and high availability. This framework also is able to make proposed decisions based on the sensor and historical data. Moreover, some existing encryption techniques have been applied to this framework in order to ensure that sensitive sensor data are securely store in the cloud. The implementation of the proposed framework will be presenting in the next chapter.

Chapter 4

Prototype Implementation

This chapter presents the design and implementation of the proposed framework. Section 4.1 describes an overview of implementation architecture for integrating wireless sensor and cloud computing, while Section 4.2 identifies the components, tools, and programming languages that have been applied to implement this framework. Scalability and availability techniques that were implemented to build a high scalable and available framework are described in Section 4.3. Finally, the security and privacy techniques that have been applied to increase the quality of services are outlined in Section 4.4.

4.1. Implementation Architecture

The design and implementation architecture of the integration between wireless sensors and cloud computing is presented in this section. Many modern technologies have been applied to develop this integration, especially in scalability, availability, and security level.

4.1.1. Overview

As shown in Figure 4.1, Raspberry Pi has been used to collect data from different types of sensors, including a body temperature sensor and a Pulse and Oxygen sensor. These data are transmitted to the cloud environment through secure wireless communication channels. A web-based system has been built in the cloud to receive, manage, and store sensor data. Some different techniques have been applied to ensure that the proposed framework provides more security. The system also uses a load balancer technique that plays an important role in terms of availability, by handling incoming user requests across multiple Virtual Machines (VMs).

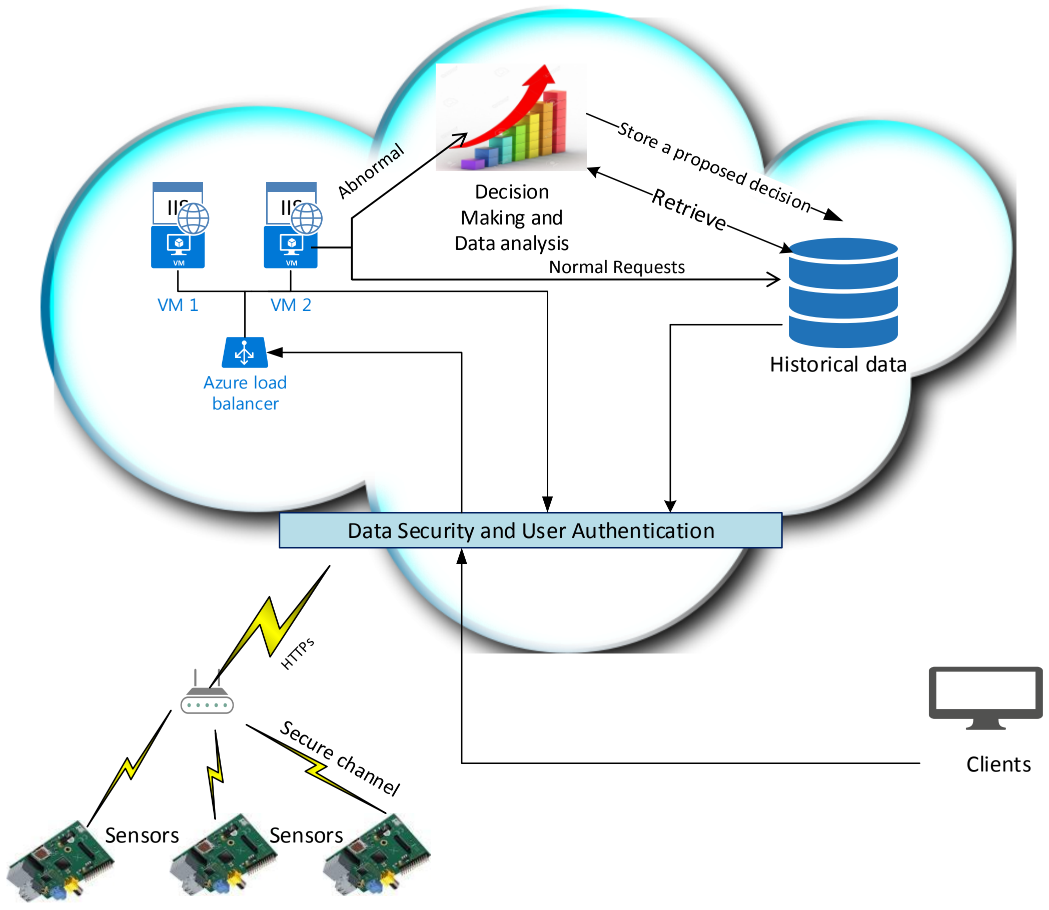


Figure 4.1: System Architecture of Prototype Implementation

After the data has been received, the system is responsible for making appropriate decisions using a decision-making algorithm. The algorithm makes the decision based on three parameters, namely, real-time sensor data, historical users' data, and system polices that are defined by system administrators. Before the algorithm begins the decision- making process, the algorithm checks if the data needs to be analyzed, or simply stored. In other words, the decision-making algorithm will be applied if the real-time data is abnormal. Otherwise, the data will be stored for future needs. The proposed decisions will be sent to individuals, who are responsible for making final decisions. The system sends final decisions to clients after the proposed decisions are approved. Therefore, the quality of service offered to users will be enhanced because the smart system supports decision makers and users by providing real-time data gathering, eliminating manual data collection, and enabling the monitoring of a vast number of sensors.

4.2. Components Implementation

This section discusses the components of the architecture and technologies that have been applied to implement this framework. New and different techniques and components are combined to ensure that the proposed framework is providing high scalability, availability, and security. The components of the architecture from the client's side, are Raspberry Pi and the sensors. In the cloud side, we use some available cloud services to increase the system scalability and availability, such as Load Balancer.

4.2.1. Client Platform

The client platform can be divided into two significant components, the Raspberry Pi and the sensors. The Raspberry Pi is a small size computer developed in the United Kingdom by the Raspberry Pi Foundation. Raspberry Pi connects to a computer monitor or TV, and uses a keyboard and mouse. The Pi's hardware includes 2 USB ports, an HDMI and Ethernet port, an SD card slot, memory, video/audio outputs, and a power source [45]. The Pi runs a Linux operating system and recognizes Python as the main programming language with support for C, Java and Perl. Raspberry Pi components are shown in Figure 4.2. We used the Linux operating system and ran a client application written in C++ on the Raspberry Pi, to read the data from the sensors and to send collected data to the cloud. A client application was created using a socket programming technique and TCP/IP protocol to transfer sensor data from the client side to the cloud. Raspberry Pi and the sensors have limitations in terms of computing, memory, storage, and power. As a result, connecting Pis' to the cloud can be one of the best solutions to overcome the challenges and limitations presented by the Pi devices themselves.

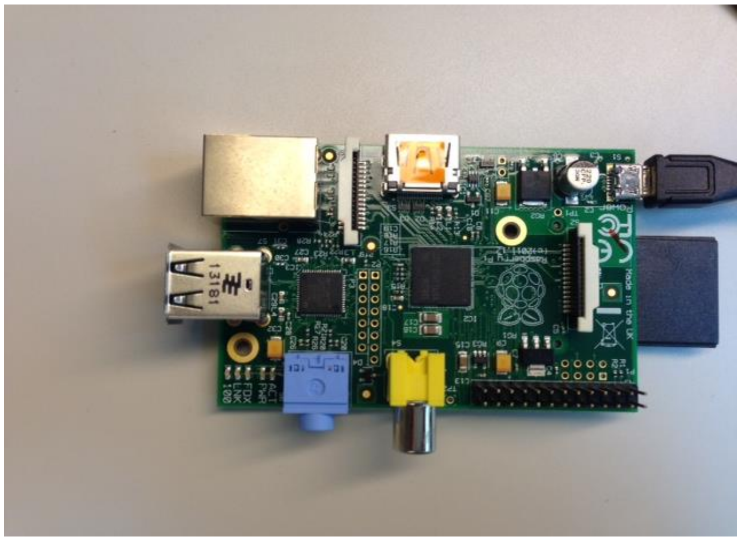


Figure 4.2: Raspberry Pi

Developers can connect different types of sensors to the Raspberry Pi. The types of sensors will be selected based on application requirements. In the case study of the design and implementaion of eHealth Smart System, commercial Medical sensors were used to implement client side connecting to these sensors [4]. The case study of eHealth Smart System is discussed in detail in Chapter 6.

In the Raspberry Pi, the client application has been written in C++ programming languages using Socket and TCP/IP protocol. This application is responsible for collecting data from sensors and sending them to the cloud. On the cloud side, the server application has been written in C# programming language. The server application is responsible for receiving sensor data and storing the data in the cloud storage. At this time, the data analysis and decision-making algorithm will be applied in the cloud. The client application code is added in Appendix A while the cloud server program code is added in Appendix B.

4.2.2. Cloud Platform

On the cloud side, the Windows Azure Platform has been selected as the cloud provider. Windows Azure offers many cloud services and components, including Virtual Machine, load balancer, traffic manager, high scalability and availability services, machine learning and data analysis tool, big data solution, load testing, and security and privacy.

After we searched for an appropriate cloud provider that can fit the need to implement the proposed framework, we chose Windows Azure [46] because it provides Azure tools for Microsoft Visual Studio, which can help developers create and publish scalable web applications and services on Azure. In addition, Azure is a cloud platform that enables developers to build, publish, mange, scale, and test applications. Azure is extremely flexible as it supports many different operating systems such as Linux, and many programming languages such as C#, Python, and PHP. One of the advantages of the Azure is that it allows companies to use elastic auto-scale to fit changing resources demand and volume needs. Azure offers SaaS, PaaS, and IaaS. Figure 4.1 shows the design of the prototype implementation on Windows Azure.

4.2.3. Decision Making Algorithm Implementation

We implemented the decision making algorithm using SQL Server Stored Procedure. We can define stored procedure as a group of SQL statements which are grouped to perform a certain task. In Chapter 6, the decision-making algorithm for an eHealth application will be explained in detail. The decision-making algorithm code has been added in Appendix C.

4.2.4. Microsoft Azure Virtual Machines

The Azure Virtual Machine provides users with full control. Customers can use Microsoft Azure Virtual Machines when they need high scalable cloud servers running a Linux or Windows operating system. Customers can take advantage of using Azure as Infrastructure as a service (IaaS) by having full control on VMs in term of stopping and restarting VMs with no loss of data or server setting.

Azure offers a gallery of virtual machine images called Virtual Hard Disks (VHDs). The two types of Virtual Machine images that were selected from Microsoft gallery options were Windows server 2012 R2 and Microsoft SQL server. Multiple Azure VMs were created running Windows server 2012 R2 as the operating system. After all VMs had been created successfully, Internet Information Services (IIS) server, which is used for hosting web sites, was configured. The sizes and description of these VMs are shown in Table 4.1. Although Azure provides different sizes and options of the VMs based on computing resources, we chose the small size because we were interested in minimizing the cost of using cloud services in the implementation. Azure offers VMs services in two tiers, basic and standard. The two tiers offer a choice of sizes. However, some capabilities are not available in the basic tier, such as auto-scaling and load-balancing [47].

Table 4.1: The Size of VM Running in Azure

Size	CPU Cores	Memory	Disk Size
A2\Standard Tier	2	3. 5 GB	135 GB

4.2.5. Azure Cloud Services

The Azure Cloud Service is an example of Platform as a Service (PaaS) which enables users to deploy their web applications to the cloud. However, with PaaS, users do not have administrative access to the VM. In our implementation, we used IaaS, where the Azure Cloud Service is used as a collection of VMs that are hosted in the cloud. Each VM has an image of the system installed on it.

4.2.6. Availability Set

One of the features of the proposed framework is providing high availability. As a result, an availability set is specified when creating a VM in the Azure Cloud Service, and two or more VMs are placed in an availability set. This technique provides several advantages which includes providing high available hardware through a 99.95% SLA guarantee for uptime from Azure. For example, if the application is running on two different availability sets, such as the United States and Europe, the cloud users will have high availability. Where one availability set becomes unavailable, the other availability set will automatically be employed and become accessible.

4.2.7. Virtual Machines Scalability

Once the VMs have been created in the same Cloud Service and they are assigned to the same availability set, scalability and auto-scale can be applied to a number of the VMs based on application requirements. Applying this technique will increase performance and throughput for cloud-based systems. To apply the scaling technique to the applications, all VMs that have been created must be added to the same availability set, and all VMs within that availability set must be of the same size, in order to apply the auto-scale technique.

Applying the auto-scale technique helps users minimize the operational cost of using Azure. Customers can specify auto-scaling based on two different types of metrics, namely, the average percentage of CPU usage, and the number of messages in a queue.

In this case study, we configured the cloud services to automatically scale up and scale down based on average CPU usage. We set the range of average CPU from 60 percent to 80 percent. When the average percentage of CPU is higher than the maximum setting, two VMs are turned on. In contrast, when the average percentage of CPU is lower than the minimum setting, VMs are turned off. We set a ten minute interval between the last scale action, which can be a scale-up action or a scale down action, and the next scale-up action. All VMs were included when calculating the average CPU usage. The average was calculated based on use over the last hour.

4.3. Applying Scalability and Availability Techniques

To provide scalability and high availability in the cloud-based system, a load balancing technique is applied, which is responsible for distributing the workload across multiple VMs. In order to implement this scalable and high available framework, two cloud services were created in Windows Azure. In each cloud service, different availability zones have been created. Two Virtual Machines have been created in each availability zone because one of the system's requirements is to ensure that, where failure occurs in all VMs in a specific availability zone, other VMs in a different availability zone can still process incoming requests, as shown in Figure 4.3.

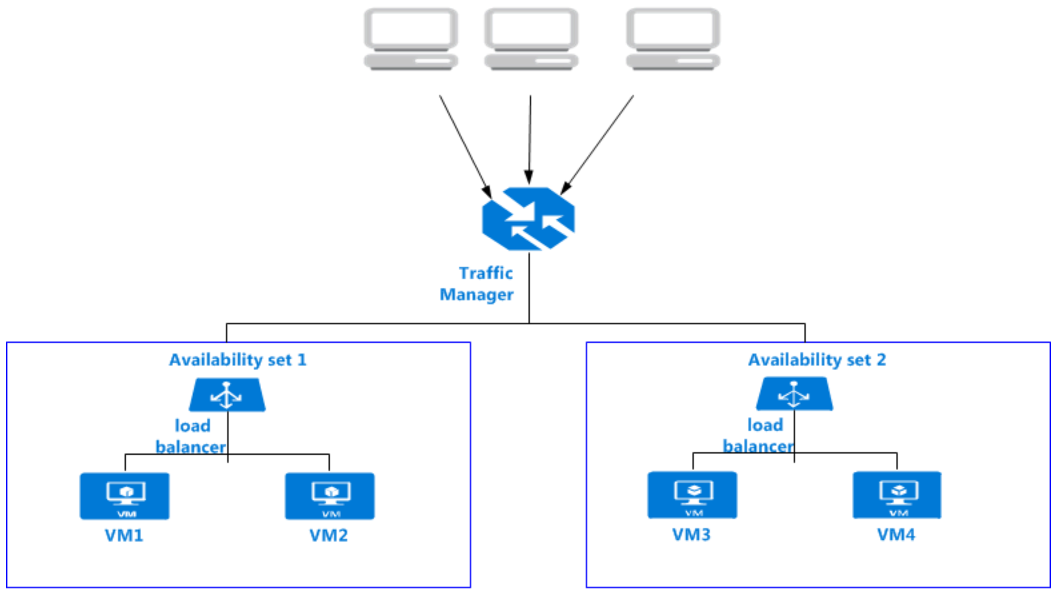


Figure 4.3: Load Balancer and Traffic Manger Implementation in Azure

Windows Azure provides two levels of load balancing for infrastructure as a service. These two levels are Network Level and Domain Name System (DNS) Level. At the Network Level, incoming internet traffic is distributing across VMs located in the same cloud service and data center. The Azure load balancer is responsible for applying load balancing at the Network Level. The DNS level is distributing incoming traffic to different cloud services located in different data centers around the world, or to external endpoints. The Azure Traffic Manager is responsible for applying the load balancing technique at the DNS level. The following section provides more details about the Azure load balancer and the Traffic Manager.

4.3.1. Azure Load Balancer

The Azure load balancer only works with VMs that are in the same region. The Azure load balancer can only apply a Round Robin technique, and routes traffic between two more private endpoints that sit behind a public endpoint. As a result, the load balancer is configured to receive incoming client requests and distribute them among healthy VMs. Since the system is web-based, the load balancer is configured to listen in on port 80 using http protocols.

4.3.2. Azure Traffic Manger

The Azure Traffic Manager provides more availability, and enhances the efficiency and performance of the system. The Traffic Manager allows developers and administrators to control incoming requests of user traffic across available health endpoints, which can include web sites and cloud services. The Traffic Manager uses an intelligent policy engine to route clients, via the DNS, to the cloud provider. As a result, the Traffic Manger supports applications by allowing them to run globally in different data centers around the world [48].

