# Database-as-a-Service Integration with Ad hoc Mobile Cloud-powered GUISET

by

**Siphelele C. Fakude**

**(201066002)**

A dissertation submitted in fulfilment of the requirements for the degree of

**Master of Science in Computer Science**

to

Department of Computer Science

Faculty of Science and Agriculture

University of Zululand

KwaDlangezwa

RSA

**Supervisor: Ms IN Mba**

**Co- Supervisor: Prof M.O Adigun**

2017

# DECLARATION

I, Siphelele C. Fakude declare that the work contained in this dissertation has not been previously submitted in whole or in part, to meet requirements for any other degree or professional qualification at this or any other higher education institution. To the best of my knowledge and belief, the dissertation contains no material previously published or written by another person except where due reference is made.

Signature: _____

# DEDICATION

To my family

# ACKNOWLEDGEMENTS

I would like to submissively thank God Almighty. Without His presence in my life, none of this would have been possible. I would also like to sincerely thank Prof M.O Adigun for his endless support and guidance throughout this study. Thank you for believing in me when I could not believe in myself

My special thanks is extended to my supervisor Ms IN Mba, for her great contributions towards this work. Not only were you my supervisor, but also a sister and a friend every person would wish for. My gratitude goes to Mr O. Oki and Mr P.Mudali. Your most valuable comments made this work what it is today. To my colleagues Tosin, Dominic and Kudzai, thank you for enlightening my burden throughout this journey with your endless support and fun personalities.

To all the CoE members, this work would not have been a success without you

Finally, I would like to thank Telkom for sponsoring me throughout this study

# TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

## LIST OF ACRONYMS

AMC – Ad hoc Mobile Cloud Computing

CC – Cloud Computing

DBaaS – Database as a Service

GUISET – Grid-based Utility Infrastructure for SMME Enabling Technology

HTTP – Hyper Text Transfer Protocol

IaaS – Infrastructure as a Service

LRU – Least Recently Used

LFU – Least Frequently Used

MCC – Mobile Cloud Computing

PaaS – Platform as a Service

POJO – Plain Old Java Object

RDBMS – Relational Database Management System

REST – Represential State Transfer

RPC – Remote Procedure Call

SaaS – Software as a Service

SMME – Small, Medium and Micro Enterprises

SOAP – Simple Object Access Protocol

UDDI – Universal Description, Discovery and Integration

URI – Universal Resource Identifier

WSDL- Web Service Description Language

XML – eXtensible Markup Language

# ABSTRACT

The World Wide Web has advanced considerably in the recent years. These advancements have also been expanded to the mobile environments, where mobile devices are now capable of doing most of the operations that were initially intended for traditional computers. Due to the resource poverty situation facing mobile devices, i.e., mobile devices having limited storage capacity, processing and battery power, these advanced capabilities come at a cost. Because of this challenge, computing paradigms such as Mobile Cloud Computing (MCC) have been introduced in an attempt to mitigate this challenge. With MCC, most of the processing and storage is moved from the mobile device to the cloud server, thus conserving the device's resources. For this to be realisable, the presence of an Internet link between the mobile device and the cloud is required. However, due to the fluctuation of wireless links, the always-on Internet connectivity between the cloud and mobile devices is not achievable at all times. The Internet disconnections result in cloud resources being unreachable by the mobile users, which has a negative impact on most applications.

In an attempt to mitigate the connectivity outage issue between mobile devices and the cloud, this work proposes the usage of a cache to maintain data availability to mobile clients who were initially using the DBaaS model to store their data. The proposed cache model is used only during Internet disconnections, and uses a prefetching mechanism to periodically retrieve its data from the central cloud. The usability of this model applies in Ad hoc Mobile Clouds (AMC), where a group of disconnected mobile devices collaborate to form a community cloud. Furthermore, the integration of an AMC with the Grid-based Utility Infrastructure for SMME Enabling Technology (GUISET) to achieve smooth running of offline clients is proposed. This work further develops a prototype of the model, and the evaluation results show that our model performs better in terms of maintaining data availability to occasionally disconnected clients.

# CHAPTER ONE

# INTRODUCTION

## 1.1 Preamble

The substantial improvement of mobile devices in relation to hardware and functionality has been observed over the years. Their ability to network wirelessly forms the basis of mobile computing (Forman & Zahorjan 1994). However, mobile devices have limitations in their performance capabilities due to their limited memory, processing power, battery life and unstable wireless connections (Satyanarayanan 1996; Gupta 2008). They have been declared to "always be resource poor" by (Kovachev et al. 2011).

Cloud Computing is a model that enables the provisioning of a range of services provided by an internet-based cluster system (Dev & Baishnab 2014). These cluster systems are made up of a group of personal computers or low-cost servers, which provide reliable, secure and transparent services such as data storage and computing to clients (Dinh et al. 2011). Cloud computing allows users to use infrastructure such as servers, storages, software and platforms that are provided by cloud providers at very low cost and in an on-demand fashion (Dev & Baishnab 2014). With the resource poverty situation experienced by mobile devices, a new computing paradigm called Mobile Cloud Computing (MCC), was established (Kovachev et al. 2011).

MCC is the integration of mobile devices with the traditional cloud computing architecture. It seeks to address the resource poverty situation faced by mobile devices by offloading most of their processing and storage to the cloud (Fernando et al. 2013). Though MCC eases the functioning of mobile devices, the inability to access the central cloud due to unreliable Internet connections is one of this technology's key challenges (Dinh et al. 2011), as it results

in inaccessible cloud resources. In attempting to resolve this issue, Ad hoc Mobile Cloud computing was proposed (Kovachev et al. 2010).

The Ad hoc mobile cloud presents a scenario where mobile devices in a community serve as cloud providers by exposing their computing resources to each other. It is particularly in cases of weak or no Internet connections, allowing devices to discover and make use of web services hosted by other devices within the network. Moreover, it allows the creation of computing communities where the mobile devices can collaborate in executing shared tasks (Kovachev et al. 2011). One of the main purposes of ad hoc mobile cloud computing is for end users to be able to further perform computations whenever the connection to the cloud is down. Other flexibilities of this technology include the ability to create virtual mobile cloud providers, which avoids a connection to infrastructure-based cloud providers while maintaining the offloading benefits.

Provisioning of database-as-a-service (DBaaS) is one paradigm that the cloud uses for data management. It is an architectural approach that enables IT providers to provide database functionality as a service to consumers (Oracle 2011). Traditionally, DBaaS provides users with the mechanism that offers functionalities enabling them to create, store and access their databases at the host site, while, at the same time, eliminating the need for enterprises to buy expensive software and hardware, to worry about the software updates and to hire the staff for administrative tasks (Hakan Hacıg¨um ¨us & Mehrotra 2002; Forrester Consulting 2012).

There are various DBaaS implementations such as Amazon RDS, Microsoft SQL Azure, Google AppEngine Datastore and Amazon SimpleDB, which are best classified by (Pizette & Cabot 2012), and one notable scenario where the DBaaS model is used to provide e-Health records (Lin et al. 2014). One desirable capability of DBaaS is for it to tolerate faults, such as connectivity outage, in such a way that data would still be available and accessible even

though the connectivity to the cloud is down. Studies have been conducted on the fault tolerance aspect of DBaaS, one of them shown in (Zhang & Chen 2011), where different approaches to the design of durable data cloud platforms were investigated, with data loss being the fault to be tolerated. More practical DBaaS implementations such as Microsoft SQL Azure (Microsoft 2012) cater for faults including hardware and software failure, but little has been done to cater for the connectivity outage fault.

This study, therefore, aims at proposing a cache model that will ensure data availability in cases of weak or no Internet connectivity between mobile devices and the cloud that result in the inaccessibility of the cloud database. This cache will be implemented in an ad hoc mobile cloud scenario and will be provided through the service composition broker called GUISET, which is an enabling technology for inexpensive web services to SMMEs (Ekabua & Adigun 2010; Sihawu et al. 2013).The GUISET infrastructure works on pay-per-use terms, allowing the SMMEs to pay for only what they use. The pay-per-use capability of GUISET infrastructure, in some cases, would result in almost cost-free usage of the services, which will be of great advantage to the small, financially unstable enterprises.

The provisioning of GUISET services through an ad hoc mobile cloud environment to support disconnected clients forms an ad hoc mobile cloud-powered GUISET environment, which enables continual access to services despite connectivity state. This may result in disconnection transparency to users, as smooth running of clients is maintained. Further integrating the model that this study proposes with this environment will ensure data availability to disconnected clients at reduced monetary and resource costs due to the following reasons:

I. GUISET strives towards developing financially disadvantaged enterprises technologically, thus, its services are provided at low monetary costs

II. Accessing a cache storage conserves the client device's resources such as battery and computation power as it is maintained locally, whether within a device or a network

Hence, as aforementioned, this study will highly favour SMMEs that are challenged by their mobility profile which results in frequent Internet disconnections.

### 1.2 Problem Statement

GUISET is a Service Composition broker that must operate where both service components and data partitions can be remotely hosted to overcome the limitations of mobile devices e.g. loss of connectivity. Hosted data, through DBaaS, must be available to support mobile communities where Internet connectivity outage is rampant, thus enabling the community members to continue accessing their data locally (Thomas 2009). For Ad hoc mobile clouds, local data provisioning is a challenge due to storage limitation in mobile devices. Thus, the challenge addressed here for Ad hoc mobile clouds is for the DBaaS data to be provided locally to cater for disconnection periods.

In addressing this challenge there is a need for a caching mechanism that takes into account the nature of an ad hoc mobile cloud-powered GUISET environment. Hence, this work proposes a caching model that will ensure data is available in an ad hoc Mobile Cloud, whenever the cloud database cannot be reached due to Internet disconnections.

### 1.3 Research Question

What will be required to make the DBaaS models tolerate possible connectivity outage in an ad hoc Mobile Cloud situation?

### 1.3.1 Sub-questions

- Why is the current DBaaS handling of offline data necessary to maintain smooth running of clients during connectivity outage not adequate in ad hoc Mobile Cloud?
- How can we ensure that DBaaS models tolerate the data requirements of Clients with regards to data availability in an ad hoc Mobile Cloud, in cases of Internet disconnections?

## 1.4 Research Goal

The goal of this research is to propose a caching model that ensures data is available to its Clients in an ad hoc Mobile Cloud during connectivity outage.

## 1.5 Objectives

I. Document why the current caching approaches to tolerate faults in DBaaS are not adequate for ad hoc Mobile Cloud.

II. Use the information gathered from I above to specify what is required to enable DBaaS to make provision for tolerating faults such as connectivity outage.

III. Design a caching model with regards to the fault tolerance specificatio which will support clients which have to operate using an ad hoc Mobile Cloud.

IV. Evaluate the data availability of the fault-tolerant caching model from III above in an ad hoc Mobile Cloud application scenario e.g. m-Healthcare.

## 1.6 Research Methodology

The research approach taken for this study is descriptive and formulative. The descriptive part involves a review of the literature while the formulative part involves the formulation of the model to suit the mobile ad hoc scenario, and the conducting of experiments to demonstrate the functioning of the model. This is explained by the research methods below.

### 1.6.1 Research Methods

#### 1.6.1.1 Literature Survey

Literature was reviewed with the purpose of getting background information on the concepts of interest which include DBaaS, mobile cloud computing, ad hoc mobile cloud and GUISET. The survey on DBaaS was to gain knowledge of how it works, how it is currently implemented and how it can be made to support caching for disconnected mobile clients. Research works on mobile cloud computing were reviewed to see its capabilities and drawbacks, and how ad hoc mobile cloud can be used to make computing for mobile devices more reliable. The knowledge gained on the service composition broker GUISET, how it is structured and how it works was used in the deployment of the cache model. The review of these works together contributed to the realisation of objectives I and II.

#### 1.6.1.2 Formulation of the caching model

A fault-tolerant caching model was formulated according to the specifications documented for the shortcomings of current DBaaS models regarding their Internet connectivity outage tolerance, which is stated by objective III. The model developed should be able to tolerate faults, mainly connectivity outage in ad hoc mobile cloud, with the assumption that there is no connectivity to the main cloud. Hence the local mobile cloud powered by GUISET served as the service composition broker where our model was deployed.

#### 1.6.1.3 Prototyping

A prototype for the proposed model was implemented and tested as the proof of concept, using the mHealth scenario. Experiments were conducted to evaluate its fault tolerance capabilities and the analysis towards the model's benefits and drawbacks was done. The evaluation was done mainly on the basis of data availability, which is defined as the number of successfully completed operations out of all the issued operations. This research method

was used to achieve objective IV. Open source software was used in the development of the prototype. This software included, but was not limited to:

- SQLlite database
- Software Development Kit (SDK)
- Eclipse IDE
- Tomcat server

## 1.7 Contribution to the Body of Knowledge

The provisioning of DBaaS by the cloud has a lot of advantages, as the literature states. However, due to unreliable Internet connections, the accessibility of DBaaS-hosted data is not achievable at all times. This results in unreachable data, requiring clients to wait for Internet link to be restored, which is undesirable for most clients. This study further eases the data storage and accessibility for users by proposing a cache that ensures data availability even in the cases of connectivity outages in ad hoc mobile cloud-powered GUISET environments. This enables end users to continue with their computations even in the face of disconnections, where the cloud database could not be accessed. Furthermore, this cache mechanism will be provisioned to clients at very low cost due to the usage of GUISET, therefore favouring even the financially unstable enterprises, SMMEs.

## 1.8 Dissertation Synopsis

The rest of the dissertation is organised as follows:

Chapter two presents the background on Cloud computing concepts and its application to mobile environments. It further introduces GUISET, and its integration requirements with ad hoc mobile cloud computing.

Chapter three presents related work on caching in mobile environments. Different forms of caching are analysed, and conclusions are drawn regarding the design of the proposed cache model.

Chapter four presents the design of the proposed cache model and the description of its components. It further proposes cache management policies and a prefetching algorithm for retrieving cache data from the cloud. A usage scenario for testing the usability of the cache is also explained.

Chapter five evaluates the cache prototype based on a number of metrics. Analysis of the obtained results is also presented.

Chapter six concludes the dissertation and gives recommendations for future work.

# CHAPTER TWO

# BACKGROUND

## 2.1 Introduction

Distributed computing is an essential mechanism for scientific computing that has been around for decades. It continues to evolve, with more computing paradigms being continuously introduced over the years. This chapter introduces the cloud computing paradigm, its evolution and its evolving technologies, mainly mobile cloud computing. Although this technology eases functioning of mobile devices by moving most of its processing to the cloud, its continual accessibility by mobile clients is not feasible due to Internet disconnections. This work looks at the mitigation of the Internet disconnection fault when accessing the cloud database model, DBaaS, by providing a cache to be used locally by the disconnected mobile clients. Furthermore, this cache is provided through the use of GUISET integrated with Ad hoc Mobile Cloud computing.

This chapter therefore further introduces the DBaaS model and caching as the mechanism to mitigate the DBaaS's unavailability. AMC is also introduced, together with GUISET, and discusses how these two concepts will be integrated for the purpose of this study. The remainder of this chapter is organised as follows: section 2.2 describes in detail the cloud computing concept, with its deployment models and web services discussed in section 2.3 and section 2.4 respectively. Section 2.5 discusses cloud service models, further looking at one form of the service models, the DBaaS model, in Section 2.6. MCC is presented in Section 2.7, together with its applications, advantages and disadvantages. To mitigate the Internet disconnection fault in MCC, a caching mechanism is introduced in Section 2.8, with the introduction of disconnected operation presented in section 2.9, which is one caching approach that is widely used to support occasionally disconnected clients. Section 2.10

presents Ad hoc Mobile Cloud computing and its applicability to this study, while GUISET and its integration requirements are presented in section 2.11. Section 2.12 summarises the chapter.

## 2.2 Cloud Computing

Over the years the Web has expanded beyond the point of being a communication medium to become a platform for enterprises and societies, due to most aspects of human lives and work being moved online. This expansion allows access of computing resources, data storage and management facilities outside the individual's device, eliminating the need to maintain local servers (Pallis 2010; Zhang et al. 2010). As a result, computing resources have become more powerful, cheaper and more widely available than ever before. Moreover, users need to pay only for the resources used. This is what forms the basis of Cloud computing technology.

According to (Neto 2014; Ahmed et al. 2012), although cloud computing was invented in the 21st century, it is actually not an entirely new technology. This is because of some of the concepts that it builds upon, which date back a number of decades. One of these concepts is the ability of multiple users to have shared access to a single resource, which is an idea that was used with mainframe computing, as an endeavour towards eliminating the need of having a mainframe for each individual, thus reducing the costs within organisations. The concept of virtualization that the cloud uses came into existence in the 1960's when IBM announced a mainframes with virtual memory (Cossio et al. 2012). Furthermore, the pay per use pricing strategy for cloud computing resembles what utility computing model is about. Some other computing concepts that the cloud paradigm is making use of include the distributed autonomic and grid computing paradigms, from which it inherits the capability of self-management within an individual computing system (Huebscher & McCann 2008), and the idea linking a number of computers to form one large infrastructure (Foster et al. 2008) respectively.

By definition, cloud computing refers to a model that aims towards the provisioning of "ubiquitous, convenient and on-demand network access to a shared pool of configurable computing resources such as networks, servers, storage, applications and services, which can be rapidly provisioned and released with minimal management effort or service provider interactions" (Mell & Grance 2011). It allows the consumption of services over the Internet by clients with little or no knowledge of the underlying hosting infrastructure. These services or resources can be accessed in an on-demand manner (i.e., according to the enterprise's needs), which implies that only services that are required need to be rented. Other characteristics of the cloud best classified by (Wang et al. 2011; Mell & Grance 2011) include:

**Broad network access** – the cloud resources are available at the provider's site and can be accessed through standardised network protocols by thick or thin client interfaces, such as smartphones, laptops or tablets.

**Resource pooling** – users select from a pool of computing resources which are dynamically signed and reassigned to them according to their demands, with the location of the services being kept transparent from the user.

**Rapid elasticity** – computing resources are elastically provisioned and released on-demand and can be assigned automatically, ensuring that the user obtains the required service at any time.

**Measured service** – the resource utilization can be monitored, measured and controlled from both the service consumer's and the provider's sides, providing transparency for both of the parties involved. This also enables the consumer to pay for only what they used.

**High reliability** – utilising cloud computing is much more reliable compared to local computing since the cloud uses mechanisms such as computing node isomorphism exchangeable and multi-transcript fault tolerance for data to ensure the service's high reliability.

### 2.3 Deployment Models

The cloud computing model provides enterprises with the flexibility to deploy the cloud infrastructure depending on their needs. The cloud primarily consists of four deployment models that organisations can choose from when deploying their cloud environments.

**Public Cloud** – The usage of the computing infrastructure which is hosted by the cloud provider is intended for the general public. The infrastructure is hosted at the provider's premises, with transparency on where and how it is hosted. This is of advantage to users as they need not worry or pay for infrastructure, but only for the services utilised. As a result, all the risks, such as failures, are shifted to the cloud provider (Jansen & Grance 2011).

**Private Cloud** – In private cloud, the computing infrastructure is intended to be used exclusively by a particular organisation. The infrastructure may be operated by that organisation, an external provider or the combination of both. Furthermore, private clouds are said to provide a higher degree of control over security, reliability and performance (Subramanian 2011).

**Hybrid Cloud** – Hybrid clouds are the combination of both public and private clouds, which work towards addressing the challenges faced by both models. The responsibility of running service infrastructure is divided between the two models: one part runs in the private cloud, the remaining runs on public cloud. Public clouds have been observed to provide more flexibility than any of the other models, as they are equipped with the ability to tighten

application data control and security in comparison to public clouds, whilst still maintaining easier on-demand service expansion and contraction (Zhang et al. 2010; Subramanian 2011a).

**Community cloud** – with community cloud, the computing infrastructure is shared among several organisations or individuals with common concerns such as security requirements, compliance considerations and jurisdiction. the infrastructure may be hosted and managed by a third party, by one or more of the organisations within the community or by the combination of both (Marinos & Briscoe 2009).

For cloud resources to be accessed, web services technology is used for connection between the consumer and the cloud repository (i.e., consumers consume cloud resources as web services).

## 2.4 Web Services

A web service is defined as a business logic, located on the Internet, and which can be accessed by any application connected to the network through Web protocols such as HTTP. Moreover, they are self-contained, self-describing and modular applications that can be used to solve problems, complete tasks or conduct transactions (Chappell & Jewell 2002). Web service functioning can vary from a simple request such as a weather report or currency converter to requests that involve the invocation of other services to complete the task (such as travel planner and packet tracking system). There are mainly two types of web services: SOAP web services and RESTful web services.

### 2.4.1 SOAP Web Services

SOAP web services are SOA-driven services. More specifically, they are defined using Web Service Description Language (WSDL), published on the Universal Description, Discovery

and Integration (UDDI) and communicated through Simple Object Access Protocol (SOAP) messages over the Internet (Curbera et al. 2002). Their characteristics include:

**Loosely coupled** – neither the web service nor the consumer requires information about internal structures of the other party. The interface of the web service can constantly change without compromising the client's interaction with the service.

**Support for RPCs** – web services provide support for RPCs, which expose the input and output parameters supported by a web service, allowing clients to invoke remote procedures, methods and functions through an XML-based protocol.

**Technology independent** – SOAP web services' accessibility is independent of its underlying platform due to its formal WSDL definition.

**Location transparency support** – service definitions and locations are stored in a repository (UDDI), which enables clients to invoke them irrespective of their actual location.

**XML based** – they use XML to represent their data which eliminates any binding (such as platform, operating system or network) that a protocol has.

### 2.4.2 RESTful Web Services

RESTful web services are mainly characterised by their simplicity when published and consumed. They are based on the Represential State Transfer (REST) architectural style, which adopts the client-server architecture of the web and relies mainly on three design principles (Adamczyk et al. 2011; Belqasmi & Glitho 2011):

**Addressability** – RESTful web services expose the interesting parts of their datasets as resources, with each resource being identified via a URI.

**Uniform interface** – any operations that can be performed on a resource is represented by one of the four HTTP methods: GET, POST, PUT and DELETE.

**Statelessness** - each and every request to the server is issued in complete isolation.

Research has been conducted in SOAP and RESTful web services comparisons towards making an architectural decision that best fits the application's needs (Pautasso et al. 2008), with some of the main conclusions drawn being:

1. RESTful web services are more suitable for simply tactical, ad hoc integration scenarios, due to asynchronous capabilities from the application layer perspective.
2. SOAP web services are preferable in enterprise application integration scenarios where more flexibility and advanced quality of service requirements are required.

Moreover, due to the recent wide adoption of smartphones and mobile technology(Hamad et al. 2010; Alshahwan et al. 2012) compare SOAP and RESTful web services to find the ones best suitable for mobile devices. The results prove RESTful web services to be more favourable for the mobile environment with greater flexibility and lower overhead. Therefore in this work, we use RESTful web services for implementation purposes.

### 2.5 Service Models

The services that are provided by the cloud can be grouped into three categories as follows:

**Infrastructure as a service (IaaS)**: the infrastructure as a service model is the one responsible for the provisioning of resources that are more hardware oriented, such as storage and computing capabilities over the network. This model enables the user to have full control of resources which can also include operating systems (i.e., a user can deploy and run their own operating systems and applications), by making use of virtualization technologies. The underlying infrastructure is managed by the cloud service provider in a manner that does not

interfere with the control of the end user (Kulkarni et al. 2012). Examples of IaaS include Google Compute Engine for virtualized operating systems and Amazon S3 for storage service.

**Software as a Service (SaaS)**: The SaaS model provides a complete application to end users. All the management tasks involved including the management of the underlying operating systems, storage, servers, network or the configuration of the application itself are completely handled by the service provider. Users access the applications through a program interface or a thin client interface (e.g., web browser) (Dubey & Wagle 2007). Examples of this model include Google docs and SalesForce.

**Platform as a service (PaaS)**: instead of providing the infrastructure, cloud service providers can offer an abstraction level where users can run their applications. The infrastructure that the user applications run on is managed by the service providers, and the application should be developed in programming languages, libraries and tools that are compatible with the service provider's side. The user is given full control over applications that are being developed, run and hosted on the cloud platform (Kolb & Wirtz 2014). Popular examples of PaaS include Google Apps Engine and Windows Azure.

One form of PaaS whose adoption is widely increasing is Database as a Service.

### 2.6 Database as a Service (DBaaS)

Database provisioning is one of the more costly facilities for enterprises. This is due to expensive hardware or software used, frequent updates and the cost of hiring administration staff. Also, in some cases, companies tend to buy a number of servers beforehand which might end up not being used. One of the mechanisms used to overcome this high-cost issue of database functionality is Database-as-a-service (DBaaS). DBaaS is a cloud-based approach

towards the provisioning and management of databases (Forrester Consulting 2012). It enables enterprises to utilise the facilities such as hardware and software provided by the service provider to set up their own database, therefore saving monetary and time costs.

(Shehri 2013; Sudha et al. 2015) outline some of the key advantages of DBaaS, which are:

**Cost reduction** – by using the DBaaS mechanism companies need not set up their own data centres, hire the administrators to manage the servers or pay for the updates. This results in a lot of monetary savings.

**Pay-per-use terms** – the user pays only for the amount of storage that is used.

**Security** – the client's data is secured with encryption standards, which prevents an intruder from being able to read it.

Unlike traditional database models which can be accessed using machines that they are specifically deployed on, the DBaaS mechanism gives users the flexibility to access their databases anywhere with only the necessity of having an Internet connection.

DBaaS is classified into two types, namely SQL databases and No-SQL databases (Arora & Gupta 2012). SQL cloud databases are those whose structure is mainly driven by tables, in the same manner as the traditional Relational Database Management Systems (RDBMSs). Normally they are transactional databases, containing sensitive data such as credit card numbers and employee data. Some of their properties are:

- ACID compliant
- Use of SQL for querying data
- Provision of support for online transactional applications

The second type, NoSQL ( an acronym for "Not Only SQL"), is mainly characterised by its capability of storing and processing big, unstructured data effectively (Han et al. 2011). This data is usually the social data generated by social media such as Facebook and Twitter, transactional data generated by transactions such as payment orders, invoices and machine data which can be the real-time data gathered from sensors or industrial equipment. Figure 2-1 gives the classification of some of the commercial DBaaS implementations.

| | Amazon RDS (MySQL) | Microsoft SQL Azure | Google Datastore | Amazon SimpleDB |
|---|---|---|---|---|
| **Type** | **RDBMS** | **RDBMS** | **NoSQL** | **NoSQL** |
| **Maximum amount of data that can be stored** | 1 terabyte per database[2] | 50 gigabytes per database[3] | Not published for entire database, but 1 MB limit on a subset of data (called an entity). Limit to the number of indexes. | 10 gigabytes per database domain (roughly equivalent to an RDBMS table)[4] |
| **Ease of software portability with similar, locally hosted capability** | High. MySQL instantiation in cloud is very similar to the local instantiated version. | High. Most SQL Server features are available in SQL Azure. | Medium/Low. Requires Java Data Objects or Datastore-specific interface and use of App Engine. | Medium. Requires SimpleDB-specific interface. |
| **Transaction capabilities** | Yes | Yes | Yes | Yes |
| **Configurability and ability to tune database** | High. MySQL instantiation in cloud. | Medium. Can create indexes and stored procedures, but no control over memory allocation or similar resources. | Low | Low |
| **Database accessible as "stand-alone" offering** | Yes | Yes | No. Requires Google App Engine application layer. | Yes |
| **FISMA Certified** | No | No | No | No |
| **Can designate where data is stored (e.g., region or data center)** | Yes | Yes | No | Yes |
| **Replication** | Yes | Yes | Yes | Yes |

**Figure 2-1: Different DBaaS models (Pizette & Cabot 2012)**

## 2.7 Mobile Cloud Computing

Mobile devices have recently become part of our everyday life. This is due to their capability of not being bound by time and place (i.e., they can be used anytime, anywhere), and their ability to network wirelessly .Although mobile devices and their associated computing paradigm (mobile computing) continue to evolve and become smarter, they are unable to

exploit their full potential due to their inherent issues such as resource poverty, intermittent network connections and security. For this reason, Mobile Cloud Computing (MCC) was introduced. MCC is simply the integration of cloud computing with mobile devices (Dinh et al. 2011). It was introduced to ease the resource poverty state of mobile devices and improve the performance of the applications by moving most of the processing and storage from the mobile device to the cloud servers, which is termed as offloading (Kumar & Lu 2010). These resources residing in the central cloud servers can be accessed through a web browser or thin client on the mobile device, over a wireless connection link. The architecture for MMC is depicted in Figure 1.

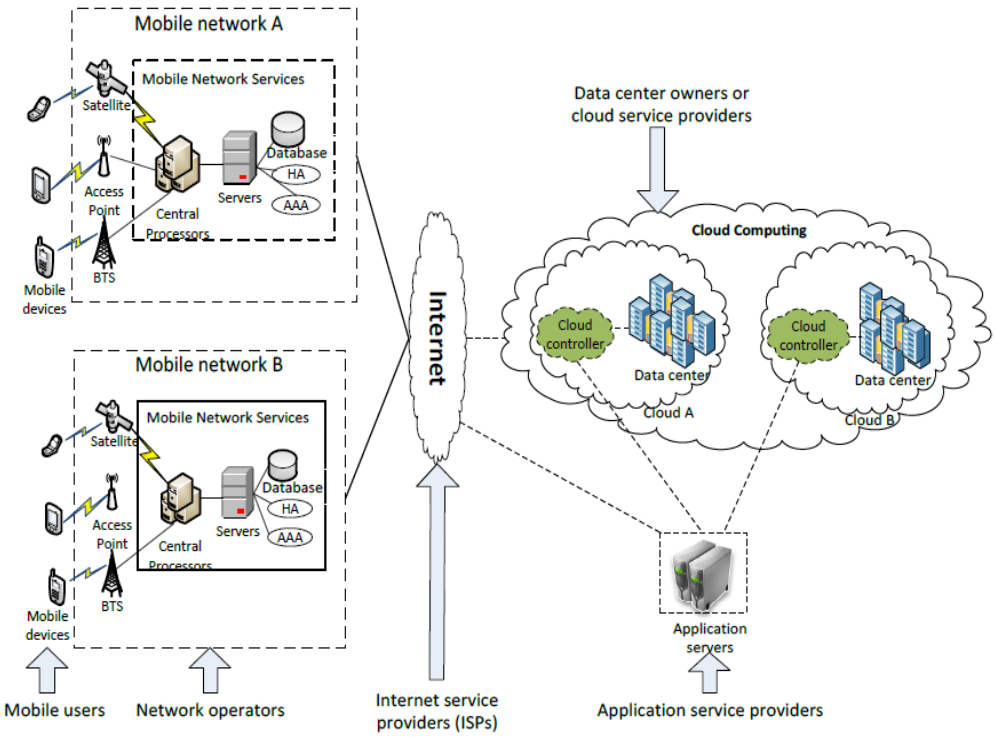



**Figure 2-2: Mobile Cloud Computing architecture(Dinh et al. 2011)**

### 2.7.1 Applications in Mobile Cloud Computing

Mobile cloud applications have emerged, in recent years, as the mobile applications that are taking advantages of MCC. Some of these typical applications are introduced in this section.

#### 2.7.1.1 Mobile Commerce (m-Commerce)

m-Commerce is a business model that deals with the conducting of electronic business transactions through a mobile device. It is a growing model, as evidenced by its less than €9 million revenue in the Western Europe in 2001 followed by its impressive growth of up to €280 million two years later in Germany alone (Tiwari et al. 2007). In 2013, mCommerce revenues were valued at $133 billion and expected to reach $616 billion globally by 2018 (statistica 2015).