The Traffic Manager has been used for many reasons:

- It is very effective in terms of improving the availability of applications by managing and monitoring endpoints of cloud services. The Traffic Manager provides automatic failover capabilities when an Azure website or cloud services go down.
- It can improve the performance of applications and response delivery times by directing clients to the endpoint with the lowest network latency.
- It enables developers to upgrade and perform service maintenance without downtime.

- It can be used for large, complex deployments. With nested Traffic Manager profiles, a Traffic Manager profile can have another Traffic Manager profile as an endpoint.

Since the object of this study is to design a framework that provides high availability, the Traffic Manager is added to our architecture as shown in Figure 4.3. The Traffic Manager offers different load balancing techniques, which are the Performance Technique, the Round Robin Technique, and the Failover Technique. Developers are responsible for selecting which load balancing method best fits the application's requirements. Developers are able to change the load balancing method at any time. The comparison among these algorithms is shown in Table 4.2. The Traffic Manager works at the DNS level, routing traffic between one or more public endpoints that sit behind a common DNS name [49].

Table 4.2: Traffic Manger Methods

Performance	Round Robin	Failover
Developers can select the performance option when an application has endpoints in different geographic regions and they need incoming request clients to be transferred to the closest endpoint in terms of the lowest latency.	Distribute load across a set of endpoints across different datacenters.	Developers can select the Failover option when they have endpoints in the same or different regions and they want to use a primary endpoint for all incoming requests; however; it provides backups in case the primary Virtual Machine or endpoints are unavailable.

4.4. Security and Privacy Implementation

Windows Azure has been applied to many techniques related to security and privacy, particularly at the network level. Thus, we implemented two important techniques to ensure that sensitive data will be stored securely in the cloud. We used the advance cryptography algorithm to store sensitive data in the cloud in an encrypted format, and stored user passwords in an SQL database that runs in the cloud, using the hashing technique. The detailed implementations of these two techniques are described below.

4.4.1. Data Encryption in the Cloud

Before storing the data in the cloud, our system is responsible for encrypting the data using a cryptography algorithm. In this way, the data is stored securely in the cloud. Different types of encryption algorithms have been used to provide cloud users with data security. The objective of these algorithms is to protect the system against malicious users, and to secure information against advanced threats. The code that has been used to implement the encryption technique is added in Appendix E.

4.4.2. Hash Functions in the Cloud

It is extremely crucial that user data authentication is stored securely. Thus, we used the hashing technique to store user passwords in the database. The code that has been used to implement the hash technique has been included in Appendix D.

4.5. Summary

This chapter presented the design and implementation of integrating wireless sensors with the cloud. Tools and techniques were employed to provide high scalability and availability and to increase the level of security. The Raspberry Pi and sensors were used on the client side to generate sensor data. The VM, Load Balancer, and Traffic Manager were used on the cloud side components, in order to provide scalability and high availability. In the next chapter, the framework was evaluated in terms of scalability and availability using a load testing tool. In this way, the capability of the proposed framework as it runs on Windows Azure was examined.

Chapter 5

Evaluation Results

This chapter presents an evaluation of the proposed framework. The performance of the proposed framework was evaluated based on the amount of time required to send and execute different sets of requests from the client side (Raspberry Pi), and store these requests in the cloud storage. This is further discussed in Section 5.1. In Section 5.2, we explain the process of building a test environment that runs in the cloud. This involves applying load testing techniques to the system in order to evaluate the scalability and availability of the system itself.

5.1. Storing Sensor Data and Performance Measurement

To determine the real-world storage performance of the proposed framework, an experiment is performed to evaluate the framework where a large amount of sensor data is sent from the Raspberry Pi (client side) to the cloud. The goal is to measure the execution time of sending a different number of requests and storing these requests to the cloud storage.

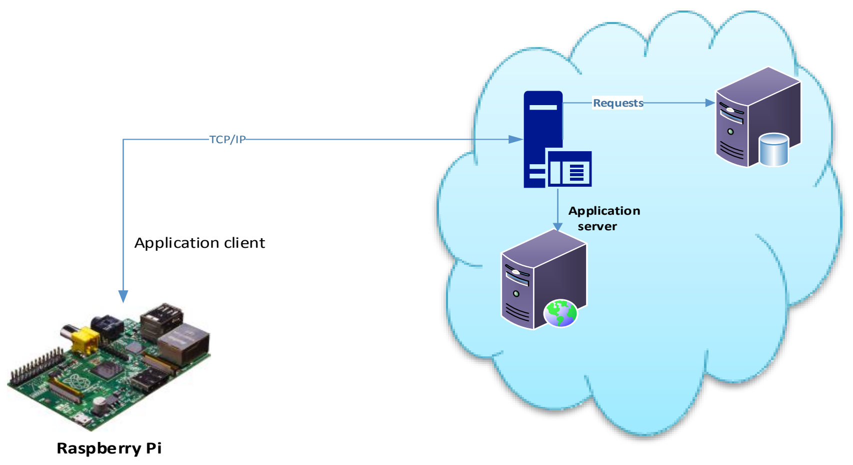


Figure 5.1: Storing Sensor Data in the Cloud

5.1.1. Experimentation Setup

Sensors have been used to monitor different aspects of the physical environment, and databases are typically used to store data generated by sensors. Thus, a key issue becomes the performance of virtual database servers and how the efficiency of this type of databases compares with physical ones. As mentioned in Chapter 4, we created the database in the cloud using a Microsoft SQL Server which runs in an Azure Virtual Machine. The SQL Server is a product used to manage and store data. The goal of this experiment is to measure the performance of sending sensor data from the Raspberry Pi, and storing such data in the cloud.

On the Raspberry Pi, an application client has been written in C++. The most important function of this application is generating large amounts of random sensor data in order to create user requests. Each request includes user id, sensor id, and sensor data. On the cloud side, an application server has been written in C#, with the aim of this application being to receive incoming requests from the client side, and to storing these requests in the cloud storage as shown in Figure 5.1.

Table 5.1 presents the average execution time in milliseconds of storing a different number of requests in the cloud. The application client code is detailed in Appendix F, while the code for the application server is included in Appendix G.

5.1.2. Experimental Results

Table 5.1 shows the average execution time of sending a different number of requests from client side and storing these requests to the cloud. In order to ensure that the results are accurate, the experiment was performed six times. Using these results, an average execution time was calculated. In this experiment, scalability and availability techniques were not applied.

Table 5.1: Performance Results of Storing Sensor Data in the Cloud

Number of requests	T1	T2	T3	T4	T5	T6	Execution time (Average) millisecond
1	2090	2117	1957	2112	1908	1856	2007
10	11159	10091	10618	15713	15818	9585	12164
100	18720	25693	22386	25287	22823	20280	19327
500	127777	125835	128103	125619	127238	126957	126921.5
1000	213601	209981	211787	213263	209856	211238	211621

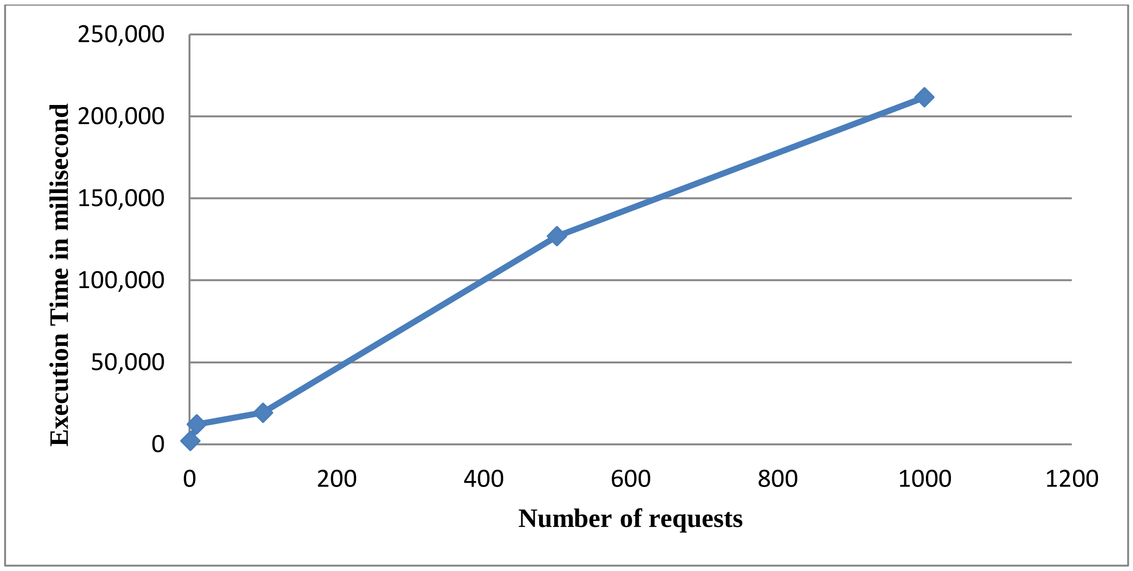


Figure 5.2: Performance Results of Storing Different Numbers of Requests in the Cloud

Figure 5.2 shows the execution time of sending a different number of requests from the client side and storing these requests to the cloud. The horizontal axis on the figure shows the number of requests, and the vertical axis shows the execution time in milliseconds. The execution time increases gradually when large amounts data have been stored as the number of requests increases. The results indicate that the system is able to transfer and store a large number of requests in a reasonable time.

5.2. Load Testing in the Cloud

In order to ensure that the proposed framework has high scalability and availability, we built a load testing environment in the cloud. Through this environment, we were able to evaluate the behavior of the Azure Load Balancer and the Traffic Manager. Details of our implementation have been included in Chapter 4. Load testing and stress testing are used to measure the performance of the system. Such tests are often performed during the application development phase, to ensure that the application has the ability to handle the expected level of load by simulating multiple virtual users. We evaluate the load balancer to investigate the following determinations:

- to determine the number of application servers that are required to support various traffic levels;
- to investigate the number of load balancers required to distribute the incoming traffic without a decrease in response time; and
- to examine the availability of the system, and to ensure that the application is able to continue to operate when a hardware or a network failure occurs.

Despite several public cloud providers offering computing services, there are different approaches in terms of infrastructure, virtualization, and software services. Windows Azure provides many features that consumers of cloud services require. The goal is to evaluate the functionality and performance of the scalability and availability services that are provided by Azure cloud providers, to help developers build high scalable applications that run in the cloud. Thus, we built cloud-based load testing which assisted us in determining if a system has the capability to handle incoming requests with a high load, and test the availability services when some of the VMs fail for any reason. The load testing environment has been built in the cloud because the cloud infrastructure provides consumers with unlimited computing resources, which are required to simulate a large number of virtual users.

5.2.1. Methodology

Testing the ability of the system to handle incoming requests is one of the most significant steps before deploying the system to the production cloud server. As mentioned in Chapter 4, an Azure load balancer and traffic manager have been applied to increase the scalability and availability of the system. Load testing is to examine if a system has the capability to handle incoming requests with a high load. Such a high load can be generated using a load testing tool such as JMeter, which can simulate thousands of virtual users for facilitating load testing under controlled stress conditions [50]. We have run our load testing to examine the system in the following cases:

- **Using a Single Virtual Machine**

  We evaluate the framework performance when the system is implemented using a single VM only, with no load balancing techniques being applied.

- **Load Balancing technique has been applied**

  The framework has been evaluated after applying the load balancing technique to the framework architecture, with the system running on multiple VMs.

- **Traffic Manger technique has been applied to the system**

  After the Azure Traffic Manger has been added to the framework architecture, we load testing to the system, to examine whether the scalability and availability of cloud services increased the system's performance and availability.

- **Applying both techniques (the Load Balancer and Traffic Manger) to the proposed framework**.

  In order to achieve the highest performance, we apply both the Azure Load Balancer and the Traffic Manager. We then evaluate the performance using the load testing tool.

5.2.2. Experimental Design and Implementation

This section describes the implementation of building the load testing environment in the cloud. We had to build the testing environment in the cloud because load testing execution requires high computational resources, which are very important for simulating a high number of virtual users. This allows us to determine the point at which the system's response time degrades or fails. Figure 5.3 on the following page presents the architecture of building the load testing environment in the cloud.

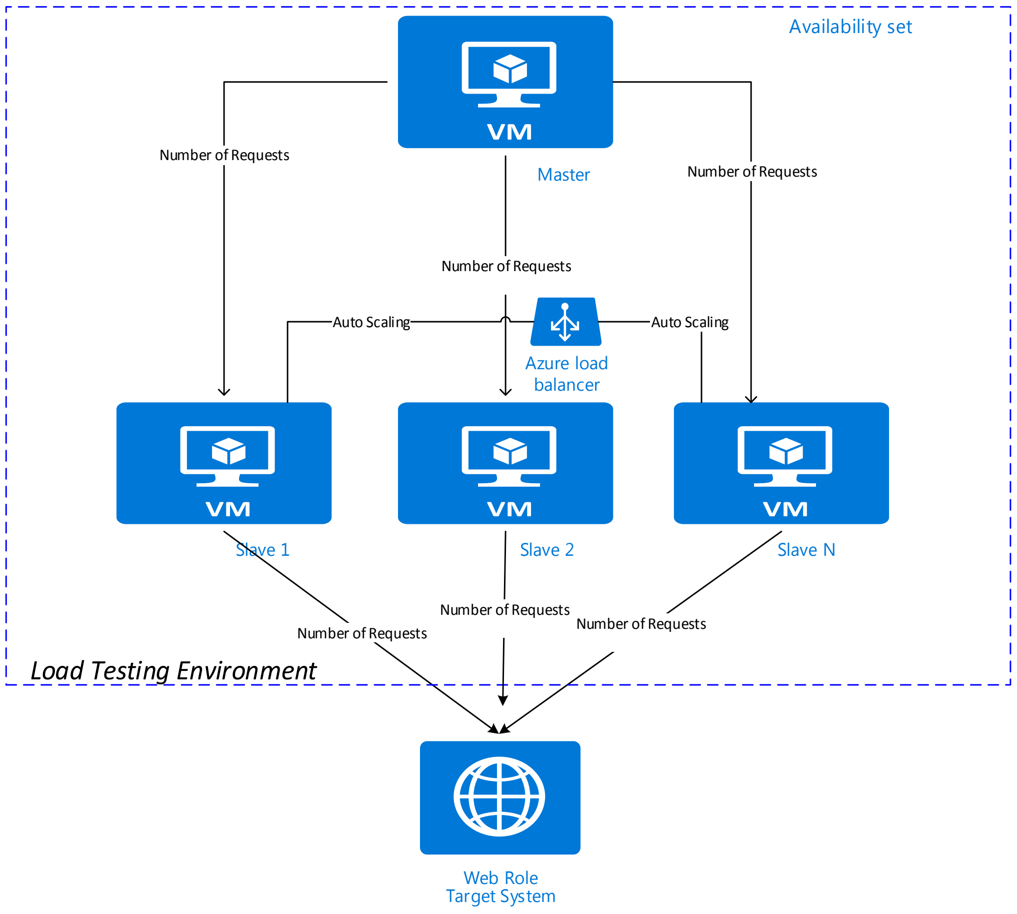


Figure 5.3: Architecture of Building Load Testing Environment in the Cloud

The architecture components in Figure 5.3 include VMs and the Azure Load Balancer. The size of all VMs is the same as in Table 5.2. All VMs have been created in a specific region located in Western Europe. We add the load balancer and auto-scale techniques to the load testing architecture, in order to increase the number of load generating slaves, based on auto scaling rules. For example, when the average CPU of other load slaves reaches a certain level, more VMs are initiated as load generators automatically.

The JMeter tool is an open source testing software that is used to apply load and performance testing to a target system. This tool can be used to generate a heavy load on a server to test the overall performance under different load types. In order to build a load testing environment in the cloud, we created a number of VMs for generating a large number of virtual users.

After we created these VMs, we configured one VM as a master, which is responsible for distributing the load of virtual users and requests among the Slave VMs. Slave VMs are responsible for performing the master's command by sending the requests to the target system. Finally, each Slave VM sends back the results to the master, including the response time and throughput. The single machine cannot simulate the real characteristics of massive user requests due to limitations of server resources. As a result, load testing has been built in the cloud which provides unlimited resources. The following steps clarify how we apply the load testing environment in the Windows Azure:

Table 5.2: The Size of VMs Used for Load Testing

Location	Size	CPU Cores	Memory	Disk Size
Europe West	Standard\A2	2	3.5 GB	135 GB

1. We created three Virtual Machines (VMs) that run in Azure. The size of these VMs is presented in Table 5.2. We ensure that all VMs running in the same availability are using the same Azure Load balancer. The number of VMs can be increased and auto-scale can be applied if more hardware resources are required to simulate more virtual users.
2. The same version of Apaches JMeter and Java JDK are installed on all the VMs.

3. We define one VM as a Master and the other two VMs as Slaves. The Master VM is responsible for controlling the test, and sending the commands to the Slave VMs. The Slave VMs are running as JMeter servers, and are responsible for receiving the commands from the Master VM and generating a number of requests to the target web role. The Web role runs using windows server 2012 with Internet Information Services (IIS).
4. Three important terminologies for building the load testing environment are :
    - Master: The VM running JMeter GUL, which is responsible for controlling the load test.
    - Slave: The VMs running JMeter-Server, which takes commands from the Master VM and sends the requests to web role target.
    - Target: The web-based server in which the load test will be executed.

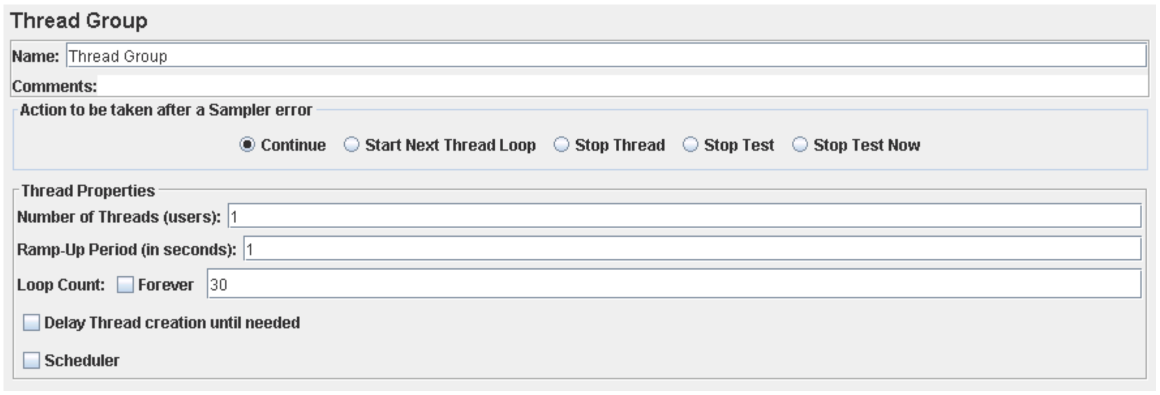
We ran the test 30 times to ensure that the results are accurate.

The following sections present the results of examining the proposed framework in terms of scalability and availability. We ran the load testing to evaluate the framework's performance in different cases, which are presented in the next four sections.

5.2.3. The System Running in a Single VM

In this section, we describe the first experiment, which was performed using a load testing environment. The proposed framework is evaluated before applying the scalability and availability techniques. In this experiment, the framework is designed to run in a single VM. This single machine may fail due to unexpected causes, such as out of memory exception or hardware defects. As a result, users cannot access the system because it is down.

Figure 5.4 shows a snapshot of setting a number of virtual users by using the JMeter tool while Figure 5.5 presents an example of the result of sending a number of requests to the cloud-based system. The green icons indicate that requests have been successfully executed.



Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users): 1

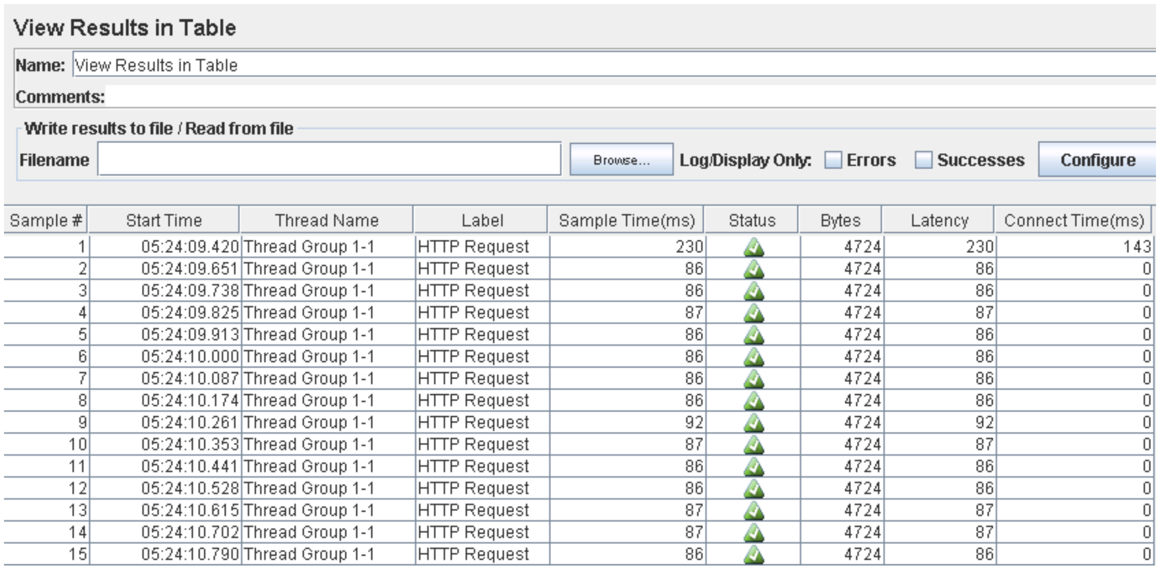
Ramp-Up Period (in seconds): 1

Loop Count: ☐ Forever 30

☐ Delay Thread creation until needed

☐ Scheduler

Figure 5.4: Thread Group Setting



View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Latency	Connect Time(ms)
1	05:24:09.420	Thread Group 1-1	HTTP Request	230	✓	4724	230	143
2	05:24:09.651	Thread Group 1-1	HTTP Request	86	✓	4724	86	0
3	05:24:09.738	Thread Group 1-1	HTTP Request	86	✓	4724	86	0
4	05:24:09.825	Thread Group 1-1	HTTP Request	87	✓	4724	87	0
5	05:24:09.913	Thread Group 1-1	HTTP Request	86	✓	4724	86	0
6	05:24:10.000	Thread Group 1-1	HTTP Request	86	✓	4724	86	0
7	05:24:10.087	Thread Group 1-1	HTTP Request	86	✓	4724	86	0
8	05:24:10.174	Thread Group 1-1	HTTP Request	86	✓	4724	86	0
9	05:24:10.261	Thread Group 1-1	HTTP Request	92	✓	4724	92	0
10	05:24:10.353	Thread Group 1-1	HTTP Request	87	✓	4724	87	0
11	05:24:10.441	Thread Group 1-1	HTTP Request	86	✓	4724	86	0
12	05:24:10.528	Thread Group 1-1	HTTP Request	86	✓	4724	86	0
13	05:24:10.615	Thread Group 1-1	HTTP Request	87	✓	4724	87	0
14	05:24:10.702	Thread Group 1-1	HTTP Request	87	✓	4724	87	0
15	05:24:10.790	Thread Group 1-1	HTTP Request	86	✓	4724	86	0

Figure 5.5: Sample Load Testing Run using JMeter

The aim is to investigate system capability when a high number of requests are received by a single VM, as shown in Figure 5.3. The results of applying load testing to the system while running in a single VM are presented in Table 5.3. The results indicate that the system has a high performance when the number of requests are fewer than 300. Following this, the average response time increases, to a peak of 232 milliseconds, for executing 1500 requests. In addition, we noticed that, at this point, some requests are executed with a low performance peak of 5230 milliseconds. There is a sharp increase in the average response time when the number of requests rose to 15,000. At this point, we noticed that some requests failed while others took a longer period of time to be processed as a single VM could not handle the large number of requests.

Table 5.3: The Results of Running the System in a Single Machine

Number of users	Number of requests	Average response time (in millisecond)	Min	Max	Throughput (in second)
1	30	90	86	202	11.1
10	300	89	85	212	83.4
50	1500	232	84	5230	172.2
100	3000	281	84	5314	313.3
200	6000	195	84	5220	687.4
300	9000	306	85	7649	821.5
500	15000	769	84	30073	451.4

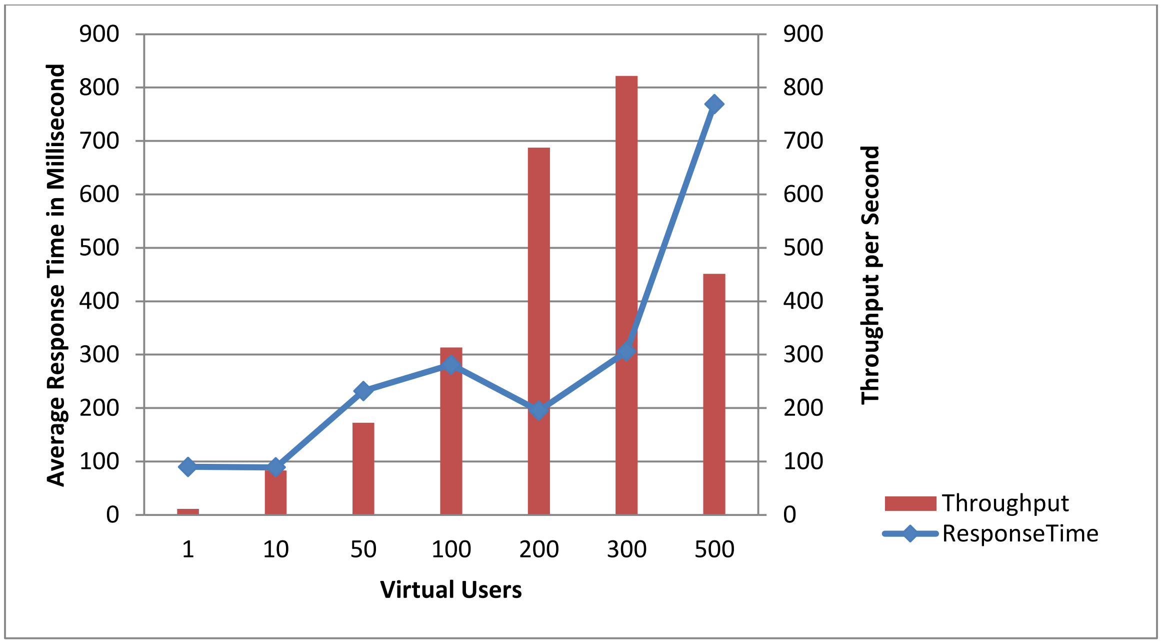


Figure 5.6: Results of Testing a System Running in a Single VM

- *Response time* is the difference between the time when the request was sent and the time when the response has been fully received.
- *Throughput* is the number of requests per unit of time. We divide the *total average time* by 1000 to convert milliseconds to seconds

Throughput = (number of requests) / (total average time / 1000)

Figure 5.6 shows the result of testing the system when it is running on a single VM. As a result, when a low number of requests are sent to a cloud-based system, the system is able to handle these requests with high performance. However, when the number of virtual users and requests are increased, the average response time peaks to a high response time.

In some cases, when the number of users was 200, the response time suddenly decrease compared with 100 users. The main reason is the load testing runs from different machines, and the frequent of the internet connection is not considered in this experiment.

5.2.4. System Performance After Applying Load Balancing

In order to solve low performance and availability, the load balancing technique was applied to the system. In this section, we evaluate the system when it is running in three VMs using the Azure load balancer to distribute incoming requests among these VMs. In Chapter 4, we explained in detail the Azure load balancer. Three VMs were created in the same availability set. We created all VMs in the same availability set in order to apply the auto scale technique. We configured the cloud services to automatically scale up and scale down based on average CPU usage. We set the range of average CPU from 60% to 80%. When the average percentage of CPU is higher than the maximum setting, two VMs are turned on. In contrast, when the average percentage of CPU is lower than the minimum setting, the two VMs are turned off. We set ten minutes to wait between the last scale action (which can be a scale-up action or a scale-down action), and the next scale action.

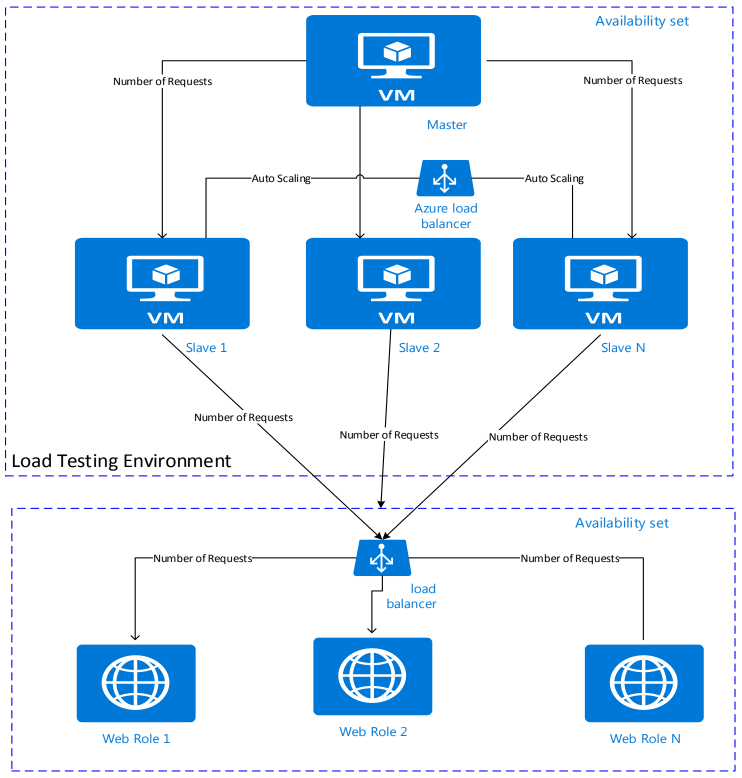


Figure 5.7: Architecture of Setting Load Testing after Applying Load Balancing Technique

Figure 5.7 presents the architecture of setting the load testing in the cloud in order to test the system after applying the load balancing technique. The results of running this load test are shown in Table 5.4. The results indicate that the system's performance improved after applying the load balancing technique. Thus, it is evident that, after applying the load balancing technique, the system was able to serve up to 15,000 requests with high performance within a reasonable response time. However, when the system was running on a single machine, the system was able to handle 9000 requests.

We also noticed that when requests surpassed 15,000, although there were no failures, the response time of some requests increased. In order to increase the capability of the system for serving more requests, more VMs were turned on.

Table 5.4: Result of Testing the System after Applying Load Balancing Technique

Number of Users	Number of requests	Average Response Time (Millisecond)	Min	Max	Throughput (per second)
1	30	98	94	216	10.1
10	300	97	93	235	78.5
50	1500	97	93	194	383
100	3000	97	93	217	766.7
200	6000	104	93	240	1435.5
300	9000	145	93	347	1609.7
500	15000	231	93	5392	1332
800	24000	251	93	11954	1412.2

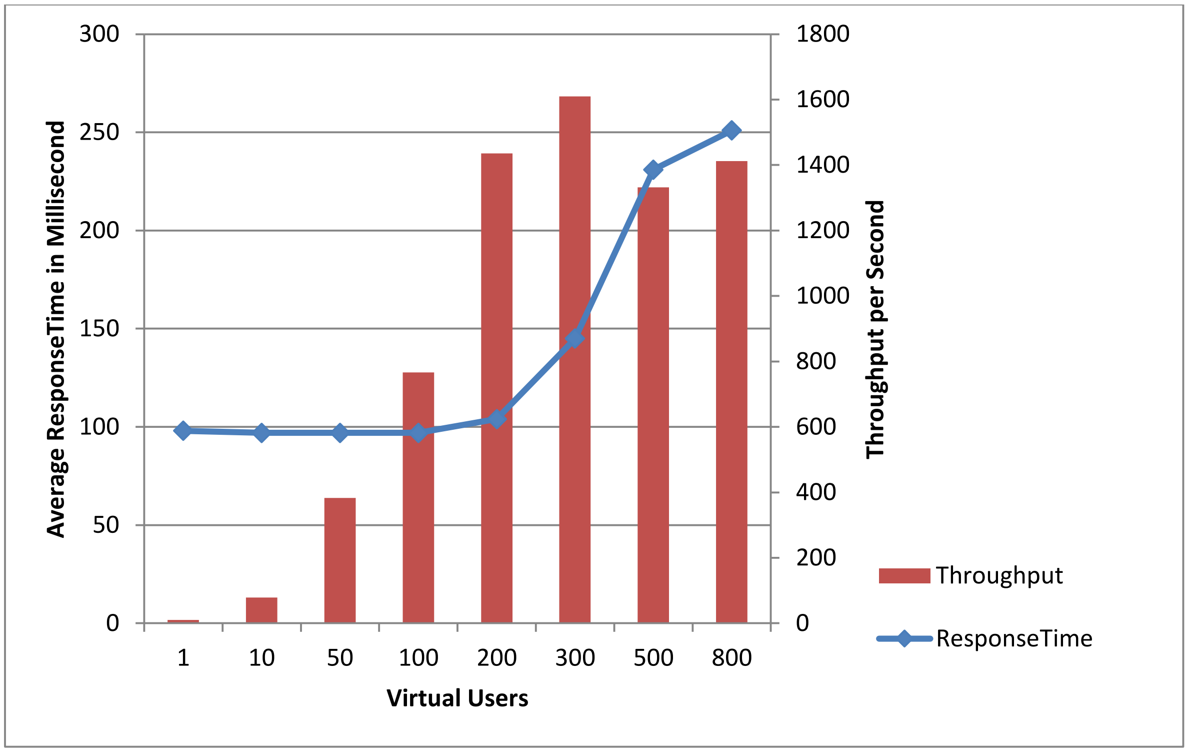


Figure 5.8: Performance of the System after Applying Load Balancing

As shown in Figure 5.8, after applying the load balancing technique to the system, the performance was increased and the response time was reduced significantly. We noted that the response was even faster than when using a single virtual machine. Thus, when the number of VMs has been increased and the Azure load balancer and auto-scale have been applied, high system performance has been achieved. In addition, we tested the availability of the system during the load testing time by turning off the one of the available VMs that was connected to the load balancer. In order to evaluate the availability of the framework, when we turned off one VM, we noticed that the load balancer automatically transfers incoming requests to a healthy VM. After the unavailable VM becomes healthy, the load balancer automatically starts to send incoming requests to this VM.