Due to the various challenges that m-Commerce applications face (such as mobile configurations complexity, bandwidth and security), their integration with cloud computing provides something of a rescue (Shaikh & Gupta 2014). This is evidenced in cases such as (Xiaoyan Yang et al. 2010), where a 3G Mobile e-commerce Platform Based on Cloud Computing was proposed. In their work, the advantages of both cloud computing and 3G are combined to enhance the security level data processing speed based on Public Key Infrastructure (PKI).

#### 2.7.1.2 Mobile Healthcare (mHealthcare)

mHealthcare is an extension of eHealthcare into the mobile environment. It enables easier access to resources such as patient's medical records to healthcare professionals. The integration of MCC into mHealthcare strives towards the minimisation of limitations (such as security, limited storage and medical errors (Kohn et al. 2000)) that traditional medical treatment faces. MCC integration with mHealthcare application can be categorised into a number of schemes. (Varshney 2007) introduces five mHealthcare applications for pervasive environments:

**Comprehensive health monitoring services** – these applications allow the monitoring of patients at any given point in time, regardless of the patients' location.

**Intelligent Emergency Management Systems** – applications of this type can effectively manage the calls received when accidents or incidents occur, together with the fleet of emergency vehicles.

**Health-aware mobile devices** – these allow the detection of certain conditions such as blood pressure, level of alcohol and pulse rate through the touch of a user.

**Pervasive access to healthcare information** – this allows access to medical history by patients or healthcare professionals.

**Pervasive lifestyle incentive management** – this is used to pay monthly network charges, healthcare expenses or donations to charities with the money generated by the user device whenever the user eats healthily or exercises.

A mobile healthcare application is proposed by (Doukas et al. 2010), which stores, retrieves and updates data using cloud computing. This application uses Android-based mobile client and Amazon's S3 Cloud Storage Service to manage patients' medical records and images. Four services utilising this setup are presented:

**Seamless connection to cloud computing storage** – enables users to retrieve, modify and upload medical contents using web services and RESTful API.

**Patient health record management system** – displays and manages information about a patient's status, image content or related bio signals through the application's interface.

**Image viewing support** – enables users to decode large image files at different resolution levels given different network states by making use of progressive coding.

**Proper user authentication and data encryption** – authenticates the user of the cloud service using SHA1 hashing for authenticating messages and SSL for encrypting data communication.

For practical usage, a number of systems have been implemented such as (Tang et al. 2010), which implements a health monitoring system through the utilisation of SMS and MMS technology. Through this system, patients are monitored, especially those with diabetes and hypertension. If the patient performs blood pressure or blood glucose measurements using a specialised measuring tool, the system automatically receives the measurements through the measuring equipment, or the user can send the measurements through an SMS. The information is stored in the cloud and later analysed and the results returned to the participant. While this system raises security concerns, (Jia et al. 2011) proposes a solution towards protection of participants' information by developing a model to provide security as a service for mobile applications utilising the cloud. This, therefore, ensures security on mobile apps data in the cloud.

#### 2.7.1.3 Mobile Learning

m-Learning is the integration of e-Learning into the mobile environment. While traditional m-Learning applications suffer from resource limitations such as storage space, low network transmission rate and limited educational resources (Hong-qing & Yan-jie 2010), cloud-based m-learning takes advantage of the infrastructure provided by the cloud to provide learners with much richer services in terms of processing speed and information size. Such is the case with (Weiqing et al. 2010), where a computer-aided platform was developed to enhance the interaction between the students and teachers. In this work, software developed for mobile devices using J2ME framework was used. Students would communicate with teachers through a website based on Google App engine, enabling teachers to acquire information on the student's level of knowledge for that course and to answer any questions asked.

Other MCC applications include mobile gaming (mGame) (Li et al. 2001), which works towards increasing the time in mobile devices by utilising the offloading concept to save energy for mobile devices. (Li & Hua 2010) is an automatic multimedia blogging MCC application that allows mobile users to share real-time information (e.g., current location and events) through clouds. With these applications it can be seen that MCC is a highly trending technology, with more applications expected in the near future.

### 2.7.2 Advantages of Mobile Cloud Computing

The basic idea behind MCC is for mobile devices to take full advantage of the cloud. This minimises the burden on the mobile device's resources as most computation will be done on the server side (the cloud), which results in the following advantages:

**Extended battery life** –the rate of energy consumption is one of the constraints in mobile computing. Different approaches to achieving low power consumption in mobile devices have been proposed (Mayo et al. 2003; Carroll & Heiser 2010), with most of them requiring a change to the mobile device's structure, which is not viable for the majority of mobile devices. The concept *of computation offloading* was introduced and has been claimed to conserve the mobile device's battery power as it migrates computations from the resource-limited mobile devices to the resourceful cloud servers (Kumar & Lu 2010).

**Improved reliability** – mobile device's hardware failures or crashes most likely result in a device's memory contents being lost. Storing data outside the mobile device (i.e. in the cloud) is much more reliable since the copy of the data is distributed across a number of virtual servers (Dinh et al. 2011).

**Improved processing power and storage capacity** – other limitations of mobile devices include storage and processing power. MCC introduced the ability for mobile devices to store

their data in the cloud, gaining access to a large amount of memory space. Examples of the storage services include Amazon Simple Service Storage, Google Cloud Storage and Dropbox (Shiraz et al. 2013).Applications such as Instagram and Flickr have been developed for MCC users. With such applications, users can share and save their pictures on the cloud, saving a substantial amount of space in their devices. Moreover, MCC makes it possible for mobile devices to utilise the cloud's processing power for compute-intensive applications executions, reducing their running cost as compared to when they are run on a mobile device.

### 2.7.3 Disadvantages of Mobile Cloud Computing

Despite the main advantage of extending the cloud flexibilities into the mobile environment, MCC also experiences a number of drawbacks. These include security, mobility and intermittent internet connectivity. This work considers the intermittent internet connectivity issue, the attempts that have been made in rectifying it and what can be done to further decrease the rate of cloud resources' inaccessibility during disconnections. Although not much work has been done in explicitly dealing with the frequent internet disconnections problem in MCC, the following sub-section looks at the caching mechanism that the cloud and its evolving technologies adopt in an attempt to provide offline access to cloud resources.

## 2.8 Caching

Caching is a mechanism used to temporarily store frequently accessed data close to a client in computing environments. It is widely used in different computing environments such as computer CPU's and distributed systems (Milutinovic 2000; Chun et al. 2004). It is mainly characterised by the following advantages:

I. Minimises bandwidth consumption.

II. Minimises latency since the requested data is now fetched at the location closest to the client.

III. Reduces the workload on the main server as some of the client requests would be serviced by the cache.

IV. Supports disconnected clients by maintaining data availability during network disconnections.

This study takes a deeper look at the advantage of maintaining data availability to disconnected clients, and the mechanisms that have been used towards achieving it. One such approach is disconnected operation, which is discussed in the following section.

### 2.9 Disconnected Operation

Disconnected operation refers to "the ability of a distributed system client to operate despite server inaccessibility by emulating services locally" (Kistler 1993). The support for disconnected operation by different computing systems is a field that has been richly explored in Mobile Computing, but with little being done in Cloud Computing and the technologies that evolve from it. Earlier systems that provided support for disconnected operation include The Coda File System (Kistler & Satyanarayanan 1992) and Bayou Architecture (Demers et al. 1994). Support for disconnected operation involves the caching of services to a computing device, for later usage when disconnections occur. It works in three stages or phases. The first one is when there is still an Internet connection link between the computing device and the main server. Contents that are most likely to be used in the near future are pre-fetched from the server into the device. During disconnections, the system emulates the behaviour of the server by servicing all the user requests on the cached data.

## 2.10 Ad hoc Mobile Cloud Computing

For mobile devices to use the cloud resources, the primary necessity is that of a continual uninterrupted connection link to the cloud. However, due to the mobility nature of the devices and the unreliability of wireless networks, the Internet connection becomes interrupted from time to time. This is one of the major parameters behind the proposal of Ad hoc Mobile Cloud Computing (AMC) (Kovachev et al. 2010). AMC presents a scenario where mobile devices within a computing community serve as service providers by exposing their computing resources to each other. This technology strives towards overcoming the resource unavailability situation in mobile devices by being able to request for, and provide services that are locally available among peers within the network, or collaborate towards the execution of shared tasks (Kovachev et al. 2011). AMC can be deployed in two scenarios: through distributed and collaborative processing, and single-node-as-a-server processing (Liu et al. 2013). The first scenario consists of mobile nodes collaborating in executing tasks. With the resource scarcity situation in mobile devices, this scenario results in compute-intensive tasks being executed in a collaborative manner, saving the device's resources such as battery lifetime and computational power (Mtibaa et al. 2013). The second scenario involves a single node with better resources (such as computation memory and storage space) acting as a server, with all the requests within the network being issued against it. Figure 2-3 illustrates this model.

For the practical aspect of the AMC paradigm, (Huerta-Canepa & Lee 2010) presents a framework to create virtual mobile cloud computing providers that mimics the behaviour of the traditional cloud providers within the vicinity of users. With this system, if a user wants to execute some heavy computation that requires more than the available resources within the mobile device, the system detects the stable nodes in the vicinity. If available, the ad hoc

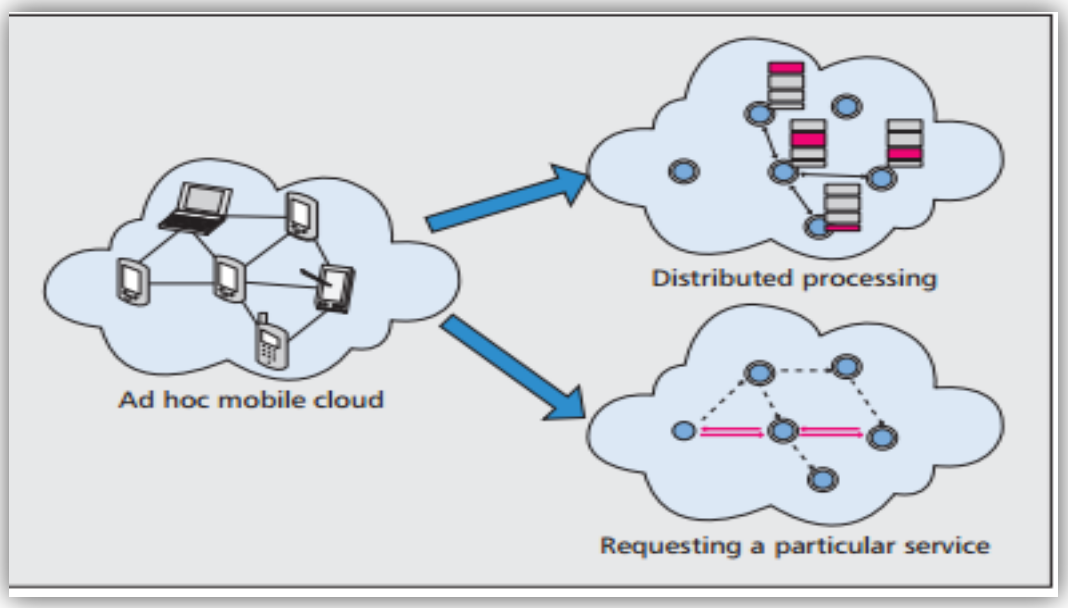



**Figure 2-3: Ad hoc Mobile Cloud architecture**(Liu et al. 2013)

cloud is formed and the loading application is intercepted and modified to use the virtual cloud.

## 2.11 GUISET

The idea of remote service provisioning through fast Internet connections brought about a number of advantages to users, as aforementioned. One of these advantages is cost reduction, as users and enterprises need no longer worry about setting up infrastructures to use. However, for SMMEs, affordability of ICT infrastructures still remained a challenge. For this reason, the GUISET project was proposed. GUISET (Grid-based Utility Infrastructure for SMME Enabling Technology) strives towards the provisioning of technology infrastructure at very low monetary cost to less-affording organisations. This is achieved through a platform that allows organisations to expose their products whilst spending close to nothing on technology.

### 2.11.1 GUISET Architecture

The GUISET architecture is divided into three layers: multi-modal interfaces layer, middleware layer and Grid infrastructure layer. This architecture is depicted in Figure 2-9.

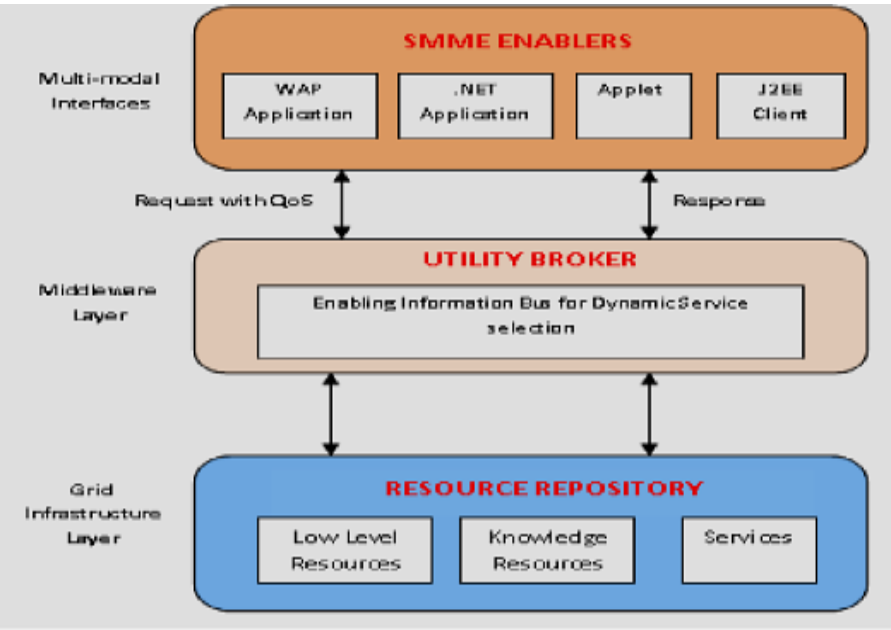



**Figure 2-4: the GUISET architecture**(Ekabua & Adigun 2010)

**Multi-modal interface layer** is where services which are exposed as interfaces to clients are hosted. This layer enables interactions between clients such as SMMEs and the other GUISET layers (middleware and grid interface layers), regardless of the clients' underlying technology.

**Middleware layer** contains the enabling information bus for dynamic service discovery and selection mechanisms. It also acts as the gluing layer between multi-modal interfaces and grid infrastructure layers.

**Grid infrastructure** layer is the resource repository layer where resources and services for GUISET reside.

Recently, research on GUISET such as (Akingbesote et al. 2014) has also delved into MCC and its application services, investigating performance issues based on the waiting time for m-healthcare's application requests. This study proposes GUISET integration with AMC technology to further reduce the costs associated with the provisioning of services to disadvantaged enterprises whilst maintaining data availability even when connectivity to the central data repository is rampant (Fakude et al. 2015).

## 2.12 Chapter Summary

Cloud computing has been one of the most highly adopted technologies of the past few years, further extending its capabilities to the mobile environment, which brought about the concept of MCC. Thorough studies on MCC, its applications, advantages and disadvantages have been conducted and have proved this computing paradigm to be one of the most effective and most used technologies in the market today. Despite this phenomenon of high adoption, MCC's drawbacks still negatively impact clients (individuals or organisations), hence, there is a need for them to be mitigated. This chapter, therefore, has introduced the background concepts of our work and looks deeper at the client disconnection issue, and the caching which is the mechanism used to provide data availability to disconnected clients.

Deeper observation of the different caching strategies used to provide data availability to disconnected clients in cloud-based and other computing paradigms is presented in the next chapter.

# CHAPTER THREE

# LITERATURE REVIEW

## 3.1 Introduction

The aim of this work as stated in the preceding chapters is to formulate a caching model that will enable users in an AMC environment to continue with their operations in the face of internet disconnections by having access to their data locally. Caching is the mechanism that has been widely used in the continued provisioning of data to temporarily disconnected clients. Different forms of caching exist for cloud computing and mobile environments. However, they impose a number of challenges, especially when used in mobile environments due to storage limitation on mobile devices. In this chapter a review of different forms of caching is presented, including their advantages and shortcomings. Moreover, we look at the Database-as-a-service model and its inability to maintain user satisfaction at all times due to faults that occur, with one of these faults being the Internet disconnection between clients and the central cloud server. Caching has also been used as the fault-tolerant strategy for the DBaaS model; therefore, this chapter further looks at how caching has been used towards ensuring the availability of data that is persisted in the cloud database during the period of its inaccessibility. Due to little or no work having being done on supporting disconnected operation in AMC environments, this research work has proposed a caching solution to fill the gap. Therefore, a review of related works under MCC and mobile infrastructureless networks is conduted based on the fact that AMC technology is mainly the integration of these two computing paradigms. Based on the works reviewed, a recommendation on the solution for the problem addressed by this research is presented and the manner in which it is going to be integrated with the AMC-powered GUISET environment introduced in the previous chapters highlighted. The rest of this chapter is organised as follows. A brief

overview of how data availability has been maintained for disconnected clients in Grid and Mobile computing paradigms is presented in section 3.2 and section 3.3 respectively. Section 3.4 discusses the different caching mechanisms in MCC and mobile infrastructureless networks. The prefetching mechanism for caching is discussed in section 3.5, and the usage of SQLite as a shared cache is presented in section 3.6. The fault tolerance in the DBaaS model and how caching has been used to cover for this model's downtimes is discussed in Section 3.7, while Section 3.8 summarises the chapter.

### 3.2 Disconnected clients support in Mobile Computing

The nature of the disconnection problem has been presented and explicitly dealt with in the context of Grid and Mobile Computing. Cloud computing has not focused much on trying to significantly address this issue, although the Internet disconnections heavily affect its accessibility by users. This subsection looks at some of the efforts made to address this problem in Mobile and Grid Computing paradigms. The Coda File System (Kistler & Satyanarayanan 1992) is one work that exploits the support for disconnected file system clients in the context of Mobile Computing. When disconnections occur, the Coda File System solely relies on its cache called Venus. Venus operates in three states: hoarding, emulation and reintegration. The system is usually in the hoarding state, but is always ready for any possible disconnection. In this state, data to be used during disconnections is pre-fetched from the main server and stored locally on Venus. Upon disconnection, it enters the emulation state, where the operations to be performed on the servers are now performed on the cache. In this state, Venus maintains the log that keeps track of the operations performed in offline mode. When the connection comes back Venus moves to the reintegration state where all the changes made during disconnection are propagated to the server.

Similarly, (Gruber et al. 1994) propose a disconnection support for an object-oriented database, Thor, which is further implemented by (Chang & Curtis 2002). This support also

works in three phases: preparing to disconnect, operating disconnected and reconnect. In the preparing to disconnect phase, which is where caching takes place, queries are executed to pre-fetch the data that the user might need during the disconnection. In operating disconnected, the client still attempts to perform transactions as it normally does when connected. In this phase, the commit becomes a tentative commit, meaning that there is a possibility of it being aborted by the Object Repository (OR) upon reconnection. A transaction log is kept for each transaction performed during disconnection so that each transaction will be replayed once the Front End (FE) reconnects with the OR. In the reconnect phase, the synchronisation of the log occurs before any other connected operation. The synchronisation with the OR consists of all the tentative transactions replayed in the order they were committed in during the disconnection. Any aborts and invalidations are handled accordingly.

Work by (Lim et al. 2013) analyses the effect of voluntary disconnected operation for energy conservation in mobile computing. In this work the authors argue that the WiFi module exhausts more energy than local memory even in its idle state, hence they propose the usage of local storage most of the time to slow the energy exhaustion in a mobile device. In their work, data that is going to be used during the period when WiFi connectivity is off is prefetched and stored locally in a mobile device. Any changes that were made locally on the data are propagated back to the cloud server when the connectivity is restored. Results indicated less energy expenditure for mobile devices employing voluntary disconnected operation.

### 3.3 Disconnected Clients Support in Grid Computing

In an almost similar operational approach to that of mobile computing, a Disconnected Operation System (DCOS) is proposed towards supporting disconnected operation in Grid computing by (Konanki & Butt 2007). DCOS achieves its goal through the use of client

modules resident on mobile devices, and the proxy components which are assumed to be always connected to wired grid nodes. The client interacts with the grid through the proxy component. DCOS also works in three phases: connected phase, disconnected phase and synchronization phase. In the connected phase, data is fetched from servers and cached on the client modules. During disconnection, any requests for data from the grid are paused by the proxy, allowing only the mobile devices to access the cached data. Only after the synchronization phase are grid requests replayed.

(Park et al. 2003) developed a job scheduling algorithm for mobile grid systems that provides reliable performance even if some nodes are disconnected from the network. In their algorithm an object called Job-Proxy is proposed, which will manage the running job on behalf of the mobile device when an error is detected with the mobile device. When a job to be executed is taken from the queuing server and submitted on a mobile device, the agent at the gateway creates the Job-Proxy to look after the mobile device and take over whenever the device becomes disconnected. When the device resumes, the proxy returns the results to the mobile device and then the device takes it from there. The algorithm they developed assumes that the networking behaviour of each device is known and available to the scheduler.

The approaches discussed in Section 3.2 and 3.3 are adopted in this research work for an ad hoc mobile cloud environment, with the purpose of maintaining data availability during Internet disconnection periods between the cloud and MCC applications. Although the literature has dealt extensively with the usage of caching towards supporting disconnected clients in mobile and grid computing paradigms, the direct applicability of these solutions to cloud-based computing paradigms is not feasible due to differences in architectural approaches between Grid, Mobile and Cloud computing paradigms (Foster et al. 2008). Hence, we further look at the attempts put forward for cloud-based computing paradigms.

### 3.4 Caching in Cloud-based Mobile Environments

This section examines different forms of cloud caching for mobile environments and how they have been used to support disconnected clients by maintaining data availability during network disconnections.

#### 3.4.1 Caching SOAP Web Services

As mentioned in the previous chapter, disconnected operation is not a new paradigm as it dates back to the 20th century (Kistler & Satyanarayanan 1992). Despite this, its support has been provided for recent technologies such as MCC. Work by (Terry & Ramasubramanian 2003) provides an insight into factors to be considered when caching web services to support disconnected operation in mobile environments. An experiment was conducted where a proxy server containing a cache was placed between the web server and the client application. The responsibility of the cache was to store the HTTP request messages from clients, along with the returned response, which would later be used during disconnections. Similarly, (Ramasubramanian & Terry 2004) proposes cache architecture for XML web services that will improve service availability during Internet disconnections. This architecture addresses the cacheability issue in web services (noting that web services were never designed to support caching) by adding annotations to web service descriptions. These annotations assist the web service cache in providing better cache consistency. Although the experiments demonstrated the usability of proxy caching during Internet disconnections, the usage of a proxy cache in a mobile device may result in a number of complications:

I. a proxy cache deployed in a mobile device may result in a single point of failure (Piatek et al. 2008).
II. the deployment of a proxy server on a mobile device is not an ideal solution as mobile devices experience storage limitations (Qi & Gani 2014), hence such a solution may

not be very suitable in working environments where a relatively high number of services are to be cached.

III. work by (Wang 1999) argues that the proxy cache might not be a feasible solution in resource-constrained devices due to its high demand of computational resources.

To overcome these proxy issues, (Elbashir & Deters 2005) propose a proxy cache architecture to support mobile disconnected clients that can be deployed as a stand-alone proxy cache or be embedded within an application. In their work, web service requests and responses are cached and for each session, a separate cache is created to ensure the session's consistency. These session caches were later used to synchronise with the proxy cache for data consistency. Although experimental evaluation of the proxy cache when used as a stand-alone component were not carried out, its possibility of performing scalably under varying amounts of workload is highlighted. Work by (Zhu et al. 2012), however, argues against the use of multiple caches within a single system, stating that it may result in increased processing and monetary costs. (Dorn & Dustdar 2006) theoretically address web services continuity in mobile environments. Their work mainly prioritises achieving continued service invocation in the midst of Internet disconnections and the continuation of paused or neglected sessions across different locations, within the same network. To achieve this, a web services framework deployed on a client middleware was proposed, which acts as a generic proxy for caching and session relocation capabilities. Their proposed framework enables caching of service requests and responses, while also allowing the service invocation request to be initiated on one device and continued on another one. However, the implementation aspect of the framework to show its usability was not carried out. Other than caching requests and responses, (Takase & Tatsubori 2004) present a client-side architecture for caching only web services responses for reduction of communication overheads. For all the cloud data requests sent, the returned responses are stored in the client middleware, provided that the client

application administrator recognises them as cacheable. Moreover, the requests are used to create cache keys, which will map to the corresponding responses. Similarly, (Papageorgiou et al. 2011) proposes a mechanism to enhance the caching of web service response by ensuring the recency of cached data (i.e., that the cached data is up-to-date at all times), especially for real life mobile web services scenarios. This is made feasible through the development of a mobility layer that transparently verifies the validity of cached data each time the data is needed. This verification in time of need may result in increased response time as the client application would need to wait for verification first after issuing a request. This may result in longer response times as the client application would have to wait for data validation from the main server.

One more challenge generally introduced by the aforementioned solutions is their platform dependency, as they were proposed and tested for SOAP web services. Recent literature points out to RESTful web services being more suitable for mobile environments compared to SOAP web services (Wagh & Thool 2014; Katsaros et al. 2011; Alshahwan et al. 2010; Alshahwan & Moessner 2010). However, due to differences in SOAP and RESTful web services architectures (Pautasso et al. 2008), which are also summarized in Table 3-1, these solutions cannot be directly integrated with the RESTful architecture. Hence, support for RESTful web services is not catered for. In the following section, the application of the caching mechanism in RESTful web services is examined.

| Criteria | SOAP-based WS | RESTful-based WS |
|---|---|---|
| **Server/Client** | Tightly coupled | Loosely coupled |
| **URI** | One URI representing the service endpoint | URI for each rsource |
| **Transport Layer Support** | All | Only HTTP |
| **Caching** | Not Supported | Supported |
| **Interface** | Non Uniform Interface (WSDL) | Uniform Interface |
| **Context aware** | Client context aware of WS behaviour | Implicit Web Service behaviour |
| **Data Types** | Binary requires attachment parsing | Supports all data types directly |
| **Method Information** | Body Entity of HTTP | HTTP method |
| **Data Information** | Body Entity of HTTP | HTTP URI |
| **Describing Web Services** | WSDL | WADL |
| **Expandability** | Not Expandable (no hyperlinks) | Expandable without creating new WS (using xlink) |
| **Standards used** | SOAP specific standards (WSDL , UDDI,WS-Security) | Web standards (URI, HTTP methods, XML, MIME Types) |
| **Security/Confidentiality** | WS-security standard specification | HTTP Security |

**Table 3-1: Differences between SOAP-based and RESTful web services (Alshahwan et al. 2010)**

### 3.4.2 Caching RESTful web services

In contrast to SOAP-based web services being made cacheable through annotations, RESTful web services are cacheable by nature, that is, they were initially designed to enable caching. This cacheability characteristic is one of the reasons RESTful web services are said to be more suitable for mobile devices, as it supports local access to services which avoids remote calls to the cloud servers, thus conserving the device's resources and enhancing the performance of the system (Fredrich 2012). Moreover, it serves as a fault tolerance mechanism since it maintains the provisioning of data even during network failures.

This cacheability mechanism is, in most cases, better achieved through the deployment of an HTTP caching proxy in front of a web server, allowing requests from clients and their responses to be cached for a specified duration, depending on the caching techniques applied. Requests to the server will first be looked up on the cache. If the response is found in the cache, it is returned to the user; otherwise, the request is further forwarded to the main server (Allamaraju 2007; Tere et al. 2014). As can be seen, this mechanism reduces the server workload as some of the requests would be serviced by the cache, which in turn reduces the network traffic, thus optimizing the overall performance of the system. But for a client to have access to the proxy caching server, the Internet connection would be one of the requirements, which makes this caching approach unsuitable for ad hoc mobile clouds where internet connectivity is rampant. Therefore, we examine client-side caching for RESTful web services which minimises the need for Internet connections and hence better supports disconnected mobile environments.

A conceptual framework aimed at reducing effort to achieve adaptive, fault tolerant and reliable service consumption, called RAFT-REST, was introduced by (Spillner et al. 2013). To achieve this, a cache is integrated with this framework to temporarily persist services, and

depending on the network connectivity state which is periodically checked by the network manager component, the system decides whether to service issued requests from the cache. RAFT-REST is provided as a client library for mobile applications, and experimental evaluations indicate the beneficial use of a cache on this framework as it resulted to much faster access to RESTful web services and a higher rate of service availability during Internet connectivity downtimes. The deployment of the framework in a client's device, however, can lead to serious performance degradation issues as all the processing relies on the mobile device's resources.

Due to the wide usage of smartphones under various wireless network conditions, (Ma et al. 2015) propose a connectivity-aware and risk-driven (CARD) approach for smooth delivering of RESTful web services in a community of mobile users sharing similar interests. This is achieved through the use of prefetching and caching of services when the devices are still connected to central servers, to enable continued access to these services during times of instability for Internet connections. CARD is provided in the form of a client-side library to be used in mobile applications. For a mobile application with CARD enabled, when accessing the application's server during stable and unstable Internet connections, CARD transparently pre-fetches and caches the additional information related to that request and uses it to provide responses to subsequent related requests. During periods of total disconnection, CARD translates the user request into a certain format that will enable the user to seek help from other neighbouring users. Although the cache performs scalably under different number and sizes of requests received, our main focus of interest in this work which is the functioning of the cache during total Internet disconnections shows CARD as solely relying on neighbouring nodes for data availability, which may impose challenges to some of MCC applications as privacy is one of their major concerns (Shravanthi & Guruprasad 2014; Wang et al. 2015).

Work by (Ollite & Mohamudally 2015a) proposes a novel transparent proxy architecture for caching web services requests and responses. To utilise this model the client needs to install it on the client device, which will then cache the requests and responses between the client application and the cloud server. All the subsequent requests sent by the application are first sent to the cache. If the response is found, it is returned to the user; otherwise, the request is forwarded to the server-side proxy for further processing. This approach is said to support client disconnections as it provides the mechanism to access the services locally without contacting the central cloud, whilst minimising network latency and processing time between issuing a request and obtaining a response. Performance testing of this approach indicates its better performance and cost efficiency in terms of bandwidth used (Ollite & Mohamudally 2015b). Moreover, this approach resulted in a higher percentage of requests serviced from the cache, making it more suitable for disconnected clients. However, as aforementioned, the deployment of a proxy server on a mobile device may result in performance and memory exhaustion issues.

Most of the works reviewed in this section rely on individual devices for all the processing involved. This is a drawback for mobile devices that do not have enough storage memory as they would not be able to host the cache and therefore not have access to data during Internet disconnection periods. This study is trying to address the latter-mentioned drawback by providing a cache that will be shared locally amongst ad hoc mobile cloud users. The following sub-section looks at how mobile devices can cooperate in caching data to be shared locally.

### 3.4.3 Cooperative Caching in MCC

The main focus of this work is to maintain data availability during Internet disconnections in a mobile environment through an infrastructureless technology which should mimic the

behaviour of the infrastructure-based technology that is normally used when Internet connectivity is available. This can be achieved through the use of a cooperative cache. Cooperative caching allows mobile devices within a network to share and coordinate cached data. It is a form of caching that is widely adopted in mobile infrastructureless networks. The literature on this concept is quite extensive (Chand et al. 2007; Yin & Cao 2006; Joy & Jacob 2012). This work, therefore, examines support for cooperative caching in mobile cloud computing and how it can be incorporated with infrastructureless cooperative caching to provide a solution to the problem addressed by this work. The possibility of a centralised and distributed shared cache for mobile clouds is explored by (Feng et al. 2014), where mobile devices retrieve and distribute data in a minimal energy consumption manner. To achieve this, a number of mobile devices, called cache agents, are selected to hold caches. Each cache agent is associated with a buffer capacity, and each request for a file by the client nodes is noted. The more a file is accessed, the more caches are maintained for that particular file. Although this approach experimentally proved to be ideal for mobile clouds, the deployment of multiple nodes as cache agents may leave some resources unused while adding more caches for frequently accessed data, resulting in complexity of the system, which may be due to least accessed caches being left idle for longer durations. This drawback, therefore, proves the unsuitability of this approach to ad hoc clouds which consists of only a small number of nodes, lasts for shorter durations and in which resource sharing is one of the major characteristics.