5.2.5. The System Performance after Applying Traffic Manger

This section describes the design and implementation of building a system that can provide high availability using the Azure Traffic Manager, as well as evaluating the performance of the Traffic Manager. Figure 5.9 presents the Architecture of setting the load testing in the cloud with applying Traffic Manager technique to the system.

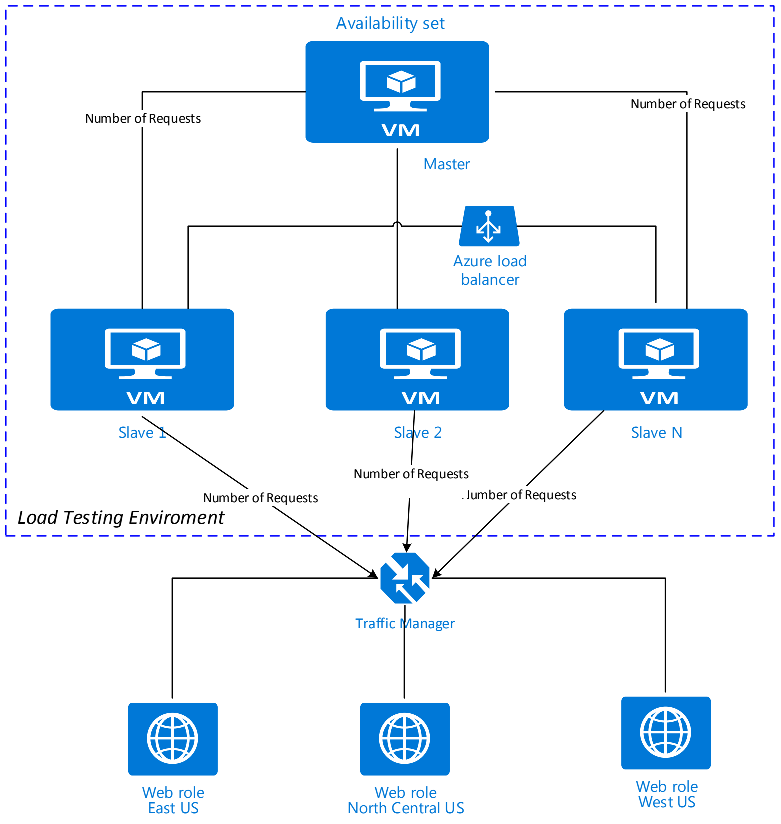


Figure 5.9: The Architecture of Setting the Load Testing in the Cloud with Applying Traffic Manager Technique to the System

The Traffic Manager offers different load balancing techniques include Performance, Round Robin, and Failover. When we evaluated the Traffic Manager performance, we set the configuration in Round Robin method. We created three VMs that are running in different regions as shown in Table 5.5. The main advantage of using the Azure Traffic Manager is to increase the availability of the system, as the Traffic Manger supports applications to run in different data centers around the world. After the application of the Traffic Manager, the availability improved. In the instance where one VM becomes unavailable in a specific region, other VMs can be employed to handle the requests.

Table 5.5: Three VMs Running at Different Regions

Location	Size	CPU Cores	Memory	Disk Size
East US	A2\Standard Tier	2	3.5 GB	135 GB
North Central US				
Europe West				

Table 5.6: The Result of Testing the System after Applying Traffic Manager

Number of Users	Number of Requests	Average Response Time (In millisecond)	Min	Max	Throughput (In Second)
1	30	91	86	225	40.8
10	300	99	85	311	147.1
50	1500	95	85	255	705.6
100	3000	115	84	347	1419.1
200	6000	161	84	399	1151
300	9000	178	84	412	1100.8
500	15000	370	84	5675	989.9
800	24000	505	85	5886	1331.8
1000	30000	359	84	10381	1224.8

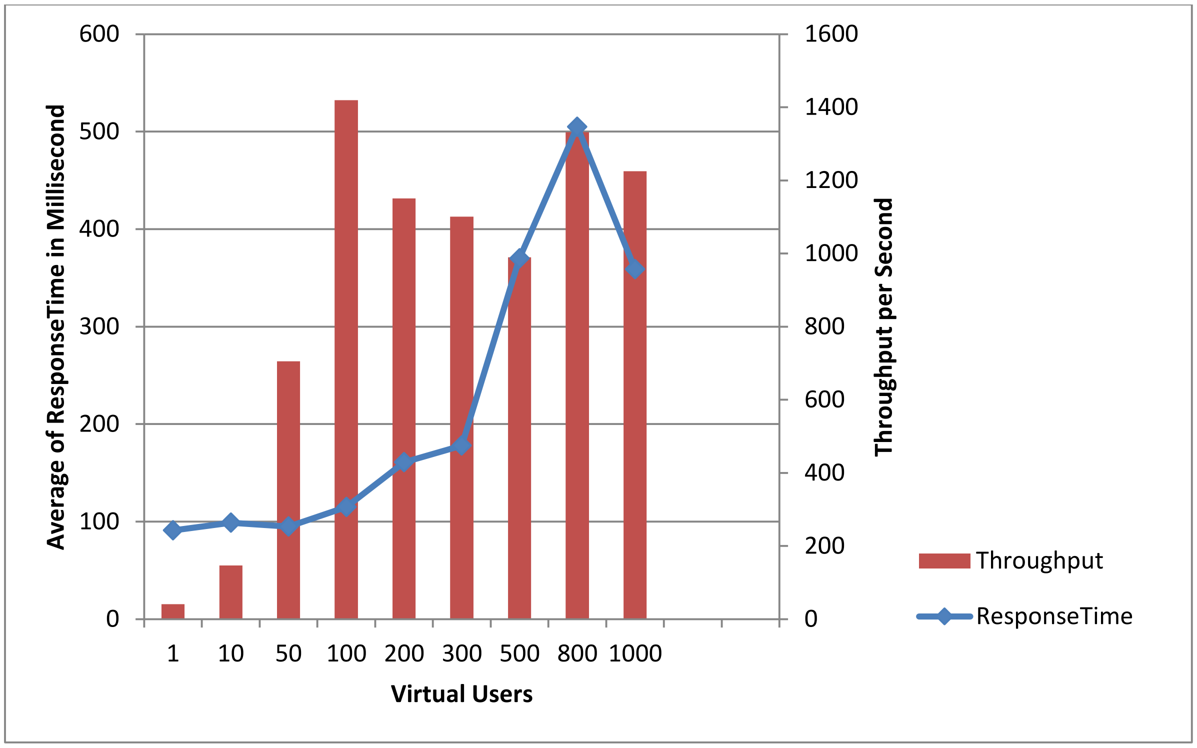


Figure 5.10: Performance of the System after Applying Traffic Manager

Figure 5.10 shows the performance of the eHealth smart system after applying the Azure Traffic Manager where the configuration is set in a Round Robin method. Even though the virtual machines were created in different regions, we noticed that the performance increased and that the average response time was similar to the result of applying the Azure load balancer as shown in Table 5.6. Furthermore, the increase in latency time was directly based on the location of the data center. Table 5.5 shows the size and location of each VM. When applying the Traffic Manager using the performance method, the performance was increased because the Traffic Manger transferred incoming requests to the closest available data centers. When we tested the system with the Traffic Manger being configured in the performance method, all incoming requests transferred to one VM running in the Western Europe region. This was because the load testing environment was running in the Western Europe region, so all requests had been sent to Western Europe VM. We present in the next section a new approach to using the Azure Load balancer and the Traffic Manager in the same architecture, to help customers and developers achieve high scalability and availability. As shown in Figure 5.10, when the number of virtual users rose up 1000, the average response time suddenly decreased compared with 800 users. The reason behind this case is the load testing run on different VMs that have different CPU and memory performance, and the load testing had applied at different time. Moreover, in this experiment, the internet connection performance measurement was not considered. Thus, all these aspects can affect the average response time.

5.2.6. The System Performance after Applying Load balancer and Traffic Manger

After we examined and evaluated the system under different scenarios, we designed and implemented a high scalable and available architecture running in Windows Azure, as presented in Figure 5.11 in the next page. We feel that this architecture achieves the desired result of establishing an architecture that ensures high scalability.

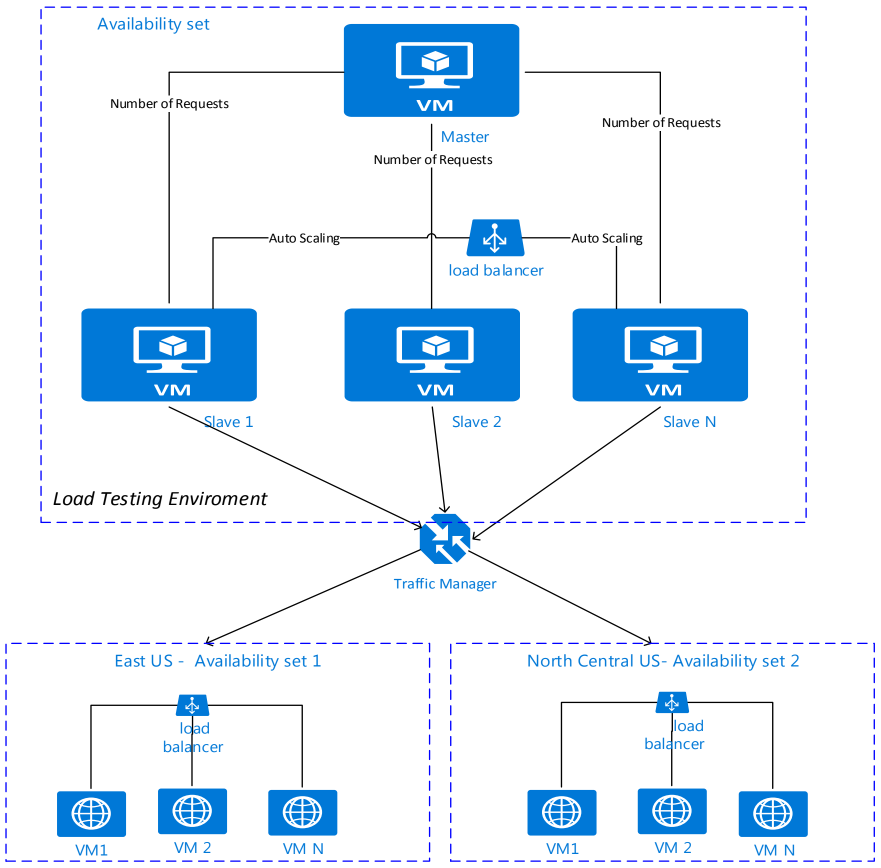


Figure 5.11: Architecture of Setting Load Testing after Applying the Load Balancer and Traffic Manager Techniques

Table 5.7: The Results after Applying the Load Balancer and Traffic Manager Techniques

Number of Users	Number of Requests	Average Response Time (In millisecond)	Min	Max	Throughput (In Second)
1	30	91	86	237	10.8
10	300	104	91	313	69.3
50	1500	100	85	235	353.1
100	3000	96	84	385	717.0
200	6000	131	84	5245	634.3
300	9000	161	85	5246	940.7
500	15000	199	85	5413	1468.9
800	24000	232	84	5346	1546.9
1000	30000	193	84	5291	1895.1
1100	33000	208	84	5337	2023.1
1200	33600	248	84	5471	1809.3
1400	42600	312	85	7566	1751.9

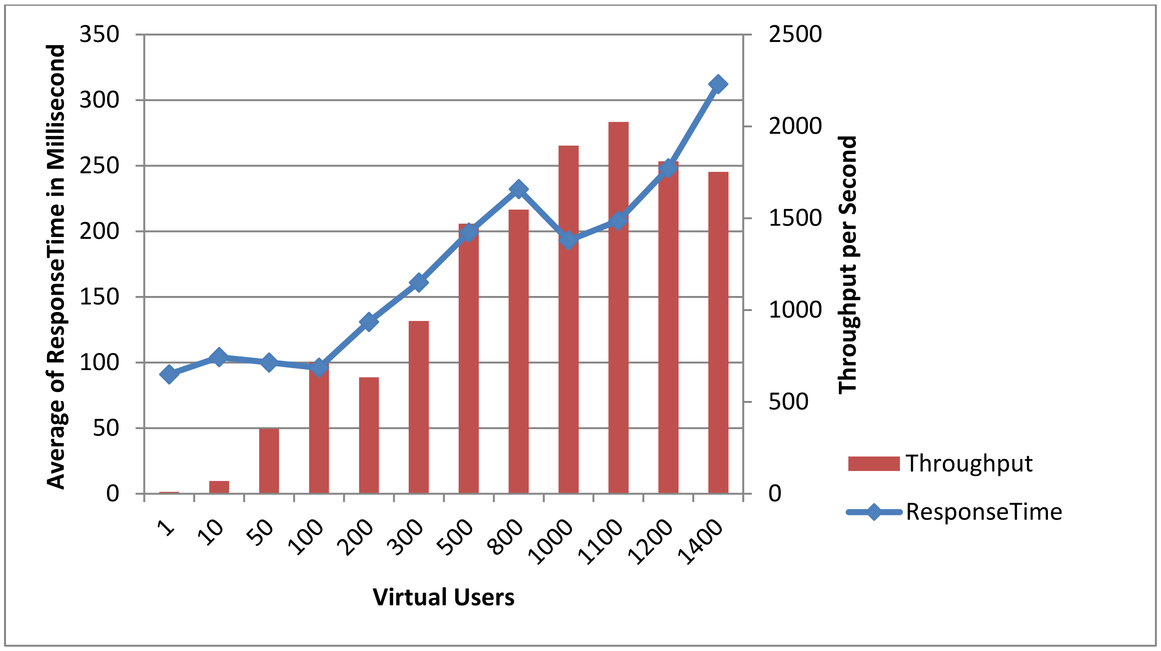


Figure 5.12: Applying Traffic Manager and Load Balancer

As shown in Figure 5.12, the system performance has increased, and a large number of requests were executed with low response time. The proposed framework provided scalability after applying load balancing and auto scaling techniques. In addition, a high availability was achieved after applying Azure Traffic Manager. Table 5.7 shows the results of a run load test after applying the scalability and availability techniques. These techniques were applied in order to achieve height performance and availability using Azure as a cloud provider. We also notice in some cases that the frequent of the internet connection, which is not considered in this experiment, can affect the response time. For example, when the number of virtual users rose up 1000, the average response time suddenly decreased compared with 800 users. In addition, the load testing run on different VMs that have different CPU and memory performance, and the load testing had applied at different time. As a result, all these aspect can affect the average response time.

5.3. Summary

This chapter presented a detailed description of the experiments that had been conducted to evaluate the framework performance. The performance evaluation of storing sensor data in the cloud was presented. This experiment evaluated the execution time of sending different numbers of requests, including sensor data, to the cloud. Load testing was implemented to examine the performance of the cloud-based system after applying the scalability and availability techniques. The results have proved that after applying the Azure Load Balancer and the Traffic Manager, the framework performance was increased and the system was able to handle a large number of virtual users. In the next chapter, the design and implementation of an eHealth smart system will be presented as a case study.

Chapter 6

Case Study: An eHealth Smart System

This case study applies the proposed framework to the eHealth system. The two main objectives of this case study are as follows: (1) presenting a proof of concept for designing and implementing a high scalable framework that uses medical sensors as infrastructure, and (2) evaluating the performance of the system by simulating an environment where a number of users are utilizing the system. This chapter provides an overview of the eHealth Smart System architecture, which is presented in Section 6.1. The components and experimental setup of the system are demonstrated in Section 6.2. The design and implementation architecture of the case study is described in Section 6.3. The evaluation of the eHealth Smart System is presented in Section 6.4.