For supporting data sharing between mobile devices in a cooperative way, a shared cache memory for mobile cloud computing is proposed by (Kasemi et al. 2014), using a cloudlet as a cache to minimise the response time for client devices. Factors that influence the user's response time are identified and evaluated, and based on these factors, an algorithm for selecting data to be cached is proposed. The algorithm selects data that is most likely to be

used during traffic hours. i.e., during the time of the day when cloud server resources and bandwidth are utilised the most. The algorithm performs well when compared with other selection algorithms, hence improving the user response time. However, the usage of a cloudlet as a cache may require the presence of an Internet link for mobile devices to access it, hence, this requirement makes this architecture less suitable for ad hoc mobile environments. As an alternative to using a cloudlet as a shared memory, (Gao et al. 2012) proposes the usage of a distributed programming layer that exposes a shared cache memory to mobile devices within a community. One node elected as a leader is required to contribute part of its memory towards the universally shared memory, which will be utilised to store variable names which when combined together make up the shared address space of the shared memory layer. The mobile devices are connected by a free, low-latency and energy-efficient ad-hoc Wi-Fi mechanism, which works towards conserving the device's resources whilst upgrading application response times. When accessing the shared memory, the mobility and cooperation coordination between mobile devices is kept transparent to the client applications, as they only see a single global address space presented to them. As can be envisioned, this model can better suit mobile ad-hoc environments mainly due to its lower costs, resource-conserving and transparency characteristics. Similarly, (Koukoumidis et al. 2011) proposes the usage of a cloud service cache architecture called *pocket cloudlet*, as a means of providing support for Internet disconnections in mobile cloud computing and minimisation on the usage of the devices' resources, mainly battery power. This architecture resides on a mobile device's non-volatile memory (NVM), allowing specific parts or even full cloud services to be stored locally on a mobile device, maintaining service availability during Internet outages and conserving the energy utilised together with latency. The authors' motivation behind storing services locally on the mobile device is the recent advances made in memory resources for mobile devices, which promises even greater things in the near

future. Similar work (Qian et al. 2012; Kiani et al. 2012) proposed caching of cloud contents into mobile devices due to this improvement. However, despite the research efforts presented in this section so far, recent literature still observes storage capacity as one of the major challenges facing mobile devices for most smartphones (Aminzadeh et al. 2015). In summary, the aforementioned approaches place a greater burden on local resources, mainly, the storage capacity. In contrast to persisting services in a device's permanent storage facility, (Vemulapalli et al. 2013) explore the possibility of an efficient data sharing mechanism for mobile cloud computing, where potential MCC users share data residing both in their cache storages and the central one respectively. To ensure the proper sharing, a layered architecture that provides balance between the latency cost when accessing cloud data, and the mobile resources cost when accessing local storage is proposed. This architecture comprises a pre-distribution scheme that is used to replicate data to those nodes that are most likely to use it in the near future. The results obtained justify the usage of the layered architecture, as it reduces the latency and conserves the mobile devices' resources when accessing both the cloud and local data. Our proposed work adopts the same concept of using architecture for service provisioning to avoid persisting the services in a mobile device. We use GUISET for cache provisioning, with our primary focus being to ensure data provisioning during Internet disconnections while conserving mobile devices' resources. Table 3-2 below summarizes the MCC based caching strategies reviewed above, their strengths and weaknesses.

### 3.5 Caching Using Prefetching Mechanism

In order to select the contents to be cached , we use the data prefetching mechanism, which is the mechanism whereby data is fetched and placed close to the client prior to its need (Vanderwiel & Lilja 1999). Prefetching data in mobile environments has been extensively explored, with the provisioning of data to disconnected clients being one of the primary reasons for applying this mechanism. Benefits of prefetching on the mobile web have been

| Approach | Purpose | Strengths | weaknesses |
|---|---|---|---|
| **Caching SOAP Web services** | to maintain service availability during Internet disconnections through cache architectures for XML web services | • Uses annotations in web service descriptions to provide better cache consistency<br>• Performs scalably under differing amounts of workloads | • Deployment of a proxy cache in a mobile device may result in a single point of failure<br>• Storage limitation in mobile devices may be a bottleneck to such solutions |
| **Caching RESTful Web services** | • To reduce the effort needed to achieve adaptive, fault tolerant and reliable service consumption<br>• To achieve smooth delivering of RESTful web services in a community of mobile users during Internet disconnection periods. | • Result in a higher rate of service availability during Internet connectivity downtimes | Can lead to serious performance degradation issues as all the processing relies on the mobile device's resources. |
| **Cooperative caching in MCC** | To minimise the response time for client devices and provide continued service availability to users during Internet disconnections, through the usage of a cache | • Improves the user response time<br>• Higher service availability during internet disconnections and decreased usage of the mobile device's battery power | • requires the presence of an Internet link for mobile devices to access it, making this architecture less suitable for ad hoc mobile environments<br>• Storage capacity is still a major concern for mobile devices, mainly smartphones |

**Table 3-2: Differences between various MCC caching strategies**

presented by (Jiang & Kleinrock 1998; M´arquez et al. 2008; Hussien et al. 2013) and include:

I. Minimising network congestion which may result from many users accessing the same resource at once.
II. Ability to maintain data availability to mobile clients who are frequently disconnected from the central servers due to weak connection links and their mobility nature.
III. Conservation of mobile device resources such as battery power and processing power as data would now be accessed locally.
IV. Reduction of latency, thus improving the system's response time.

The prefetching mechanism has also been widely used in mobile infrastructureless networks, where a number of mobile devices in pre-loading data to cache storage. This can be observed in (Denko 2007), where a cooperative prefetching and caching mechanism for mobile infrastructureless networks is proposed. Mobile devices organised into clusters work in a collaborative manner to prefetch data from data sources to a local cache, with each cluster consisting of cache agents for hosting the shared caches, data sources, a cluster head and mobile hosts that request cached data. Results show that a prefetching mechanism improves data accessibility and query delay for different cache sizes. Similar approaches have been explored that take into consideration the user's context (Chauhan et al. 2009; Chauhan et al. 2011) and the user's site (Said et al. 2009), which have proved to work in favour of the system's usability. Additionally, (Chang et al. 2011) propose a context-aware prefetching framework called ProMWS for mobile web services where a user's query is predicted based on current context and used to pre-fetch related services to the mobile device's cache for future requests. ProMWS dynamically deploys the predicted query as a service and broadcasts it to neighbouring devices to assist in prefetching the requested data before a request to utilise it is made. The ProMWS framework works in a peer-to-peer environment,

where a mobile device pre-fetches and caches data from another device. Application of context-awareness on this system minimises the probability of prefetching unnecessary data, thus improving the usability of the cache.

A prefetching and caching mechanism for the mobile cloud is proposed by (Cheluvaraju et al. 2011), where characteristics such as user's request pattern and the mobile device's connection state are considered prior to the prefetching and caching process. For every request $N_i$ that a user issues, all the related items to that search $\{N_1, N_2, N_3, \ldots N_K\}$ are pre-fetched and cached in the mobile device's memory. The prefetching of these item-sets is done during high bandwidth availability periods, in a manner transparent to the user. This model is bandwidth-adaptive, frequently checking the connectivity state and adapts to varying bandwidths, adjusting its prefetching and caching capacity accordingly. It is reported to accurately prefetch the itemsets from the cloud server, with little overhead.

Moreover, (Hussien et al. 2013) indicate the good prefetching mechanism to be timely, which refers to data being hoarded at periodic intervals not too early or too late, before it is needed. Also, the prefetching should be such that the pre-fetched data is most likely to be used in the near future, thus minimising the possibility of wasting resources by caching data that is not going to be utilised. For the purpose of this work, we follow the same criteria for developing the prefetching mechanism that is going to be used to fetch data to be persisted in the cache prior to disconnections. In addition to the above reviewed works on prefetching, our prefetching approach uses the time parameter as the basis of the data to be pre-fetched from the cloud database.

### 3.6 Using SQLite as a Cache Engine

Caching is a mechanism widely used for different purposes including performance enhancement by reducing the workload on back-end servers (Saemundsson et al. 2014),

maintain data locality (Amiri et al. 2003) and support for disconnected clients (Ramasubramanian & Terry 2004). A number of open source cache implementations are available, including Memcached (Memcached 2016), Squid (Squid 2016) and Polipo (Polipo 2016). For the purpose of this study we use the SQLite server as a cache. SQLite is an open source lightweight, transactional SQL database engine which is most noted for its zero-configuration and self-containment abilities (SQLite 2016a). Moreover, it is a server-less implementation which is stored as a file in a device. It is highly compatible with mobile operating systems and of late, has been integrated into the Android Operating System (SQLite 2016b). The aforementioned characteristics influenced the choice of SQLite as a shared cache for this work.

### 3.7 Fault-tolerance in Database As A Service

Cloud databases have recently emerged as one mechanism that delivers database functionality as a service to clients, where the responsibility of database maintenance is taken up by the service provider. Faults leading to cloud database unavailability are observed both on client and server sides. A number of causes may influence server side unavailability such as downtime due to maintenance or updates and too much workload imposed on a server, which has been experienced by enterprises such as Amazon (Gagnaire et al. 2012; Fiedler et al. 2013). Similarly, for client-side unavailability, the most common fault is connectivity outage. As with all services located on remote cloud servers, the necessity is that the Internet link should be present for the client to be able to access the database service. However, due to the instability of connectivity links, the possibility of an Internet link that is up at all times is not feasible. Due to this impossibility, the tendency is to tolerate the aforementioned faults in a DBaaS model, with most efforts being made to address the server side faults. These include a system to improve data availability for cloud databases that is presented by (Neamtiu et al. 2013), where the application's database schema is updated periodically and in a transparent

manner to clients. Database applications periodically undergo schema changes, which usually call for an application to be shut down, leading to critical service interruptions for platforms such as healthcare. Therefore, the on-the-fly schema update introduced by Neamtiu et al allows the database to switch from one schema to another at runtime, accompanied by the shutting down of the database, hence improving data availability to client applications.

Moreover, (Pettersen et al. 2014) present Jovaku, a caching layer for cloud databases. In their work, data requests sent to the cloud database are relayed through the DNS protocol, allowing the returned results to be cached in DNS servers that are close to clients. The caching servers are up and running at all times, both in the presence and absence of the main cloud database, allowing them to be used for reduction of the load imposed on cloud database models while minimising response time for clients. During unavailability of the cloud database, the clients' requests rely solely on the DNS servers for responses. The results obtained indicate only a small percentage of cache misses, which shows the effectiveness of the Jovaku approach.

In the interest of mitigating the Internet disconnectivity fault, (Shanmugam & Maluk 2012) propose a surrogate object mechanism for minimizing latency between mobile clients and the cloud, together with the provisioning of data storage and transaction processing during times of limited connectivity. In this approach called Surrogate Object Based Cloud Caching (SOCC), a surrogate object for each mobile device is created in the cloud, to mimic the behaviour of the actual device in terms of request and response issuing. Client applications store their application data on the surrogate object's cache, so that when Internet disconnections occur, the surrogate object can still perform transactions and store data on behalf of the mobile device.

As mentioned previously, attempts at continuous provisioning of data during the cloud database's inaccessibility periods have been concentrated mostly on the server side.

However, for the client side, little has been done to address DBaaS data provisioning during Internet disconnection faults. This is due to client devices not having enough resources, such as storage memory, to host the cache. Thus, the opportunity to use a cache for continued data availability for mitigating the connectivity outage fault between the client and the cloud server exists, where data will be available on the mobile device only when it is needed. This study further provides a fault tolerance mechanism for DBaaS clients by providing a cache that will periodically prefetch data from the cloud and be available to ad hoc mobile cloud users locally through the usage of GUISET.

### 3.8 Summary

Mobile cloud computing users frequently get disconnected from central servers due to unreliability of wireless links, which results in inaccessibility of the data stored in the cloud. Caching has been used as the mechanism to provide data to mobile users locally, thus maintaining data availability to occasionally disconnected clients. This support comes at a cost, as the cache will have to reside in a mobile device. This might work well when only a small number of services need to be cached, but introduces a problem when more services need to be persisted, due to storage limitation experienced by mobile devices. This chapter essentially looked at different caching approaches for supporting occasionally disconnected clients in cloud-based and mobile infrastructureless technologies. Based on these reviewed approaches, no single solution was observed to provide continuous data availability to disconnected clients with consideration of the following.

I. The properties of an ad hoc mobile cloud environment.
II. The resource poverty challenge of mobile devices from the perspective that some of the mobile devices within the community might not have enough space to hold the cache.

Therefore what is proposed is a shared cache within the AMC-powered GUISET environment where a single node will be used to provision the cache to fellow community nodes.

# CHAPTER FOUR

# FORMULATION OF A CACHE MODEL

## 4.1 Introduction

The previous chapter presented a literature review with the aim of providing motivation for the model to be proposed, which will be used in an AMC environment formed by those disconnected users. This chapter proposes the cache model based on the motivational context provided in Chapter 3. Its components are explained, and their importance highlighted, as stated in the literature. For a cache system to yield better results in terms of the parameters of interest observed, guidelines for managing the cache must be applied. Hence, cache management policies are introduced, along with the presentation of the associated algorithms. Furthermore, to retrieve data to be cached from the cloud DBaaS model a prefetching algorithm is proposed.

Finally, to demonstrate the usability of the proposed cache, a scenario is presented based on the mobile healthcare environment. The remainder of this chapter is organised as follows. Section 4.2 presents and explains the model architecture, together with its components. The cache management policies for governing the proposed cache are explained in section 4.3. Section 4.4 presents the proposed prefetching algorithm for retrieving the cloud database contents, while section 4.5 explains the usage scenario. The chapter is concluded in section 4.6.

## 4.2 Model Architecture

This section presents the cache model that addresses the challenges of Internet disconnections in Ad hoc mobile cloud computing, such that users can continue to have access to their data despite the inaccessibility of the cloud infrastructure. The model diagram is depicted in

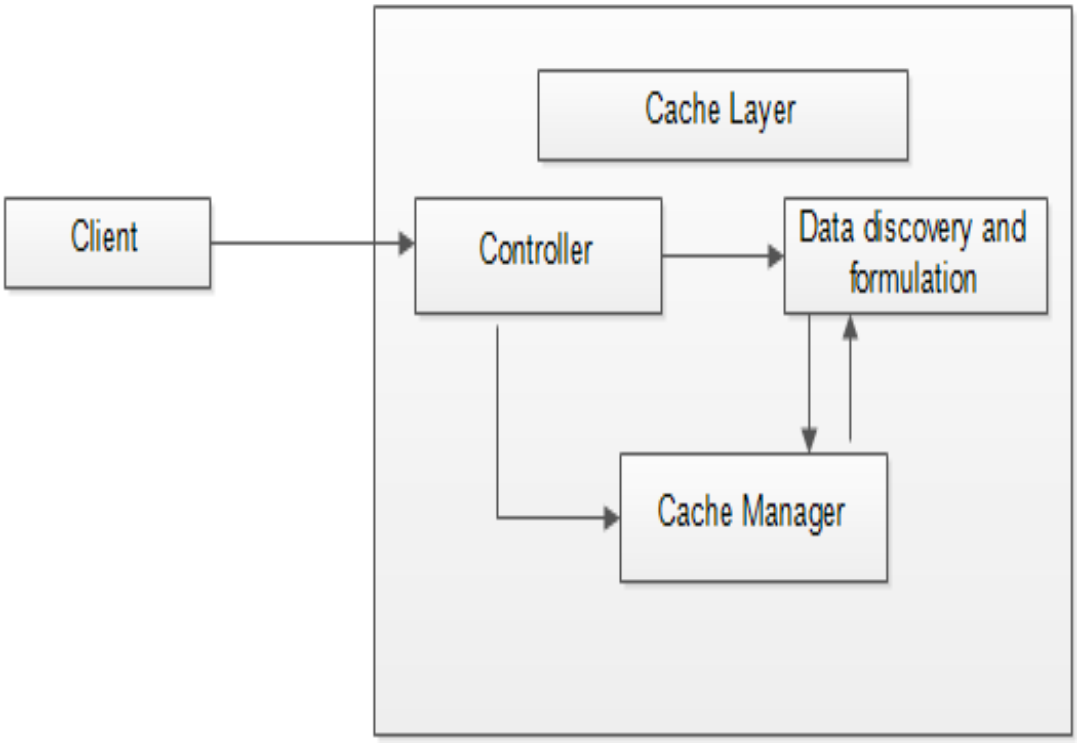



**Figure 4-1: Proposed AMC cache model**

Figure 4-1 and consists of four components: client, controller, data discovery and formulation, and the cache manager. These components work together to present the required data to the end user and also to enable the user to append new data without any notice of the infrastructure being used.

### 4.2.1 Client

A client is any application running on the end user's device that interacts with the cache. For example, an mHealth application used by a healthcare facility's staff for data maintenance.

### 4.2.2 Controller

The controller acts as a security gateway, where authorized and unauthorized users are filtered. It provides two functions (Lopez et al. 2004):

- Authentication – the controller component identifies whether the particular user is who they claim to be based on the credentials supplied.

- Authorization – after authenticating the user, access to certain data that the user is authorized to use is granted.

Authentication and authorization to cached data is a widely recognised problem in mobile ad hoc networks. This is, in most cases, due to the nature of the caching mechanism adopted within the network. Mobile ad hoc networks caching schemes usually involve individual cache maintenance per node and on top of that, allow communication and searching of each other's cache for interesting data if ever their own caches could not satisfy their requests. This introduces high-security threats as each and every node within the network would have to provide security mechanisms for authenticating the neighbouring nodes. Widely used approaches for ensuring security amongst the network nodes involve key management mechanisms (Sen 2010; Ngai & Lyu 2004) and the use of certificates (Xiaodong et al. 2010; Suguna & Subathra 2011), which introduce challenges such as the complexity and computational load imposed on the mobile device's resources (Wu et al. 2007). The proposed model takes the shared cache approach (Houngbadji & Pierre 2010), which maintains a single cache within the network. The proposed shared cache provides authentication at a cache level, as it is the cache that provides the security restrictions towards accessing the cached data. This improves the security of the system as a whole which works in favour of many applications, particularly mHealth applications, as they recognise security as one of their important aspects (Avancha et al. 2012). A flow chart representation of this component is depicted in Figure 4-2.

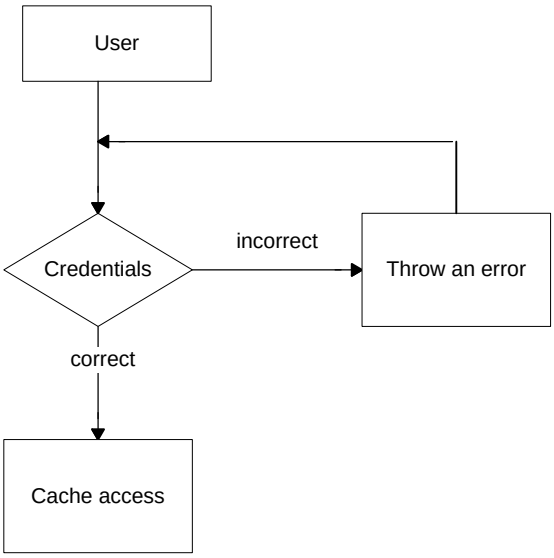



**Figure 4-2: Flow chart for the controller component**

Although this work considered security aspects for the proposed model, we do not delve deeper into them as they were not the main focus of the work.

### 4.2.3 Data Discovery and Formulation

The data discovery and formulation component works in two scenarios, as shown in Figure 4-3 and explained below.

**Scenario 1**

The first scenario is for discovering data requested by a client. When a client issues a request for particular data, this component analyses the request to determine if it is issued in the correct format, e.g., it analyses the format of a particular SELECT query that was issued. If the query is in order, the request is passed to the cache manager to ascertain the availability of the requested data; otherwise, an exception is raised. For every query issued, the probability of it being successful is called a *cache hit*, and the probability of it being unsuccessful a *cache miss*. These probabilities can be expressed statistically as follows:

$$P(hit) = 1 - P(miss) \qquad (1)$$

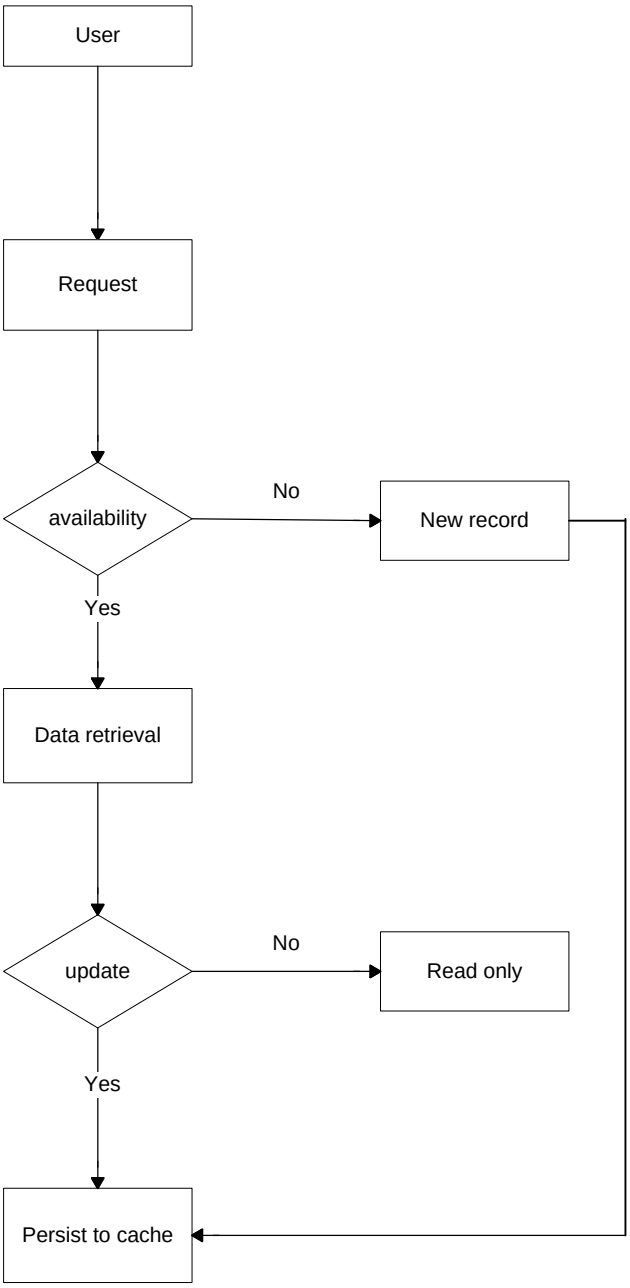



**Figure 4-3: flow chart for data discovery and formulation component**

And consequently,

$$P(miss) = 1 - P(hit) \qquad (2)$$

**Algorithm 1:** Read

**Require**: *param* ≠ null

**Ensure** *: record* is returned

**for**(*param* ≠ null)

//check if the parameter is available in the cache

    **if** (*param.isAvailable(true)*)

    //return record associated with the parameter

        **return** *record*

    **else**

        **return null**

    **end if**

**end for**

**Table 4-1: Algorithm for read operation**

The decrease in the probability of cache misses results in an increased probability of cache hits. This inverse proportionality relationship between these probabilities is what this work seeks to achieve, that is, this work aims at increasing the probability of cache hits, thereby minimising the probability of cache misses. An algorithm for performing read operations in a cache is presented in Table 4-1.

**Scenario 2**

The second scenario is for writing data to a cache. This is done when either creating a new record or updating an existing one. After verifying the validity of the query, the request is passed to the cache manager for persisting data; otherwise, the component raises an exception. Algorithms for creating new records in a cache or updating existing one are shown in Table 4-2 and Table 4-3 respectively.

**Algorithm 2:** Create

**Require**: *param* ≠ null

**Ensure** *: record* is persisted

**for**(*param* ≠ null)

//check if the parameter is available in the cache

**if** (*param.isAvailable(true)*)

//notify the user

else if (*param.isAvailable(false)*)

//persist the record in the cache

**persist**(record)

//return the persisted record

**return** *record*

**else**

**return null**

**end if**

**end for**

**Table 4-2: Algorithm for creating new data**

### 4.2.4 Cache Manager

All the operations performed in a cache by the aforementioned components are issued against the cache for responses, if any. The role of this component is to manage the data in a cache together with the access privileges to that data. For the controller component, the cache manager validates if the person requiring access to the certain data is indeed permitted to utilise that data. If true, the access is granted to utilise the cache, restricted to only the data that is associated with them. Similarly, for the data discovery and formulation component, for every data request issued, cache lookup is done to check if the requested data is available. If true, it is returned; otherwise, a response is sent back to report on the unavailability of the data.

**Algorithm 3:** Update

**Require**: *param* ≠ null

:newValue≠null

**Ensure** *: record* is updated

**for**(*param* ≠ null)

//check if the parameter is available in the cache

**if** (*param.isAvailable(true)*)

//return record associated with the parameter

**return** *record*

//update *record* using the new parameters

//assign this value to a new object *newRecord*

newRecord = record.update(newValue)

//return the newly persisted record

Return newRecord

**else if** (*param.isAvailable(false)*)

**return null**

**end if**

**end for**

**Table 4-3: Algorithm for updating cached data**

For appending new data, the sent request containing the parameters to be saved in a cache is used to search through the cache if the similar record exists. When updating an already existing record, the new parameters are merged with the old ones; where there are differences in parameter values, the old data is replaced with the new data. Also, synchronization of data between the cache and the cloud server is managed by the cache manager component.

To maintain the cache manager functionalities mentioned above and others such as evicting some data from the cache to make space for new data, a number of policies are used. We refer to these policies in the next subsections.

**Require**: response *s*, last occupied count *c*
**Ensure**: *s* is persisted in a cache
**for** $s \neq$ null
    get *c*
    persist *r* in *c+1* cell
**end for**

**Table 4-4: cache placement policy algorithm**

## 4.3 Cache Management Policies

For a cache model to function in the best possible way, a set of principles governing the decision-making processes on admission and eviction of data must be applied to a cache manager component. Cache placement and replacement policies are examined, together with how they are applied in the proposed cache.

### 4.3.1 Cache Placement Policy

Cache placement policies generally deal with where to place data (Jouppi 1993). The most common method under this policy is to append data at the next available block. Also, the decision of which part of the incoming data to persist, whether from the client or from the main server (mostly in the case of update operations), is made by this policy (Palpanas et al. 2001). The proposed cache adopts the aforementioned methods, i.e., every incoming data is placed at the next available block towards the end of the cache. Given that the count of the last item in a cache is *n*, the next item will be appended at the *n+1* block. A cache placement algorithm is shown in Table 4-4

### 4.3.2 Cache Replacement Policy

When a cache reaches its maximum capacity due to cached data, the decision on which data items to be evicted from the cache has to be made. This is the main functionality of a cache replacement policy: to decide on the data to be removed from the cache so as to make room for new data (Podlipnig & Böszörmenyi 2003). To achieve data replacement, a number of strategies have been widely used, mainly Recency-based strategies such as the Least Recently Used (LRU) and Frequency-based strategies such as Least Frequently Used (LFU).

With the LRU strategy, an object that is referenced the least recent is removed from the cache. To implement LRU, a list of the objects in a cache is created. Every object being referenced is removed from its current position and inserted at the top of the list. Eviction is done at the bottom of the List. In LFU strategy, data that is least frequently requested is removed from the cache. Its implementation involves maintaining the count of the number of times an object has been requested. The one that has the lowest count is removed from the cache (Zahran 2007; Zebchuk et al. 2008).

For Mobile Cloud applications these strategies need to be altered to suit the nature of the application whose data is cached accordingly (Khandekar & Mane 2015). Moreover, they need to be implemented in a less resource-intensive way for mobile devices (Fathy et al. 2008). However, for the purpose of this study, the usage of an LFU cache replacement is assumed for the following reasons

I. the cache is used in an AMC scenario, which is formed and used only during Internet disconnections
II. the disconnections are assumed to last for shorter durations

Hence, the usage of a cache to its maximum capacity is not feasible.

| | Strategy | Aim | Example |
|---|---|---|---|
| **Cache replacement policy** | Frequency-based strategies | data that is least frequently requested is removed from the cache | Least-Frequently Used (LFU) |
| | Recency-based strategies | object that is referenced the least recent is removed from the cache | Least-Recently Used (LRU) |

**Table 4-5: Cache replacement strategies**

Table 4-5 summarizes and gives examples of cache replacement strategies.

### 4.3.3 Synchronization

Part of the responsibility taken up by the cache manager is to ensure synchronization between the cached data and the cloud. By definition, synchronization occurs when changes made locally to the cached data are propagated to the main server. For read-write caches, synchronization is an essential mechanism as it maintains data consistency between the main server and the cache storage by ensuring that the main server is updated with the latest changes accordingly (Kim 2006; Singh & Dutta 2013). In ad hoc mobile clouds, synchronization is bound to occur when Internet connectivity is restored. However, this work did not focus much on synchronization as its main goal was to ensure data availability to AMC users during Internet disconnections.

## 4.4 Prefetching Mechanism

To periodically hoard the content of the proposed cache, we make use of the prefetching mechanism as discussed in chapter 3, which is presented in Table 4-6.

**Require**: request $r \in R$, response $s \in S$, response $s \in S'$

**Where**: R is the set of requests to be sent

S is the set of responses returned

S' is the set of responses with date and time parameters

*t* is current time

*d* is current date

**Ensure**: *r* is sent to the server

*s* is persisted in a cache

**while** $r \neq$ null **do**

send *r* to the server

**return** corresponding *s*

persist *s* in a cache

**if** , $s \in S'$ **then**

check *s* relative to *t*

check *s* relative to *d*

if time in $s > t$ && date in $s == d$

persist *s*

**else**

move to next *s*

**end if**

**end if**

**end while**

**Table 4-6: Proposed prefetching mechanism algorithm**

This is transparently done during times of high Internet availability through one of the mobile devices within the community. The selection of data to be cached is based on its usability in the near future. Unlike most prefetching methods that generally use the user preferences and history as defining parameters for data to be pre-fetched, we adopt the approach by (Pendse & Ksitta 1999), where the preference of data to be pre-fetched is specified by the user. This is essential for mobile cloud applications such as mHealthcare (Saleh et al. 2012), where data to be used in the near future is not necessarily determined by the usage history but is preferably specified by the client. Moreover, our approach to prefetching uses time as the parameter to decide on data to be hoarded. To demonstrate the usability of the cache, we paint a scenario

in an mHealthcare environment in the following section. We develop our prefetching algorithm based on this scenario.

### 4.5 Cache Host Selection

In order for the cache to be available within the AMC-powered GUISET environment, one device has to be selected as the host. That is, a mobile device which best qualifies to be the host is selected. To achieve this, a mechanism to check and compare mobile devices based on certain metrics such as the mobile device's resources is needed. Selection schemes have been proposed to select the best qualifying node in mobile infrastructureless networks on certain bases, which include:

**Connectivity-based selection**: within the network, the node which favours the maximum number of neighbouring nodes is selected to be the host. That is, the node to which the shortest distance is required to connect to, is the one that becomes the hosting node (Preetha & Chitra 2014).

**Mobility-aware selection**: this approach selects the most stable node to be the host based on its speed and variance. That is, the node with the minimum running speed and variance within the community is selected, as it is the less mobile node and hence implies more stability (Basu et al. 2001).