6.1. Overview

This case study presents the design and implementation of an eHealth smart network system. The eHealth system is designed to prevent delays in the transmission of patient medical information to healthcare providers. For example, in accident and emergency situations, where timing is crucial, it eliminates the need to manually enter data. An effective eHealth system would also contribute to the efficient use of hospital resources, as administrators would be able to make effective decisions based on real-time data. The architecture for this system is based on the use of medical sensors, which, once connected to the patient, allow for the measurement of a patient's physical condition. These sensors transfer data collected from a patient's body, over the wireless network, to cloud services. The system is responsible for generating appropriate medical decisions using a decision making algorithm, which is based on standard medical practices and a patient’s historical medical data. Proposed decisions will be sent to medical staff, responsible for a patient’s healthcare, for approval. The system sends final decisions to the patient after medical staff approval. Therefore, the patient will have high quality services because the eHealth Smart System supports medical staff by providing real-time data gathering, eliminating manual data collection, and enabling the monitoring of a large number of patients at one time.

There are many advantages of using this system, such as:

1) providing real-time data gathering;
2) eliminating the manual data collection process, which sometimes includes data entry errors;
3) monitoring a large numbers of patients, who are depending on a limited number of medical staff;
4) ensuring that bed occupancy in hospitals is only for patients who need them; and
5) helping medical staff from different health providers, benefit from each other's experiences.

6.2. Physical Components of eHealth System

In the case study, we used commercial medical sensors which are connected with an eHealth Sensor Shield that is placed on top of the Raspberry Pi. The eHealth Sensor Shield and medical sensors have been designed by Cooking Hacks, in order to assist researchers and developers in measuring real-time sensor data, which can be used for experimentation purposes [51]. Given that the eHealth platform does not have medical certification, its use is limited. It cannot be used to monitor critical patients who need accurate medical monitoring devices. The eHealth Sensor Shield was originally designed for Arduino. In order to utilize the eHealth Sensor Shield on Raspberry Pi, we used a Connection Bridge, which allows any shield, board, or module, originally designed for Arduino, to be used with Raspberry Pi. Figure 6.1 further illustrates this concept.

Even though many medical sensors are available, we used only two types of sensors to minimize the cost of our experiment. The Pulse and Oxygen in Blood sensor (SPO2) allows one to measure the amount of Oxygen saturation in blood. The Body Temperature sensor, allows one to measure an individual’s body temperature. Figure 6.1 shows the components that were utilized in the experimental setup, including the Raspberry Pi, the eHealth Sensor Shield, and the Raspberry Pi to Arduino Shields Connection Bridge. Medical sensors and other main components are connected together, as shown in Figure 6.2.

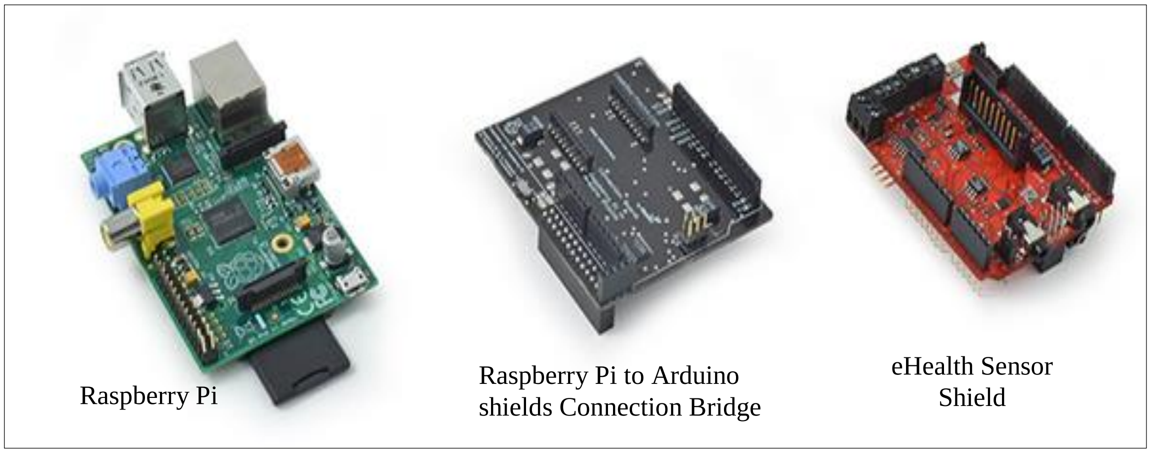


Figure 6.1: eHealth System Components

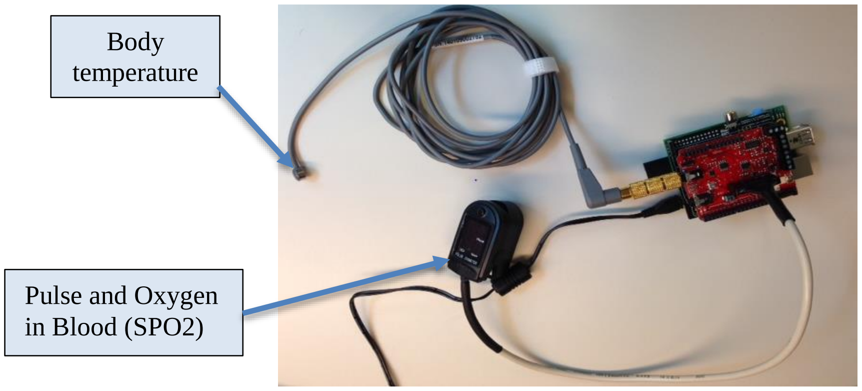


Figure 6.2: Medical Sensors and eHealth Sensor Shield Connected to Pi

6.3. eHealth Smart System Architecture

This architecture is designed based on the integration between medical sensors (which are responsible for measuring a patient's physical medical condition), and the cloud environment. This integration becomes the basis for a smart medical system.

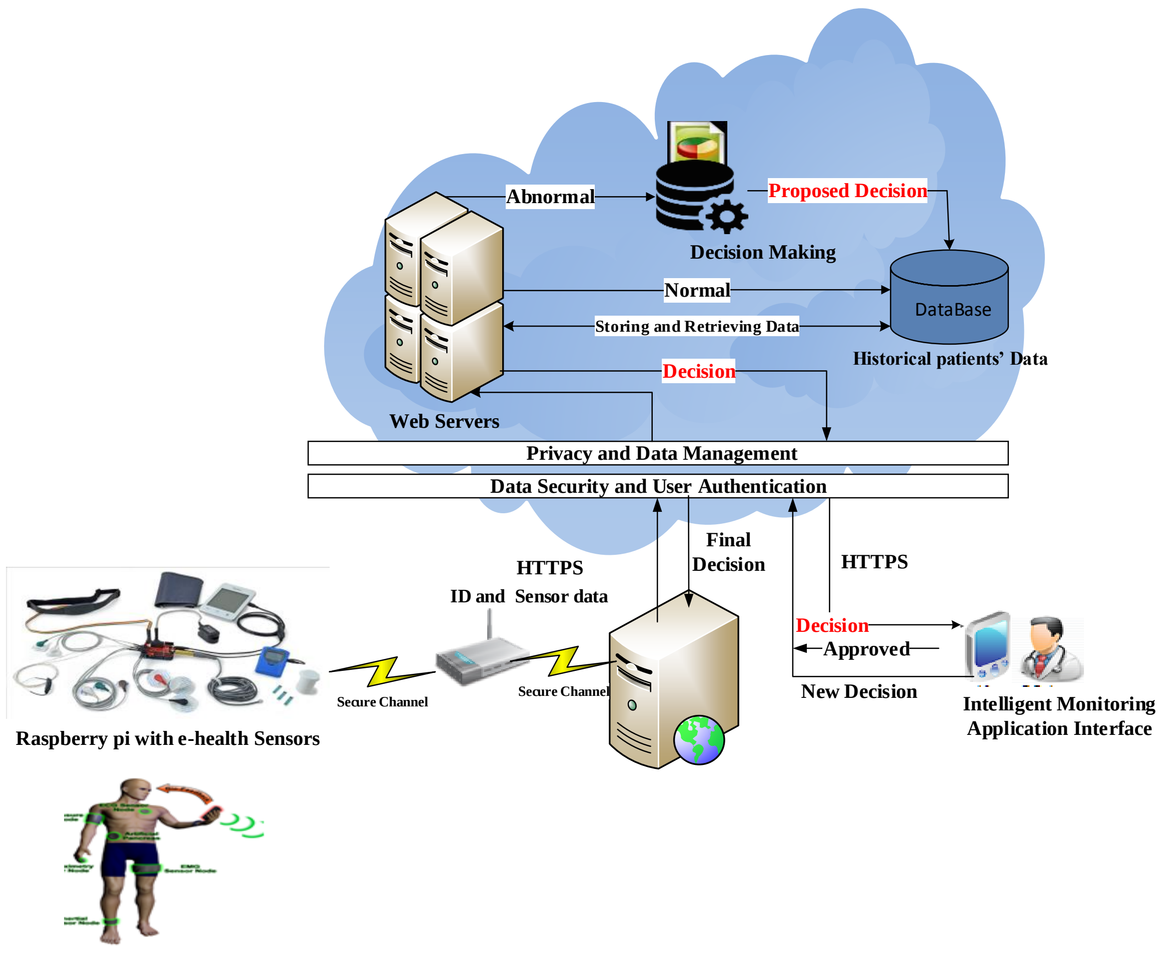

Abnormal
Proposed Decision
Decision Making
Normal
DataBase
Storing and Retrieving Data
Historical patients' Data
Decision
Web Servers
Privacy and Data Management
Data Security and User Authentication
Final Decision
HTTPS
HTTPS
ID and Sensor data
Decision
Approved
New Decision
Secure Channel
Secure Channel
Intelligent Monitoring Application Interface
Raspberry pi with e-health Sensors

Figure 6.3: eHealth Smart System Architecture

As shown in Figure 6.3, there are medical sensors that are connected with Raspberry Pi. This Raspberry Pi is responsible for collecting data from the sensors, and transmitting this data, through wireless communication channels, to platform services hosted in the cloud. This platform provides the following services:

(1) storage service, which is responsible for storing patient data;

(2) data analysis service, which is responsible for providing medical decisions based on a patient's historical medical data; and

(3) managing service, used for updating, reviewing and testing a patient's data that is needed by medical staff. Medical staff and patients can utilize the cloud application from different mobile and stationary devices, using the Internet. Figure 6.4 shows the system's home page.

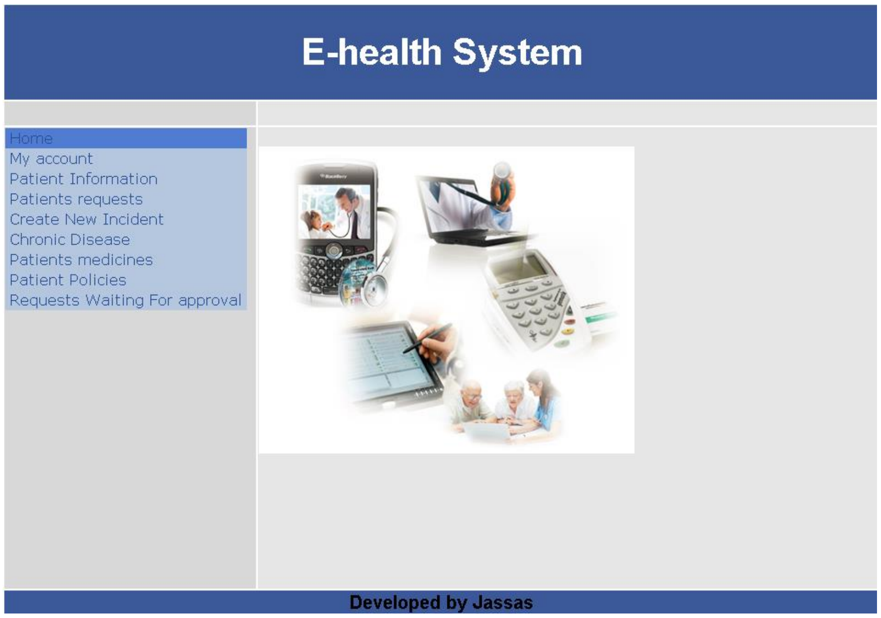


Figure 6.4: The Home Page of eHealth Smart System Running in Windows Azure

Security and privacy are two significant factors that relate to the cloud environment because cloud computing is a multiple-range environment, in which numerous computing resources are shared. The sharing of hardware resources and storage data areas in the cloud present a high risk of insider or outsider attacks. In order to achieve data security and privacy in our system, we apply three techniques, which are as follows:

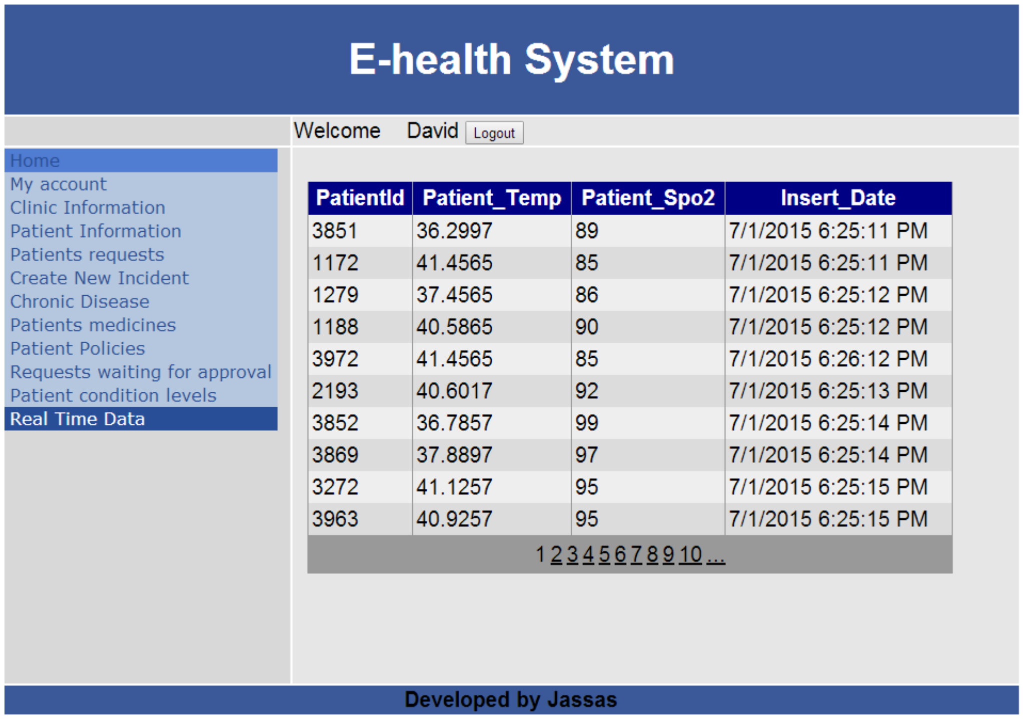
1) Socket Secure Layer (SSL), which is one of the most popular techniques for establishing an encrypted channel between a web server and the Raspberry Pi, to transmit patient data to the cloud application, through secure channels;

2) Cryptography Algorithm - Since SSL is not responsible for encrypted data in a cloud, we use cryptography algorithm to ensure that the data has been stored in an encrypted format; and
3) Hashing user authentication passwords.

More details for implementing security and privacy techniques are mentioned in Chapter 4. The following sections describe the design and implementation of the eHealth system in detail:

A) Data Collection

Medical sensors measure patient physical parameters using wireless sensors. All medical sensors are connected to the Raspberry Pi to read data from patients' bodies and transfer the data (patient id and sensor data) over the wireless network channel to the cloud environment. To send sensor data from clients to the cloud platform, we use socket programming, which is the sending or receiving of data over the TCP/IP protocol. The code was written in C++ programming language (the source code is listed in Appendix A). From the perspective of security, data will be transmitted to the cloud through an SSL secure channel. These sensors generate real-time patient data, with the Raspberry Pi being responsible for sending the data to the application server that runs continuously in the cloud. We also configured the Raspberry Pi application to apply delay time, because we noticed that the body temperature sensor takes time to measure the right body temperature. Figure 6.5 presents the real-time sensor data for patients.


E-health System

Welcome David Logout

PatientId	Patient_Temp	Patient_Spo2	Insert_Date
3851	36.2997	89	7/1/2015 6:25:11 PM
1172	41.4565	85	7/1/2015 6:25:11 PM
1279	37.4565	86	7/1/2015 6:25:12 PM
1188	40.5865	90	7/1/2015 6:25:12 PM
3972	41.4565	85	7/1/2015 6:26:12 PM
2193	40.6017	92	7/1/2015 6:25:13 PM
3852	36.7857	99	7/1/2015 6:25:14 PM
3869	37.8897	97	7/1/2015 6:25:14 PM
3272	41.1257	95	7/1/2015 6:25:15 PM
3963	40.9257	95	7/1/2015 6:25:15 PM

Figure 6.5: Real-time Data Generated from Medical Sensors

B) Decision Making Algorithm

Figure 6.6 shows the system decision making algorithm process. We used the Microsoft SQL Stored Procedure technique in order to implement this algorithm [52].

The decision making algorithm procedures are responsible for creating appropriate medical decisions based on three parameters, namely, patient id, sensor type, and sensor current data. When the application receives the data from the sensors, the algorithm will check if the sensor data is normal or abnormal based on the normal ranges of laboratory medical tests and patient medical policies, which are defined in the system. In health-related fields, normal range of laboratory medical tests usually describe the variations of a measurement or value in healthy individuals. Reference ranges are often determined by taking the lowest and highest values of results obtained from a normal population [53].

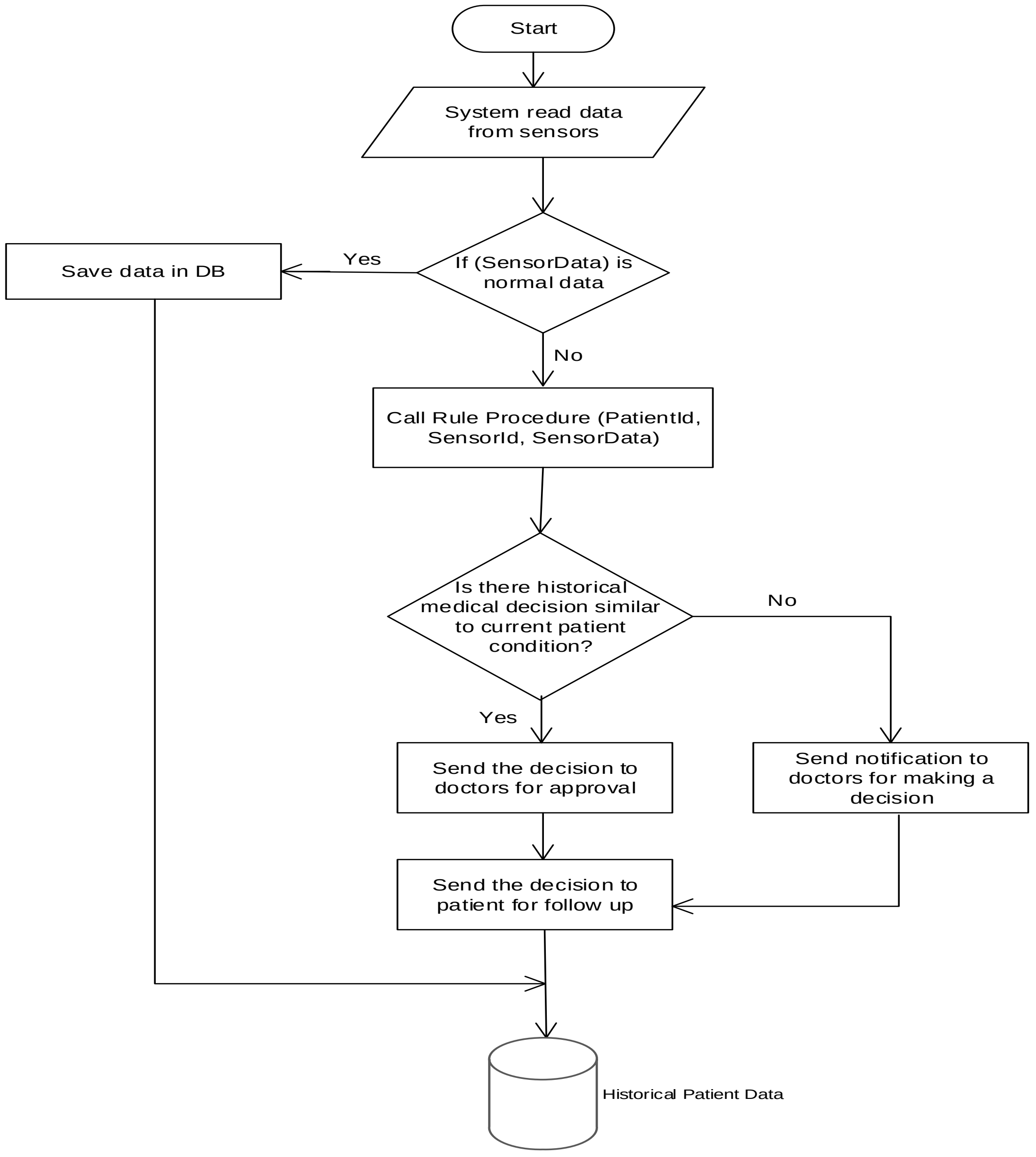

Start
System read data from sensors
If (SensorData) is normal data
Yes
Save data in DB
No
Call Rule Procedure (PatientId, SensorId, SensorData)
Is there historical medical decision similar to current patient condition?
No
Yes
Send the decision to doctors for approval
Send notification to doctors for making a decision
Send the decision to patient for follow up
Historical Patient Data

Figure 6.6: Decision Making Process

Every patient has a medical policies profile in the system based on the sensor type, to assist the system in creating compatible medical decisions. For example, some patients have chronic diseases, so in these cases, medical staff should consider some change to normal ranges of laboratory medical tests. Medical staff can set specific policies for every patient based on patient condition. The following are two examples:

Example 1:

If Patient id = "1000001" & SPO2 < "89" & TEMP > "37.5" = "Set Status is Abnormal" & Making Decision & (send notification to the doctor).

Example 2:

If Patient id = "1000002" & SPO2 < "85" & TEMP > "39.5" = "Set Status is Abnormal" & Making Decision & (send notification to Emergency Medical Services (EMS).

If the data is normal, the algorithm will store this data in sensor information tables in the database to feed patient historical data. Otherwise, the algorithm will create a medical decision based on the patient's historical medical data. If the patient does not have any historical medical data for the same condition, the system will make a medical decision based on historical statistical data of patients who have a similar health conditions. Before storing the data in the cloud, our system is responsible to encrypt the data using a cryptography algorithm. Therefore, the data is stored securely in the cloud.

C) Decision Approver

After the system makes appropriate decisions, these decisions are sent to medical staff who are responsible for patient healthcare, for approval. There are different ways to notify medical staff, such as SMS and email. They can use the browser or their mobile devices to review and update these decisions. Medical staff are responsible for reviewing patient historical information based on their medication and chronic diseases. After that, medical staff can decide if the decisions created from the system are appropriate for the patient's condition, or whether they need to change and update them. After the medical staff approve the decisions, the system will confirm them, save them, and send the confirmed decisions to the patient to follow the staff's instructions.

6.4. Performance Evaluation and Results

This section describes the experiments that have been applied to evaluate the performance of the system while uploading the sensor data from a Raspberry Pi to the cloud storage. We applied these experiments by using one body temperature sensor (Pulse and Oxygen) because we wanted to minimize the cost of the experiment. It is important to note that most of the performance evaluation results are obtained based on generating random patient data.

The experiment was conducted myself as the subject patient. In this experiment, we focused on measuring the body temperature for myself at different times. The purpose of this experiment was to prove that the system is able to continuously monitor during a 24-hour period.

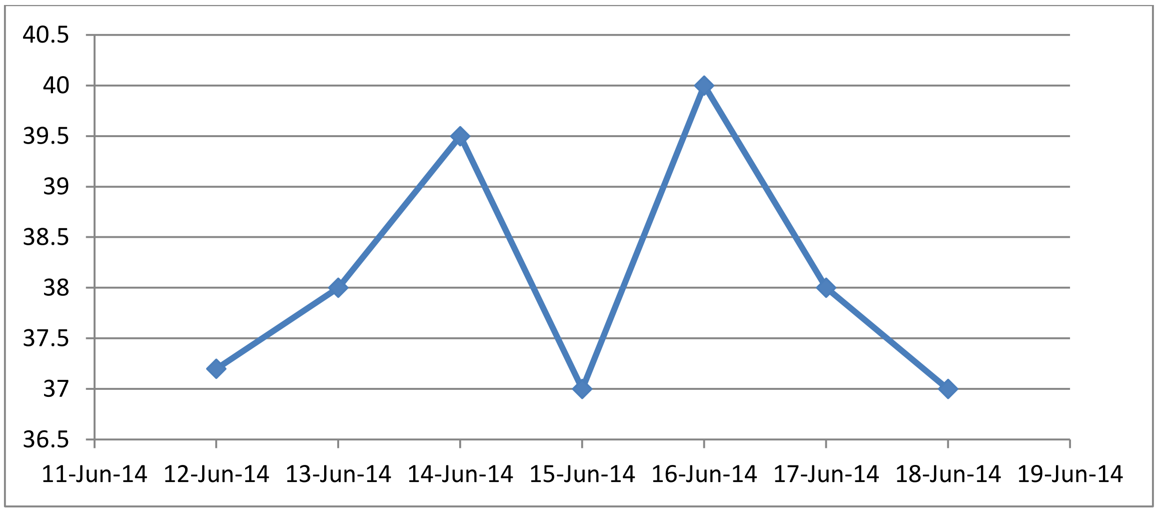
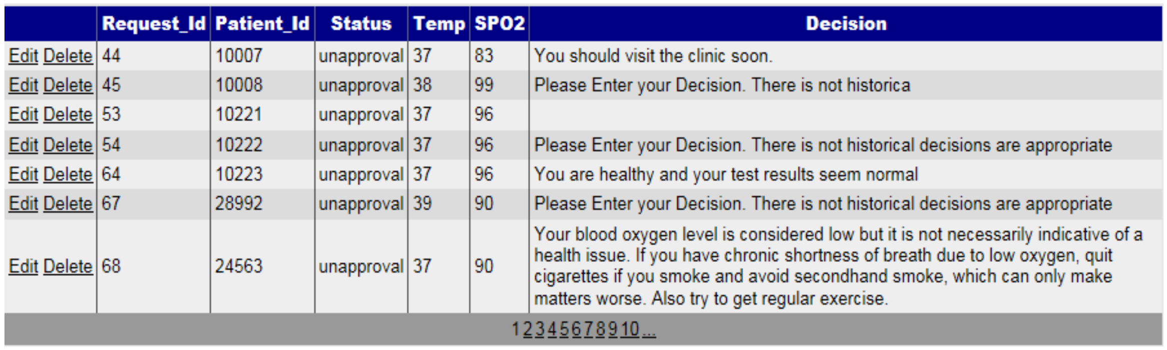


Figure 6.7: Body Temperature Monitoring for One Patient at Different Times

We collected body temperatures for one patient at different times from June 12, 2014 until June 18, 2014. Thus, medical staff can easily monitor the patient during any chosen day, as shown in Figure 6.7. The medical staff also can make decisions based on the patient’s condition, if they noticed that the patient was an emergency case. To ensure a high level of accuracy, as the body temperature sensor takes time to generate the correct temperature, we applied the delay technique. We noticed that the 30-second delay in collecting body temperature will assist obtaining high accuracy in temperature measurement because the temperature sensor takes 30-second in order to read the correct temperature. However, applying the delay technique is not required if the patient already wearing a private sensor. As a result, we noticed that every occasion needed a maximum of one minute to read the body temperature includes the delay time. We found this result to be quick in comparison to the traditional manual technique.

Table 6.1 shows the decisions created by the system. SPO2 is standard for peripheral capillary oxygen saturation. These decisions will be sent to medical staff for approval. Medical staff are responsible for reviewing patient historical information based on their medication and their chronic diseases. As a result, they will decide if the current decisions are appropriated for the patients or if they need to change and update them. These results are based on commercial medical sensors. The performance can be improved by using separate body temperature sensors for every patient, and using real medical sensors, which do not require the delay technique for enhancing the accuracy.

Table 6.1: System Decisions



	Request_Id	Patient_Id	Status	Temp	SPO2	Decision
Edit Delete	44	10007	unapproval	37	83	You should visit the clinic soon.
Edit Delete	45	10008	unapproval	38	99	Please Enter your Decision. There is not historica
Edit Delete	53	10221	unapproval	37	96	
Edit Delete	54	10222	unapproval	37	96	Please Enter your Decision. There is not historical decisions are appropriate
Edit Delete	64	10223	unapproval	37	96	You are healthy and your test results seem normal
Edit Delete	67	28992	unapproval	39	90	Please Enter your Decision. There is not historical decisions are appropriate
Edit Delete	68	24563	unapproval	37	90	Your blood oxygen level is considered low but it is not necessarily indicative of a health issue. If you have chronic shortness of breath due to low oxygen, quit cigarettes if you smoke and avoid secondhand smoke, which can only make matters worse. Also try to get regular exercise.

1 2 3 4 5 6 7 8 9 10 ...

6.5. Summary

In this chapter, the design and implementation of the eHealth Smart System was described in order to present a case study of the proposed framework. We also described the decision making algorithm process. The decision making algorithm process is responsible for creating appropriate medical decisions based on three parameters, namely, patient id, sensor type, and sensor current data. Moreover, an encryption algorithm has been applied to provide cloud users with data security and privacy. The conclusion and future work will be presented in the next chapter.

Chapter 7

Conclusion and Future Work

In this chapter, we present a summary of the contributions of this thesis, and highlight research ideas for future research directions.

7.1. Contributions

The main contributions of this thesis are as follows:

- An approach has been presented for designing and implementing a framework for the integration of wireless sensors and cloud computing. This approach provides wireless sensor technology by high scalability and availability, as well as more security features. Moreover, storing sensor data in the cloud, which provides unlimited storage capability, can be an efficient solution for applying data analytic techniques.
- A decision making algorithm was designed and implemented based on real-time sensor data and historical sensor data.
- From perspective of security, we applied some security mechanisms to ensure that sensitive data had been transferred and stored securely.
- An eHealth system was designed and implemented and used as a case study that can be applied to serve healthcare communities. After the eHealth Smart System was implemented, we applied some experiments to evaluate the system's performance. We used the cloud infrastructure for building a testing environment,

and examined the performance of the Azure services, which included the Azure Load Balancer and the Traffic Manager.

7.2. Future Work

Many interesting research statements that are out of the scope of this thesis, remain open to research. Some of them are listed below:

- Many cloud providers have started to offer new services which can be used for storing, managing, and analyzing a large volume of data. For instance, Azure offers HDInsight, which provides Big Data analysis services that can be used to process unstructured and semi-structured data. Azure HDInsight has deployed and provisioned Apache Hadoop clusters in the cloud. It provides a software framework that is designed to manage and analyze the Big Data with high availability and reliability [54]. Thus, these new services need to be evaluated in terms of quality.
- Machine Learning (ML) tools that are offered by cloud providers can be a new approach to using cloud computing for data mining tasks. However, these tools need to be evaluated to ensure that cloud users can achieve high accuracy results when they use ML tools that run in the cloud. Data mining concepts and techniques can be applied to historical data in order to make high accurate decisions by the system. Also, wearable technology can be used for data collection.
- We implemented and evaluated the scalability at the application level. Scalability in the database tier should have been applied. Different types of testing could then have been applied to evaluate the database performance and examine the quality of services.

- Comparing the performance of different types of cloud storage offered by different cloud providers is a very important research area, as it can help cloud users in selecting a cloud provider that best fits their needs.
- Comparing the performance of storing sensor data using relational databases such as MS SQL Server, with non-relational databases, such as Document DB.
- From a security perspective, we applied two important mechanisms to ensure that the proposed framework is secure. However, many other techniques can be practically applied to this framework at the software level to increase security and privacy.