**Low cost maintenance selection**: the mobile node with the lowest ID within the network is selected to be the host. Each mobile node keeps the set of all the neighbouring nodes' IDs, together with its own ID. The one found to have the lowest ID is declared to be the host (Bentaleb et al. 2013).

**Power-aware selection**: this scheme mainly uses energy level to determine whether the particular node can be a leader or not, and the amount of storage available to be able to store

information about the other nodes within the network. The node with the highest energy level and enough storage memory is selected. Moreover, direct connection is used between the hosting node and the client node, to further conserve energy for client nodes (Fathi & Taheri 2010; Hussain et al. 2013).

Combined-weight based selection: this approach takes into consideration a number of other selection approaches in order to select the best qualifying node. Approaches such as the aforementioned ones are all evaluated so as to select the most suitable node, that is, the node which is connected to most neighbours, is less mobile and has a higher energy level (Tolba et al. 2007; Agarwal & Motwani 2009).

This work adopts the Combined-weight based selection scheme for selecting the best qualifying node to host the cache in the AMC environment. Figure 4-4 depicts the mechanism used for selecting a mobile node to host the cache. For situations where two or more mobile nodes have equal levels of resources, the one to host the cache is selected randomly.

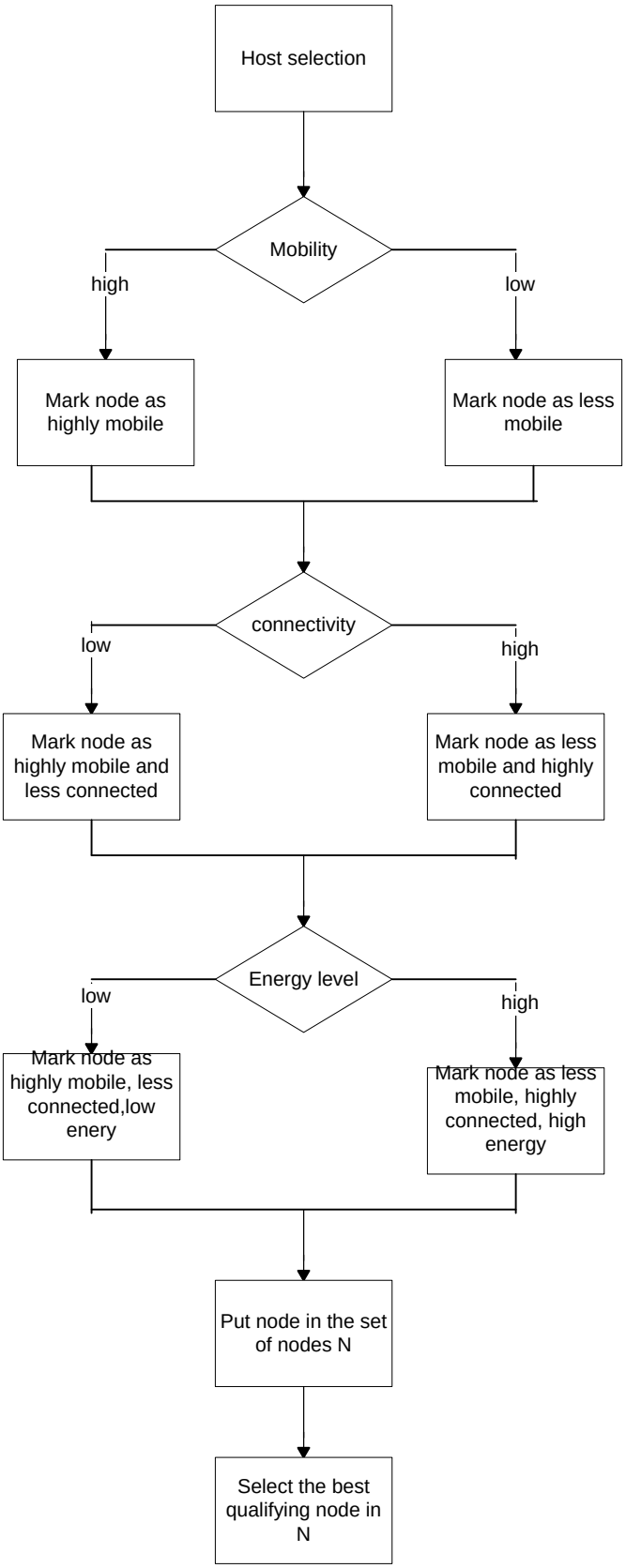



**Figure 4-4: cache host selection mechanism**

### 4.6 Usage Scenario : mHealthcare

In a hospital, a doctor wants to check his next appointment. While trying to access the mobile application to retrieve the data from the cloud, he finds that the Internet is down. He then joins the ad hoc cloud formed by the disconnected nodes within the hospital. Amongst these mobile devices the most stable one is selected to host the cache provided through GUISET, based on the cache host selection mechanism. Assuming that the doctor's device has enough resources to invoke the cache including going through the security checks which are provided by the cache and finally requesting the data, the results are then returned. The doctor may then perform further operations in the returned data such as cancelling a particular appointment, with those changes being propagated back to the cache. The major design goal is for the application to continue functioning during Internet disconnections, with an increased data availability rate, as though the application is connected to the cloud database. Figure 4-5 depicts an abstract representation of this scenario.

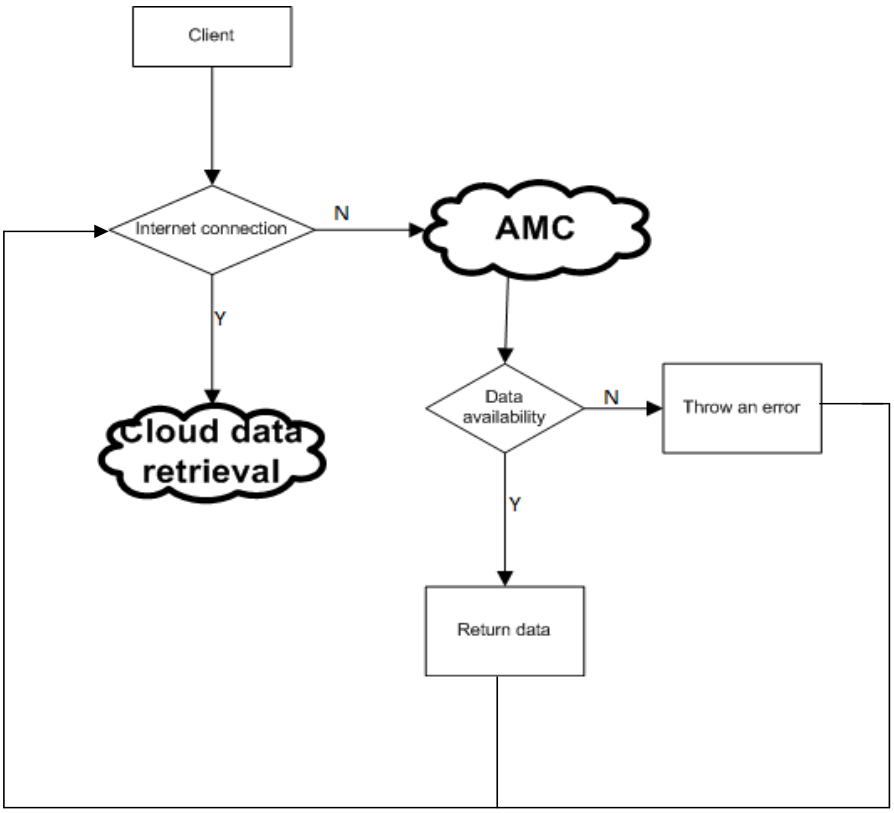



**Figure 4-5: Flowchart diagram for mHealthcare scenario**

The design of this prototype consists initially of a DBaaS model, where the records are kept. This DBaaS model has five database tables: Employees, Login, Patients, Appointments and Diagnosis, which are explained below.

**Employees** table consists of the details of the hospital's employees such as addresses, contacts and their respective positions.

**Login** table stores the employees' credentials for accessing the database.

**Patients** table stores details about the hospital's patients. It comprises personal details, residential addresses, contacts and medical history.

**Appointments** table consists of the appointments that the health practitioners have with their patients. Details such as the time, date and reason for appointment are stored in this table.

**Diagnosis** table comprises different illnesses and their treatment methods.

During prefetching, the contents of these tables are retrieved and stored on the cache, which also consists of the same tables. Data retrieval occurs as defined by the prefetching mechanism. During disconnections, the application directs its requests to the cache.

Figure 4-6 presents a use case diagram illustrating how the cache prototype works. The system consists of one actor, which is a hospital employee. The employee has to first login to the system to be able to perform further actions. Requests issued by the application to a cache are in the form of a query. During the login session, the controller component is responsible for authenticating the user, and granting access if the user is legitimate. After login in, the user is presented with an interface with options to choose from. After selecting one of the options, e.g., to search for a particular patient, the data discovery and formulation is invoked, which fetches data from the cache and presents it to the user. If any modifications were made on the data, a cache manager component is then responsible for replacing the old data with the new data.

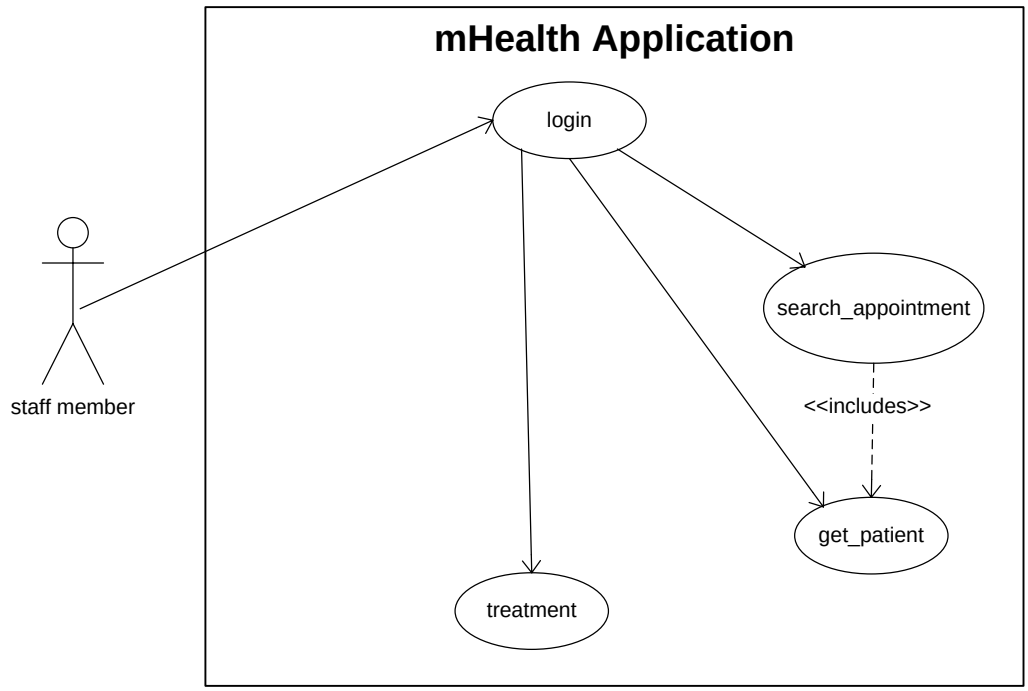



**Figure 4-6: use case diagram for the mHealthcare scenario**

## 4.7 Summary

In this chapter, the proposed cache model to support disconnected clients in an ad hoc mobile cloud environment was formulated and explained, together with its different components and their interactions. We further proposed policies to assist in the management of the cache and its data. To retrieve contents of the cache from the cloud DBaaS model, a prefetching algorithm was proposed, which takes into account the relevance of the data based on the time of its usage. To demonstrate the functioning of the proposed cache model, an mHealthcare scenario was formulated and explained. The next chapter will demonstrate the performance of this model based on the formulated usage scenario.

# CHAPTER FIVE

# MODEL PROTOTYPING AND EVALUATION

## 5.1 Introduction

The main goal of this work is to propose a caching model to be used to support data availability to occasionally disconnected clients. Chapter 2 presented GUISET integration requirements in AMC environments, with chapter 3 providing the recommendation on the cache model to be provided in these environments. Based on these, the previous chapter formulated the proposed model and described in detail its components. Also, the mHealth usage scenario was presented to show the usability of the proposed cache during disconnections in mobile healthcare environments.

This chapter presents the prototype and evaluation of the proposed caching model described in the previous chapter, through the usage of the mHealthcare scenario. The remainder of this chapter is organised as follows. Section 5.2 presents the detailed design of the cache prototype. Simulation setup for both cloud and ad hoc environments are explained in sction5.3. Section 5.4 presents the results from the conducted experiments while section 5.5 discusses the obtained results. Finally, section 5.6 summarises the chapter.

## 5.2 Cache Prototype Design

In the previous chapter, the cache model was proposed and each of its components explained in detail. This section focuses on showing how these components can work together to provide data to end users during disconnections. The use case model, sequence diagram and class diagram are presented to best illustrate the collaboration of these components, through the use of the mHealth use case scenario presented in the previous chapter.

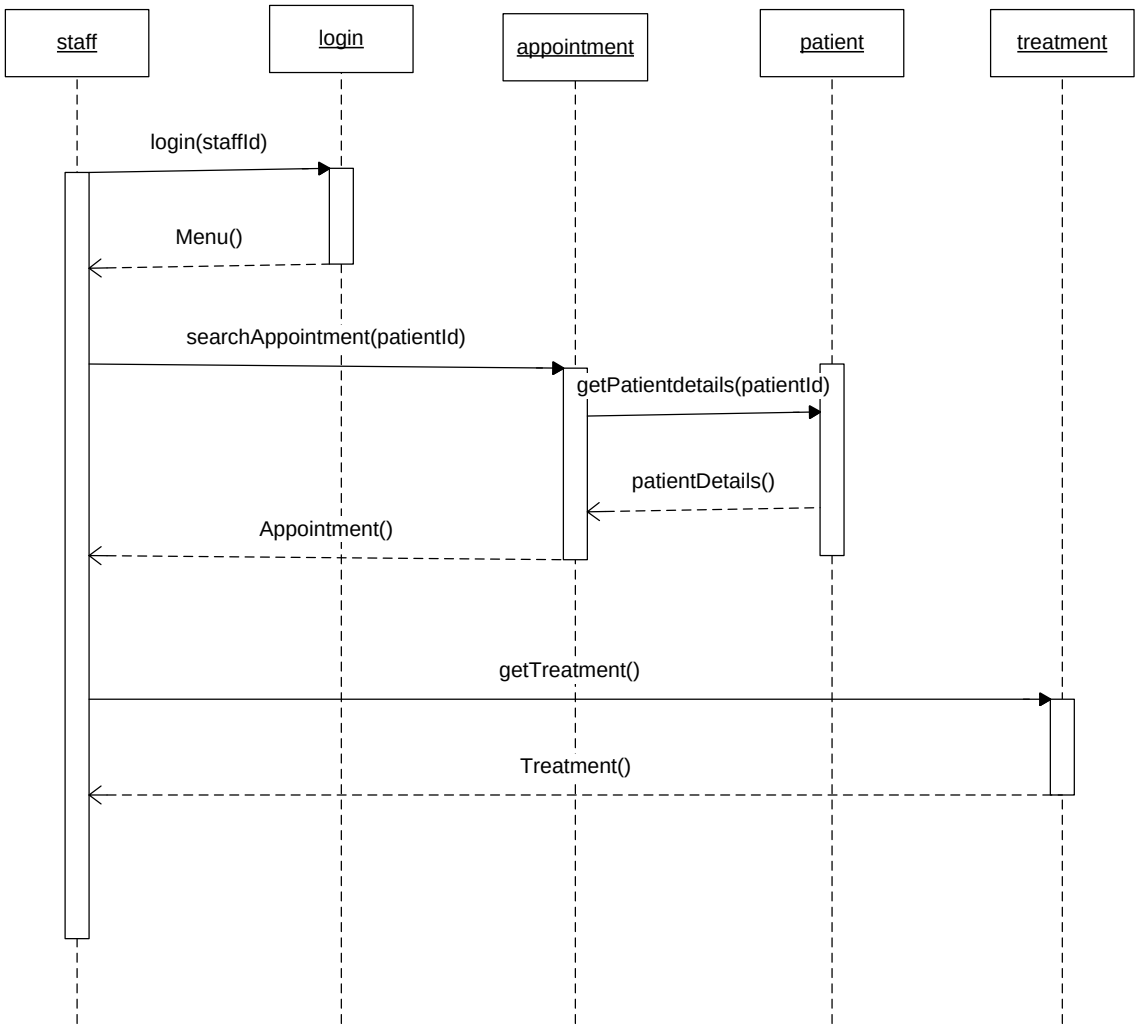



**Figure 5-1: sequence diagram for the mHealthcare scenario**

In Figure 5-1, the flow of messages passing through the system components sequentially is shown through a sequence diagram. The messages are initiated by an employee through the login interface. After the user is presented with the interface to choose from, a query to search for an appointment is issued. For the results to be returned, another message to get the details of a patient is sent to the patient component. After returning the response containing patient details, the appointment details are then sent to the user. The data discovery and formulation component of the cache manager is the one responsible for searching and returning the

required details. Similarly, if any medication is required for the patient, the message is sent, to which the response is the medication.