Appendix A : The Client Application in Raspberry Pi

In Raspberry Pi, the client application has been written in C++ programming languages using Socket and TCP/IP protocol. This application is responsible for collecting data from sensors and sending the data to the cloud.

```
// Client app
//Include eHealth library
#include "eHealth.h"
#include "arduPi.h"
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<string>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netdb.h>
#include <sstream>

using namespace std;

class tcp_client
{
private:
    int sock;
    std::string address;
    int port;
    struct sockaddr_in server;

public:
    tcp_client();
    bool conn(string, int);
    bool send_data(string data);
    string receive(int);
};

tcp_client::tcp_client()
{
    sock = -1;
    port = 0;
    address = "";
}


  // Connect to a server on using IP + port number

bool tcp_client::conn(string address , int port)
{
    //create new socket if it is not be created
    if(sock == -1)
    {

        sock = socket(AF_INET , SOCK_STREAM , 0);
        if (sock == -1)
        {
                perror("error happen when the applicaion try to create socket");
            }

            cout<<"Socket has been created\n";
        }
        else    {   cout<<"error \n"; }

        //setup address structure
        if(inet_addr(address.c_str()) == -1)
        {
            struct hostent *he;
            struct in_addr **addr_list;

            //resolve the hostname, its not an ip address
            if ( (he = gethostbyname( address.c_str() ) ) == NULL)
            {
                //gethostbyname failed
                herror("gethostbyname");
                cout<<"Failed to resolve server name\n";

                return false;
            }


            addr_list = (struct in_addr **) he->h_addr_list;

            for(int i = 0; addr_list[i] != NULL; i++)
            {

                server.sin_addr = *addr_list[i];

                cout<<address<<" resolved to "<<inet_ntoa(*addr_list[i])<<endl;

                break;
            }
        }

        //plain ip address
        else
        {
            server.sin_addr.s_addr = inet_addr( address.c_str() );
        }

        server.sin_family = AF_INET;
        server.sin_port = htons( port );

        //Connect to remote server
        if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
        {
            perror("connect to server is failed. Error");
            return 1;
        }

        cout<<"Connected\n";
        return true;
    }


       // Sending data to the connected host

```

```
bool tcp_client::send_data(string data)
{
    //Sending sensor data
    if( send(sock , data.c_str() , strlen( data.c_str() ) , 0) < 0)
    {
        perror("Sending is failed : ");
        return false;
    }
    cout<<"Data send\n";

    return true;
}


   // Receiveing data from the connected server

string tcp_client::receive(int size=512)
{
    char buffer[size];
    string reply;

    //Receive a reply from the server
    if( recv(sock , buffer , sizeof(buffer) , 0) < 0)
    {
        puts("recving failed");
    }

    reply = buffer;
    return reply;
}

std::string float_to_string(float f)
{
    std::ostringstream s;
    s << f;
    return s.str();
}



int main(int argc , char *argv[])
{
    tcp_client c;
    string host;
    string s0;

    c.conn("52.24.15.7", 1108);

    //send some data
    for (int i=0; i<40;i++)
    {
        float temperature = eHealth.getTemperature();
        s0 = float_to_string(temperature);
        printf("Temperature : %f \n", temperature);
        s0= s0+"$";

        delay(10000);
        c.send_data(s0);

        //Receiveing Data
        cout<<"**********************\n\n";
        cout<<c.ReceiveData(1024);
        cout<<"\n\n**********************\n\n";

    }

    //done
    return 0;
}
```

Appendix B : The Server Application in The Cloud

In the cloud side, the server application has been written in C# programming language; the server application is responsible for receiving sensor data and storing them in the cloud storage. Then, data analysis and the decision-making algorithm have been applied in the cloud[55].

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Text;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text.RegularExpressions;
using System.Data.SqlClient;
using System.Configuration;
using System.Data.Sql;
using System.Data;

public partial class ConnectToPi : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {


    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        // SocKet open connection

       string data;

        IPEndPoint ip = new IPEndPoint(IPAddress.Any, 999);
        Socket socket = new Socket(AddressFamily.InterNetwork,
    SocketType.Stream, ProtocolType.Tcp);

        socket.Bind(ip);
        socket.Listen(10);

        Socket client = socket.Accept();

        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;

        Label1.Text = "Connected";

        // Client Send Data
        // Create a System.Net.Sockets.NetworkStream from the above Socket:
        NetworkStream ns = new NetworkStream(client);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);

        string welcome = "Welcome";
        sw.WriteLine(welcome);
        sw.Flush();


        data = sr.ReadLine();
        txtSensorId.Text += data;

        data = sr.ReadLine();
        txtPatientId.Text = data;

        data = sr.ReadLine();
        txtTemp.Text = data;

        sw.Flush();
        socket.Close();
        Label1.Text = "Disconnected";

        SqlConnection conn = new SqlConnection(@"Data Source=ec2-54-191-23-
    134.us-west-2.compute.amazonaws.com\WIN-MJ8KG93S5MU,1433;
        Initial Catalog=FinalProject;Persist Security Info=True;User
    ID=sa;Password=sa_1234");

        DateTime now = DateTime.Now;

         System.Data.SqlClient.SqlCommand cmd = new
    System.Data.SqlClient.SqlCommand();
         cmd.CommandType = System.Data.CommandType.Text;

         cmd.CommandText = "INSERT FP_SENSORS_INFO (Sensor_Id, Patient_Id,
    Sensor_Data, Insert_Date) VALUES (@Sensor_Id, @Patient_Id, @Sensor_Data,
    @Insert_Date)";
         cmd.Parameters.AddWithValue("@Sensor_Id", txtSensorId.Text);
         cmd.Parameters.AddWithValue("@Patient_Id", txtPatientId.Text);
         cmd.Parameters.AddWithValue("@Sensor_Data", txtTemp.Text);
         cmd.Parameters.AddWithValue("Insert_Date", now);

         cmd.Connection = conn;

         conn.Open();
         cmd.ExecuteNonQuery();

        // decision making Proc start
         SqlCommand cmd2 = new SqlCommand("FP_SYSTEM_DECISION_SP", conn);

        //Specify that the SqlCommand is a stored procedure
         cmd2.CommandType = System.Data.CommandType.StoredProcedure;

       //Add the input parameters to the command object
         cmd2.Parameters.AddWithValue("@PatientId", txtPatientId.Text);
         cmd2.Parameters.AddWithValue("@SensorId", txtSensorId.Text);
         cmd2.Parameters.AddWithValue("@SensorData", txtTemp.Text);

         cmd2.ExecuteNonQuery();

        // close the connection
         conn.Close();
      sw.Close();
         sr.Close();

    }
}
```

Appendix C : Decision Making Algorithm Code

We implemented the decision making algorithm using SQL Server Stored Procedure. We can define stored procedure as a group of SQL statement which are grouped to perform a certain task.

```
USE [FinalProject]
GO
/****** Object:  StoredProcedure [dbo].[SYSTEM_DECISION]    Script Date: 21/07/2015
    10:56:06 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[SYSTEM_DECISION]
     (
    @PatientId int,
    @SensorId varchar(10),
    @Temp_SensorData  float,
    @SPO2_Sensor int,
    @Gluo_Sensor int
     )
AS
begin
declare @maxseq     int,
@Decision_Desc     varchar(max),
@result     int, @Sensor_Typed  varchar(30),@Spo2_result int, @Gluo_result int,
    @NormalPercent int

-- Check data normal or abnormal based on ranges of laboratory medical tests and
    patients medical policies
-- Start

set @Spo2_result= (Select count(1) from  FP_Sensors_Policies where Patient_Id=
    @PatientId
And Sensor_typed='SPO2' and @SPO2_Sensor between [from] and [to])

set @Gluo_result= (Select count(1) from  FP_Sensors_Policies where Patient_Id=
    @PatientId
And Sensor_typed='Gluo' and @Gluo_Sensor between [from] and [to])

 set @result  = (SELECT     COUNT(1)
      --into       @result
FROM            FP_Sensors_Policies
    WHERE Patient_Id = @PatientId
    and   @Temp_SensorData between [from] and [to] -- or between Normal Range between
    [37] and [38]
    and  @Spo2_result=1
    and  @Gluo_result=1)


set @Sensor_Typed = (SELECT Sensor_Typed FROM FP_Sensors_Policies
    WHERE Patient_Id = @PatientId
    AND   Sensor_Id = @SensorId)

    --Normal sensor data based patients medical policies
    if (@result > 0)
    begin
    set @Decision_Desc = ('You are healthy and your test results seem normal')
    end

    if (@result = 0) -- End the Check if Data Norm or Abnor
    begin
--if (@Sensor_Typed like 'Temp')
--begin

set @maxseq = (SELECT MAX (Request_Id)
FROM FP_Patient_Requests_Inf WHERE Patient_Id = @PatientId
    --AND   round(Temp_Sensor,2,1) = round(@Temp_SensorData,2,1)
    AND Temp_Sensor between @Temp_SensorData and (@Temp_SensorData + 1)
    AND SPO2_Sensor between @SPO2_Sensor and (@SPO2_Sensor + 5)
    AND Gluo_Sensor between @Gluo_Sensor and (@Gluo_Sensor + 10)
    AND   lower(Request_Status) = 'approval')

set @Decision_Desc = (SELECT    Request_Dec
    FROM  FP_Patient_Requests_Inf
    WHERE Patient_Id = @PatientId
    AND   Temp_Sensor between @Temp_SensorData and (@Temp_SensorData + 1)
    AND SPO2_Sensor between @SPO2_Sensor and (@SPO2_Sensor + 5)
    AND Gluo_Sensor between @Gluo_Sensor and (@Gluo_Sensor + 10)
    AND   lower(Request_Status) = 'approval'
    AND   Request_Id = @maxseq)
    end

     -- Check sensor data If Temp_Sensor > 37.5 {print temp is high}
     -- else (Print temp is normal) If (SPO2 between 60, 90) else ....
    if (@Decision_Desc is null)
    begin
    set @maxseq = (SELECT MAX (Request_Id) FROM FP_Patient_Requests_Inf WHERE
    --AND   round(Temp_Sensor,2,1) = round(@Temp_SensorData,2,1)
        Temp_Sensor between @Temp_SensorData and (@Temp_SensorData + 1)
    AND SPO2_Sensor between @SPO2_Sensor and (@SPO2_Sensor + 5)
    AND Gluo_Sensor between @Gluo_Sensor and (@Gluo_Sensor + 10)
    AND   lower(Request_Status) = 'approval')

set @Decision_Desc = (SELECT    Request_Dec
    FROM  FP_Patient_Requests_Inf
    WHERE
    Temp_Sensor between @Temp_SensorData and (@Temp_SensorData + 1)
    AND SPO2_Sensor between @SPO2_Sensor and (@SPO2_Sensor + 5)
    AND Gluo_Sensor between @Gluo_Sensor and (@Gluo_Sensor + 10)
    AND   lower(Request_Status) = 'approval'
    AND   Request_Id = @maxseq)
    end

    -- Normal data for those don't have police records because they are first time
using   the   system
    if ((@Decision_Desc is null) and (@Temp_SensorData between 37 and 37.5)
    and (@SPO2_Sensor between 95 and 100) and (@Gluo_Sensor between 100 and 120) )
    begin
    set @Decision_Desc = ('You are healthy and your test results seem normal')
    end

    -- Check sensor data If Temp_Sensor > 37.5 {print temp is high} else (Print temp
is      normal)
    -- If (SPO2 between 60, 90) else ....
    if (@Decision_Desc is null)
    begin
    set @Decision_Desc = ('Please Enter your Decision. There is not historical
decisions      are appropriate')
    end

    set @NormalPercent = (@Spo2_result + @Gluo_result)
    if (@NormalPercent>=2)
    begin
    set @NormalPercent= (100)
    end
    else if (@NormalPercent<=1)
    begin
    set @NormalPercent= (50)
    end

    update FP_Patient_Requests_Inf
    SET Request_Dec= @Decision_Desc
    WHERE Request_Id=(SELECT MAX (Request_Id)
    FROM FP_Patient_Requests_Inf WHERE Patient_Id= @PatientId);
END
```

Appendix D : Applying Hashing Technique

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.Sql;
using System.Data;
using System.Data.SqlClient;
using System.Collections;
using System.Configuration;
using System.Web.Security;


public partial class login : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        string hashresult2 =
    FormsAuthentication.HashPasswordForStoringInConfigFile(txtPass.Text,
    "SHA1");

        SqlConnection conn = new SqlConnection(@"Data Source=ec2-52-24-15-
7.us-west-2.compute.amazonaws.com\WIN-MJ8KG93S5MU,1433;Initial
    Catalog=FinalProject;Persist Security Info=True;User
    ID=sa;Password=sa_1234");

        conn.Open();
        string checkuser = "select count(*) from Users where User_Id= '" +
    txtUsername.Text + "' and password= '" + hashresult2 + "'";
        string checktype = "select type from Users where User_Id= '" +
    txtUsername.Text + "'";
        string checkName = "select User_Name from Users where User_Id= '" +
    txtUsername.Text + "'";

        SqlCommand com = new SqlCommand(checkuser, conn);
        SqlCommand com2 = new SqlCommand(checktype, conn);
        SqlCommand com3 = new SqlCommand(checkName, conn);


        string temp = com.ExecuteScalar().ToString();
        string type = com2.ExecuteScalar().ToString();
        string Name = com3.ExecuteScalar().ToString();
        conn.Close();


        if (temp == "1")
        {
            if (type == "Patient")
            {
                //Response.Write("password is not correct");
                Session["user"] = txtUsername.Text;
                Session["User_Name"] = Name;
                Session["User_Type"] = type;
                Response.Redirect("test.aspx");
            }
            else
            Session["user"] = txtUsername.Text;
            Session["User_Name"] = Name;
            Session["User_Type"] = type;
            Response.Redirect("test.aspx");
        }
        else
        {
            Label4.Text = "Username is not correct";
        }
    }
}
```

Appendix E : Implementation of Encryption Technique

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;
using System.Net;
using System.Text.RegularExpressions;
using System.Collections;
using System.Security.Cryptography;
using System.Text;

public partial class WordCount : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void UploadButton_Click(object sender, EventArgs e)
    {
        if (FileUpload1.HasFile)
        {
            // Add code to upload file with encryption
            byte[] file = new byte[FileUpload1.PostedFile.ContentLength];
            FileUpload1.PostedFile.InputStream.Read(file, 0,
FileUpload1.PostedFile.ContentLength);

            string fileName = FileUpload1.PostedFile.FileName;

            // key for encryption
            byte[] Key = Encoding.UTF8.GetBytes("asdf!@#$1234ASDF");
            try
            {
                string outputFile =
Path.Combine(Server.MapPath("~/UploadedFiles"), fileName);
                if (File.Exists(outputFile))
                {
                    // Show Already exist Message
                }
                else
                {
                    FileStream fs = new FileStream(outputFile,
FileMode.Create);
                    RijndaelManaged rmCryp = new RijndaelManaged();
                    CryptoStream cs = new CryptoStream(fs,
rmCryp.CreateEncryptor(Key, Key), CryptoStreamMode.Write);
                    foreach (int data in file)
                    {
                        cs.WriteByte((byte)data);
                    }
                    cs.Close();
                    fs.Close();
                }
                StatusLabel.Text = "Upload status: File uploaded!";
                string filename = Path.GetFileName(FileUpload1.FileName);
                Label3.Text = filename;
                //PopulateUploadedFiles();
            }
            catch
```

```
                {
                    Response.Write("Encryption Failed! Please try again.");
                }

            }
        }
}
```

Appendix F : Generating Random Sensor Data From Client Side

In the Raspberry Pi (Client Side), we create an application written in C++ to generate a large number of requests based on random sensor data.

```
// Adding eHealth library
#include "eHealth.h"
#include "arduPi.h"
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<string>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netdb.h>
#include <sstream>

using namespace std;


  // TCP connection Client class

class tcpconn_client
{
private:
    int sock;
    std::string IpAddress;
    int PortNo;
    struct sockaddr_in server;

public:
    tcpconn_client();
    bool conn(string, int);
    bool SendData(string SensorData);
    string ReceiveData(int);
};

tcpconn_client::tcpconn_client()
{
    sock = -1;
    PortNo = 0;
    IpAddress = "";
}


    // connect a client to a cloud server using Ip + port number

bool tcpconn_client::conn(string IpAddress , int PortNo)
{
    //create new socket if it is not be created
    if(sock == -1)
    {
        //Create a new socket
        sock = socket(AF_INET , SOCK_STREAM , 0);
        if (sock == -1)
        {
            perror(" error happen when the applicaion try to create
socket");
        }

        cout<<"socket has been created\n";
    }
    else    {   cout<<"error \n"; }

    //setup Ip Address structure
    if(inet_addr(IpAddress.c_str()) == -1)
    {
        struct hostent *he;
        struct in_addr **addr_list;

        //check a server name (True ip or not)
        if ( (he = gethostbyname( IpAddress.c_str() ) ) == NULL)
        {
            //If the function is failed, the next error will be presented
            herror("gethostbyname");
            cout<<"Failed to resolve server name\n";

            return false;
        }


        addr_list = (struct in_addr **) he->h_addr_list;

        for(int i = 0; addr_list[i] != NULL; i++)
        {
            //strcpy(ip , inet_ntoa(*addr_list[i]) );
            server.sin_addr = *addr_list[i];

            cout<<IpAddress<<" resolved to 
"<<inet_ntoa(*addr_list[i])<<endl;

            break;
        }
    }

    //plain ip IpAddress
    else
    {
        server.sin_addr.s_addr = inet_addr( IpAddress.c_str() );
    }

    server.sin_family = AF_INET;
    server.sin_port = htons( PortNo );

    //connect to a server
    if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        perror("Error connect failed. ");
        return 1;
    }

    cout<<"The client are connected to the server\n";
    return true;
}


   // After the client has been connnected to cloud server, data will be 
sent

    bool tcpconn_client::SendData(string SensorData)
{
    //Send Sensor data to the cloud
    if( send(sock , SensorData.c_str() , strlen( SensorData.c_str() ) , 0) <
0)
    {
        perror("Sending data is failed: ");
        return false;
    }
    cout<<"Client sent Data to the cloud \n";

    return true;
}


  //receiveing data from the cloud

string tcpconn_client::ReceiveData(int size=512)
{
    char Rbuffer[size];
    string ReplyFromServer;

    // the cloud server sends reply to the client
    if( recv(sock , Rbuffer , sizeof(Rbuffer) , 0) < 0)
    {
        puts("ReceiveData from server is failed");
    }

    ReplyFromServer = Rbuffer;
    return ReplyFromServer;
}

int sys_random(int min, int max) {
srand(static_cast<unsigned int>(time(0)));
  return (rand() % (max - min+1) + min);
}

float RandomFloat(float min, float max)
{
srand(static_cast<unsigned int>(time(0)));
float r = (float)rand() / (float)RAND_MAX;
return min + r * (max - min);
}


std::string float_to_string(float f)
{
    std::ostringstream s;
    s << f;
    return s.str();
}

std::string NumberToString ( int num )
 {
     std::ostringstream ss;
     ss << num;
     return ss.str();
 }



 int main(int argc , char *argv[])
{
    tcpconn_client c;
    string host;

string s0;
string s1;
string s2;

int patientid;
float patient_Tem;
int Spo2;

   cout<<"Enter hostname : ";
   cin>>host;

   //connect to host
   c.conn(host, 1109);

   //send some data
for (int i=0; i<1;i++)
{
patientid= sys_random(1000,4000);
s0= NumberToString (patientid);
//cout<< patientid << endl;

patient_Tem= RandomFloat(36,42);
//cout<< patient_Tem << endl;
//float temperature = eHealth.getTemperature();
s1 = float_to_string(patient_Tem);


Spo2= sys_random(80,99);
s2= NumberToString (Spo2);
//cout<< Spo2 << endl;

s0= s0+s1+s2+"$";
//str.append(str2);
//delay(10000);
c.SendData(s0);

//ReceiveData
   cout<<"**********************\n\n";
   cout<<c.ReceiveData(1024);
   cout<<"\n\n***********************\n\n";

}

   //done
   return 0;
}
```

Appendix G: Cloud Server Application

This section presents the source code of Cloud server application which has been used for receiving sensor data from Raspberry Pi; then, storing sensor data in the cloud storage. The execution time is accounted to evaluate the performance[55].

```
using System.Threading.Tasks;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text.RegularExpressions;
using System.Data.SqlClient;
using System.Configuration;
using System.Data.Sql;
using System.Data;

namespace ConsoleApplication13
{
    class Program
    {
        static void Main(string[] args)
        {
            TcpListener serverSocket = new TcpListener(1109);
            int requestCount = 0;
            TcpClient clientSocket = default(TcpClient);
            serverSocket.Start();
            Console.WriteLine(" >> Server Started");
            clientSocket = serverSocket.AcceptTcpClient();
            Console.WriteLine(" >> Accept connection from client");
            requestCount = 0;

            Console.WriteLine(" Data from client");
            Console.WriteLine("Patient Id......................Temp..............SPO2");

            TimeSpan ts = (DateTime.UtcNow - new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc));
            long millis = (long)ts.TotalMilliseconds;

            while ((true))
            {

                try
                {
                    requestCount = requestCount + 1;
                    NetworkStream networkStream = clientSocket.GetStream();
                    byte[] bytesFrom = new byte[10000];
                    networkStream.Read(bytesFrom, 0, (int)clientSocket.ReceiveBufferSize);
                    string dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
                    string dataFromClient1;
                    string PatientId = dataFromClient.Substring(0, 4);
                    string Patient_Temp = dataFromClient.Substring(4, 7);
                    string Patient_Spo2 = dataFromClient.Substring(11, 2);
                string Check_Data = dataFromClient.Substring(0, 5);
                string Checker = "\0\0\0\0\0";


                    int j = 0;
                    SqlConnection conn = new SqlConnection(@"Data Source=ec2-52-24-15-7.us-west-2.compute.amazonaws.com\WIN-MJ8KG93S5MU,1433;Initial Catalog=FinalProject;Persist Security Info=True;User ID=sa;Password=sa_1234");
                    conn.Open();

                    if (Check_Data != Checker)
                    {

                        DateTime now = DateTime.Now;
                        System.Data.SqlClient.SqlCommand cmd = new System.Data.SqlClient.SqlCommand();
                        cmd.CommandType = System.Data.CommandType.Text;
                        cmd.CommandTimeout = 0;

                        cmd.CommandText = "INSERT Temp (PatientId, Patient_Temp, Patient_Spo2,Insert_Date) VALUES (@PatientId, @Patient_Temp,@Patient_Spo2, @Insert_Date)";
                        cmd.Parameters.AddWithValue("@PatientId", PatientId);
cmd.Parameters.AddWithValue("@Patient_Temp", Patient_Temp);
cmd.Parameters.AddWithValue("@Patient_Spo2", Patient_Spo2);
                        cmd.Parameters.AddWithValue("Insert_Date", now);

                        cmd.Connection = conn;
                        cmd.ExecuteNonQuery();


                        Console.WriteLine(PatientId + "..................." + Patient_Temp + "..........." + Patient_Spo2);

                      string reply = "recived recored from client--> Patient id=" + PatientId;

                        Byte[] sendBytes1 = Encoding.ASCII.GetBytes(reply);
                        networkStream.Write(sendBytes1, 0, sendBytes1.Length);
                        networkStream.Flush();

                    }
                    else break;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
        }

        TimeSpan ts2 = (DateTime.UtcNow - new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc));
        long millis2 = (long)ts2.TotalMilliseconds;
```

```
                    long elapsed = millis2 - millis;
                    String x = String.Format("{0:#,##0}{1}", elapsed, "
millis");
                    Console.WriteLine("The execution time for receiving and
                    storing data in the cloud for 100 Patients is {0}:",x);

                    clientSocket.Close();
                    serverSocket.Stop();
                    Console.WriteLine(" >> exit");
                    Console.ReadLine();


                }
            }
        }
```

Bibliography

[1] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, Apr. 2010.

[2] H. Takabi, J. B. D. Joshi, and G.-J. Ahn, “Security and Privacy Challenges in Cloud Computing Environments,” *IEEE Secur. Priv. Mag.*, vol. 8, no. 6, pp. 24–31, Nov. 2010.

[3] D. Zissis and D. Lekkas, “Addressing cloud computing security issues,” *Futur. Gener. Comput. Syst.*, vol. 28, no. 3, pp. 583–592, 2012.

[4] M. S. Jassas, A. A. Qasem, and Q. H. Mahmoud, “A smart system connecting e-health sensors and the cloud,” in *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2015, pp. 712–716.

[5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey,” *Comput. Networks*, vol. 38, no. 4, pp. 393–422, Mar. 2002.

[6] J. Yicka, B. Mukherjeea, and D. Ghosal, “Wireless sensor network survey,” *Comput. Networks*, vol. 58, no. 12, pp. 2292–2330, 2008.

[7] G. Simon, M. Maróti, Á. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, “Sensor network-based countersniper system,” Association for Computing Machinery, 2004.

[8] S. K. Dash, J. P. Sahoo, S. Mohapatra, and S. P. Pati, “Sensor-Cloud: Assimilation of wireless sensor network and the cloud,” in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 2012, vol. 84, no. PART 1, pp. 455–464.

[9] M. Castillo-Effer, D. H. Quintela, W. Moreno, R. Jordan, and W. Westhoff, “Wireless sensor networks for flash-flood alerting,” *Proc. Fifth IEEE Int. Caracas Conf. Devices, Circuits Syst. 2004.*, vol. 1, 2004.

[10] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees, “Deploying a wireless sensor network on an active volcano,” *IEEE Internet Comput.*, vol. 10, no. 2, pp. 18–25, 2006.

[11] B. Swathi and H. Guruprasad, “Integration of Wireless Sensor Networks and Cloud Computing,” *Int. J. Comput.*, 2014.

[12] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, “A view of cloud computing,” *Commun. ACM*, vol. 53, no. 4, p. 50, Apr. 2010.

[13] R. Buyya, R. Buyya, C. S. Yeo, C. S. Yeo, S. Venugopal, S. Venugopal, J. Broberg, J. Broberg, I. Brandic, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Futur. Gener. Comput. Syst.*, vol. 25, no. 6, p. 17, 2009.

[14] P. Mell and T. Grance, “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,” *Nist Spec. Publ.*, vol. 145, p. 7, 2011.

[15] “Heroku | Cloud Application Platform.” [Online]. Available: https://www.heroku.com/. [Accessed: 01-Sep-2015].

[16] “Google App Engine: Platform as a Service - App Engine — Google Cloud Platform.” [Online]. Available: https://cloud.google.com/appengine/docs. [Accessed: 01-Sep-2015].

[17] “Amazon Web Services (AWS) - Cloud Computing Services.” [Online]. Available: https://aws.amazon.com/. [Accessed: 01-Sep-2015].

[18] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A Survey of Mobile Cloud Computing : Architecture , Applications , and Approaches,” *Computer (Long. Beach. Calif).*, no. Cc, pp. 1–38, 2011.

[19] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, “Dynamically scaling applications in the cloud,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, p. 45, Jan. 2011.

[20] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, “Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment,” in *2009 IEEE International Conference on e-Business Engineering*, 2009, pp. 281–286.

[21] C. Zenon, M. Venkatesh, and A. Shahrzad, “Availability and Load Balancing in Cloud Computing,” *Int. Conf. Comput. Softw. Model. IPCSIT vol.14 IACSIT Press. Singapore*, vol. 14, pp. 134–140, 2011.

[22] L. M. Kaufman, “Data Security in the World of Cloud Computing,” *IEEE Secur. Priv. Mag.*, vol. 7, no. 4, pp. 61–64, Jul. 2009.

[23] D. G. FENG, M. ZHANG, Y. ZHANG, and Z. XU, “Study on Cloud Computing Security,” *J. Softw.*, vol. 22, no. 1, pp. 71–83, Mar. 2011.

[24] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[25] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Comput. Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[26] K. Ashton, “That ‘Internet of Things’ Thing,” *RFiD J.*, 2009.

[27] M. Weiser, R. Gold, J. S. Brown, B. Sprague, and R. Bruce, “The origins of ubiquitous computing research at PARC,” *IBM Syst. J.*, vol. 38, no. 4, pp. 693–696, 1999.

[28] G. Fortino, M. Pathan, and G. Di Fatta, “BodyCloud: Integration of Cloud Computing and body sensor networks,” in *CloudCom 2012 - Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, 2012, pp. 851–856.

[29] B. B. P. Rao, P. Saluia, N. Sharma, A. Mittal, and S. V Sharma, “Cloud computing for Internet of Things & sensing based applications,” in *Sensing Technology (ICST), 2012 Sixth International Conference on*, 2012, pp. 374–380.

[30] H. M. Perumal, B., Rajasekaran, M., & Ramalingam, “WSN Integrated Cloud For Automated Telemedicine (ATM) Based E-Healthcare Applications,” *International Proceedings of Chemical, Biological & Environmental Engineering, 29*, 2012. [Online]. Available: http://www.ipcbee.com/vol29/30-ICBBT2012-H10011.pdf. [Accessed: 11-Aug-2015].

[31] S. R. Vemuri, N. Satyanarayana, and V. L. Prasanna, “Generic Integrated Secured WSN-Cloud Computing U-life care,” *Int. J. Eng. Sci. Adv. Technol.*, no. 4, pp. 897–907, 2012.

[32] W. Chung, P. Yu, and C. Huang, “Cloud computing system based on wireless sensor network,” *Comput. Sci. Inf. Syst.*, pp. 877–880, 2013.

[33] A. Lounis, A. Hadjidj, A. Bouabdallah, and Y. Challal, “Secure and scalable cloud-based architecture for e-Health wireless sensor networks,” in *2012 21st International Conference on Computer Communications and Networks, ICCCN 2012 - Proceedings*, 2012.

[34] K. Ahmed and M. Gregory, “Integrating Wireless Sensor Networks with Cloud Computing,” *2011 Seventh Int. Conf. Mob. Ad-hoc Sens. Networks*, pp. 364–366, Dec. 2011.

[35] K. L. Tan, “What’s NExT?: Sensor + Cloud!?,” *the 7th International Workshop on Data Management for Sensor Networks*, 2010. [Online]. Available:

http://www.vldb2010.org/proceedings/files/vldb_2010_workshop/DMSN_2010/individual-files/01.keynote.pdf. [Accessed: 11-Aug-2015].

[36] C. H. Yun, H. Han, H. S. Jung, H. Y. Yeom, and Y. W. Lee, "Intelligent management of remote facilities through a ubiquitous cloud middleware," in *CLOUD 2009 - 2009 IEEE International Conference on Cloud Computing*, 2009, pp. 65–71.

[37] C. O. Rolim, F. L. Koch, C. B. Westphall, J. Werner, A. Fracalossi, and G. S. Salvador, "A Cloud Computing Solution for Patient's Data Collection in Health Care Institutions," *2010 Second Int. Conf. eHealth, Telemedicine, Soc. Med.*, no. ii, pp. 95–99, Feb. 2010.

[38] B. Adler and S. Architect, "Load balancing in the cloud: Tools, tips and techniques," 2012.

[39] W. Lijun and H. Yongfeng, "Medoop: A medical information platform based on Hadoop," *2013 IEEE 15th Int. Conf. e-Health Networking, Appl. Serv. (Healthcom 2013)*, pp. 1–6, Oct. 2013.

[40] C. M. Huang, H. H. Ku, and Y. W. Chen, "Design and implementation of a web 2.0 service platform for DPWS-based home-appliances in the cloud environment," in *Proceedings - 25th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2011*, 2011, pp. 159–163.

[41] C. H. Lu, H. H. Kuo, C. W. Hsiao, Y. L. Ho, Y. H. Lin, and H.-P. Ma, "Localization with WLAN on smartphones in hospitals," in *2013 IEEE 15th International Conference on e-Health Networking, Applications and Services (Healthcom 2013)*, 2013, pp. 534–538.

[42] J. J. Hwang, H. K. Chuang, C. H. Wu, and Y. C. Hsu, "A business model for cloud computing based on a separate encryption and decryption service," in *2011 International Conference on Information Science and Applications, ICISA 2011*, 2011.

[43] I. L'm, S. Szebeni, and L. Butty'n, "Tresorium: Cryptographic File System for Dynamic Groups over Untrusted Cloud Storage," *2012 41st Int. Conf. Parallel Process. Work.*, pp. 296–303, Sep. 2012.

[44] T. Erl, R. Puttini, and Z. Mahmood, *Cloud Computing: Concepts, Technology, & Architecture*. 2013.

[45] "Raspberry Pi - Teach, Learn, and Make with Raspberry Pi." [Online]. Available: https://www.raspberrypi.org/. [Accessed: 17-Aug-2015].

[46] "Microsoft Azure: Cloud Computing Platform & Services." [Online]. Available: http://azure.microsoft.com/en-us/. [Accessed: 11-Aug-2015].

[47] K. Davies, "Virtual machine sizes," *Microsoft Corporation,* 2015. [Online]. Available: https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-size-specs/. [Accessed: 11-Aug-2015].

[48] J. Madureira, "Traffic Manager Overview," 2015. [Online]. Available: https://azure.microsoft.com/en-us/documentation/articles/traffic-manager-overview/. [Accessed: 11-Aug-2015].

[49] J. Madureira, "Manage Traffic Manager profiles." [Online]. Available: https://azure.microsoft.com/en-us/documentation/articles/traffic-manager-manage-profiles/. [Accessed: 11-Aug-2015].

[50] "Apache JMeter - Apache JMeter^{TM}," *The Appach Software Foundation*. [Online]. Available: http://jmeter.apache.org/. [Accessed: 11-Aug-2015].

[51] "e-Health Sensor Platform Complete Kit V2.0 for Arduino, Raspberry Pi and Intel Galileo [Biometric / Medical Applications]," *cooking hacks*. [Online]. Available: https://www.cooking-hacks.com/ehealth-sensors-complete-kit-biometric-medical-arduino-raspberry-pi. [Accessed: 11-Aug-2015].

[52] "Create a Stored Procedure," *Microsoft Corporation*. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms345415.aspx. [Accessed: 11-Aug-2015].

[53] "UCSF Departments of Pathology and Laboratory Medicine | SFGH Lab Manual | Reference Ranges & Critical Test Values." [Online]. Available: http://labmed.ucsf.edu/sfghlab/test/ReferenceRanges.html. [Accessed: 11-Aug-2015].

[54] "What is Hadoop in HDInsight: Cloud big data analysis | Microsoft Azure." [Online]. Available: https://azure.microsoft.com/en-in/documentation/articles/hdinsight-hadoop-introduction/. [Accessed: 04-Sep-2015].

[55] "C# Server Socket program - Print Source Code." [Online]. Available: http://csharp.net-informations.com/communications/files/print/csharp-server-socket_print.htm. [Accessed: 12-Oct-2015].