A class diagram representing the structure of the system through classes and objects, together with the relationships amongst them is shown in Figure 5-2. The system consists mainly of two types of classes: the Plain Old Java Objects (POJOs) and the classes implementing the main functionality. The functionality classes consist of two principal methods: the invokeWS() method and the invokeCache() method. The first method takes care of the invocation of a web service hosted on the cloud when there is Internet connectivity. When the Internet connectivity outage situation strikes, the invokeCache() method is invoked. It provides access to the data stored locally on the cache, mimicking the behaviour of the invokeWS() method.

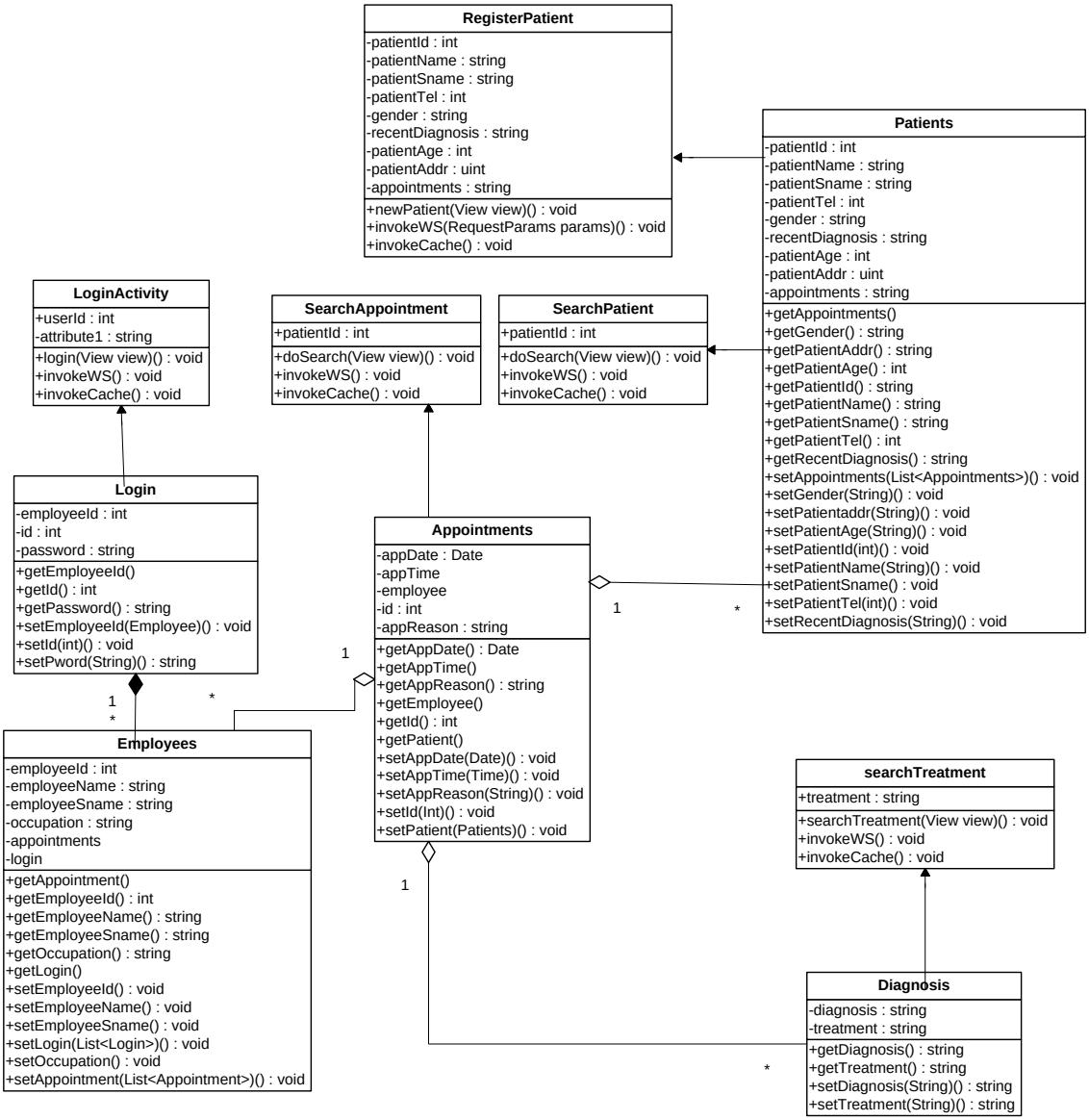



**Figure 5-2: class diagram for mHealthcare system**

## 5.3 Simulation Environment Setup

The simulation environment setup was divided into two parts: the cloud environment where the DBaaS is hosted, and the ad hoc mobile environment that utilises the cache. The cloud environment was used for prefetching purposes, in order to get data to be persisted in the cache. Web services were developed to test the usability of the developed application during connections. This section describes these components in detail.

### 5.3.1 Cloud environment setup

#### 5.3.1.1 MySQL on Amazon RDS

The Amazon Relational Database System (RDS) was used as the main cloud database for the application. MySQL is the world's most popular and widely used open source Relational Database Management System (RDBMS). On top of inheriting the SQL databases' properties mentioned in chapter two, it is also reported to be a secure, scalable, platform independent and easy to use database system. The Amazon RDS DBaaS model was accessed through the address:

mhealthapp.c2imaydr797p.us-west-2.rds.amazonaws.com:3306

#### 5.3.1.2 Eclipse and Tomcat

For mobile devices to communicate with backend servers such as MySQL, web services are used. The mobile device calls a web service, which in turn calls the server. Consequently, the server sends the response to the web service, and the web service forwards it to the mobile device. For this purpose, the Eclipse IDE was used to implement RESTful web services to be used by our mobile application to communicate with RDS. The web services were deployed in Tomcat container, hosted in Amazon EC2. These concepts are explained below, together with the Jersey framework used for RESTful web services development.

##### 5.3.1.2.1 Eclipse IDE

Eclipse is an open source Integrated Development Environment (IDE), primarily developed in Java but supporting applications' development and testing in other programming languages by using plugins. It was introduced in 2001 when IBM donated source code worth $40 million to the newly formed Eclipse consortium. Eclipse mainly uses its Software

Development Kit (SDK), which comprises the Java compiler to compile the applications and the Java source files for code analysis. Eclipse has a number of different platform releases and, for the purpose of this work, the Juno release was used, as it is one of the releases that best support web services development. Web services were developed locally on a desktop and then deployed as a WAR file in Tomcat running in the Amazon EC2 instance.

#### 5.3.1.2.2 Jersey (JAX-RS)

Jersey is an open source web service framework for implementing RESTful web services in Java, developed by Oracle. It provides support for JAX-RS APIs (API for RESTful web services) and JSR 311 (support for annotations), and assists in simplifying the development of RESTful services and clients. This work uses version 2.22.1 of the Jersey framework.

#### 5.3.1.2.3 Tomcat server

Tomcat is an HTTP application server developed by the Apache Software Foundation, initially released in 1999. Tomcat provides support for Java EE specifications such as servlets and JSPs. Furthermore, it can also be used as either a standalone web server in which Java code runs or can be integrated with other web servers such as the Apache web server. Since Tomcat handles HTTP requests over the HTTP transport protocol, web services are easily deployed in Tomcat as they are based on HTTP methods. Tomcat version 7.0.45 was used for the deployment of web services.

#### 5.3.1.2.4 Amazon EC2

Amazon provides the platform for scalable computing capacity in which users can create and use virtual computers to develop and deploy their applications faster, namely, Amazon Elastic Compute Cloud (EC2). This is achieved through an Amazon Machine Image (AMI), which uses a web service to configure the user's virtual environment (instance) consisting of

the operating system and any desired software. For this work, an EC2 instance for hosting the Tomcat server was created. The server was accessed through the address:

http://ec2-52-33-170-59.us-west-2.compute.amazonaws.com/8080

### 5.3.2 Ad hoc mobile cloud environment

For performing the experiments to show the usability of the proposed cache, the mHealth use case scenario explained in chapter four was implemented. We configured an environment consisting of five devices: four mobile devices and a laptop to mimic the ad hoc mobile cloud environment. The mobile devices were HUAWEI Ascend Y330, Dual-core processor @ 1.3 GHz and 512M RAM each, running the Android 4.2 Jelly Bean operating system. The laptop was an ASUS Intel Core processor with an i7 CPU @ 1.8GHz and 8GB RAM, running the Windows 8 operating system. The devices were connected in an ad hoc manner via the IEEE 802.11b wireless standard. The laptop was used as the main node with enough resources to host the cache to be shared amongst the nodes in an ad hoc mobile cloud, also running the mHealth application. The contents of the cache were the database tables presented in the previous chapter. The mobile devices were used as the client devices running the mHealth application. Apache Jmeter was used to measure response time and throughput from the cache with increasing load. This setup is illustrated in Figure 5-3.

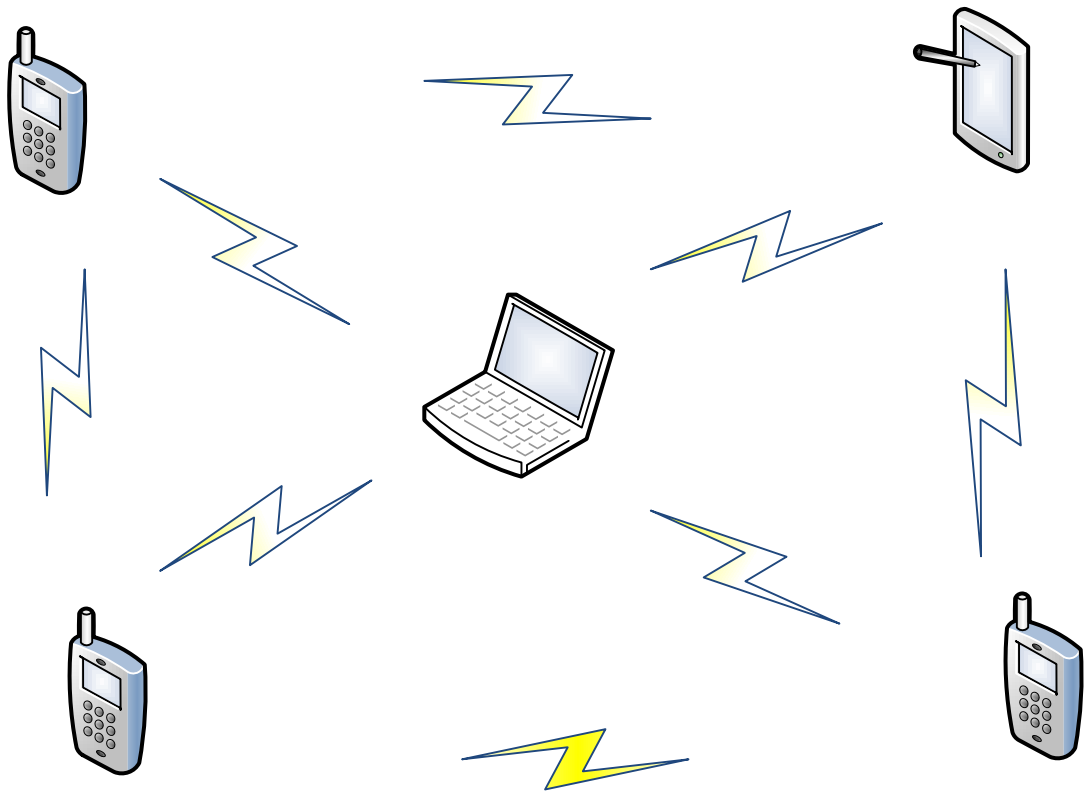


**Figure 5-3: AMC environment setup**

### 5.3.3 Experimental Design

#### 5.3.3.1 Data availability

The initial experiment measures the availability of data when requested/needed during the periods of Internet disconnections. The availability is measured for each database table. To achieve that, the random sampling technique presented by (Olken 1993) is used. In this technique, a subset S' $\epsilon$ S consisting of a certain number of parameters is randomly selected from the superset S to test the availability of the system under randomised samples. More specifically, Given S = {$N_1$,$N_2$, $N_3$,.., $N_j$},

The subset:

S' $\epsilon$ S is selected

Where: S is the collection of all the datasets

S' consists of the randomly selected datasets

20 attributes were randomly selected from each table from the main cloud database. Requests for data associated with these randomly selected attributes are issued in the cache, through an application's interface. The number of successfully serviced requests on the cache (i.e., the requests that returned responses that are not empty) was observed. The experiments were conducted 20 times for each table, and the results were recorded in a table

#### 5.3.3.2 Performance Testing

The second set of experiments conducted observe the performance of a cache under a varying number of user requests. The performance is measured for read and write requests, using two performance metrics, namely, response time and throughput. The metrics are explained below.

**Response Time**

Response time is defined as the time it takes for a system to return the requested response to the client. It is measured as the time between the sending of the request and the receival of the response between the client and the server. This relationship is presented by the equation below:

Average response time = $\frac{1}{n}\sum_{i=1}^{n}(T_f - T_0)i,$

Where: *n* represents the number of requests

$T_f$ represents the time of receiving the response

$T_0$ represents the time when the request was sent

$i$ represents the request's index

Response time was measured to show how well the cache responded to increasing number of requests sent concurrently.

**Throughput**

Throughput presents the number of transactions processed per given time. In this work, throughput was measured to observe how many transactions the cache can process per second with an increasing number of requests issued concurrently. This relationship is given by the formula below:

Average throughput = $\frac{1}{n}\sum_{i=1}^{n} TT$

Due to the increase in the number of requests from users to a point of exceeding the physically available nodes, Jmeter was used to simulate the increasing number of requests. For each number of user requests, the experiment was done ten times and the average taken.

## 5.4 Experimental Results

The proposed cache was evaluated under the centralised approach of the ad hoc mobile cloud test case, described in Chapter 2. We performed two experiments to test the availability of the cache and its performance under differing load. The results of the experiments are examined below.

### 5.4.1 Data Availability

For the first experiment, the results were as shown in Table 5-1 below. The resulting graph is presented in Figure 5-4. For the first two tables (Employees and Login), all the requests issued were successful. This behaviour was expected due to the caching mechanism explained in the previous chapter which enables prefetching of all the data contained in the

tables associated with employees. For the Patients table, 85% of the requests were successful, with the Appointments table successfully responding to all the requests. This means that all the appointments were pre-fetched. Lastly, the Diagnosis table serviced 75% of the issued requests.

| Table | No. of successful requests | Percentage |
|---|---|---|
| Employees | 20 | 100% |
| Login | 20 | 100% |
| Patients | 18 | 85% |
| Appointments | 8 | 100% |
| Diagnosis | 15 | 75% |

**Table 5-1: data availability rate**

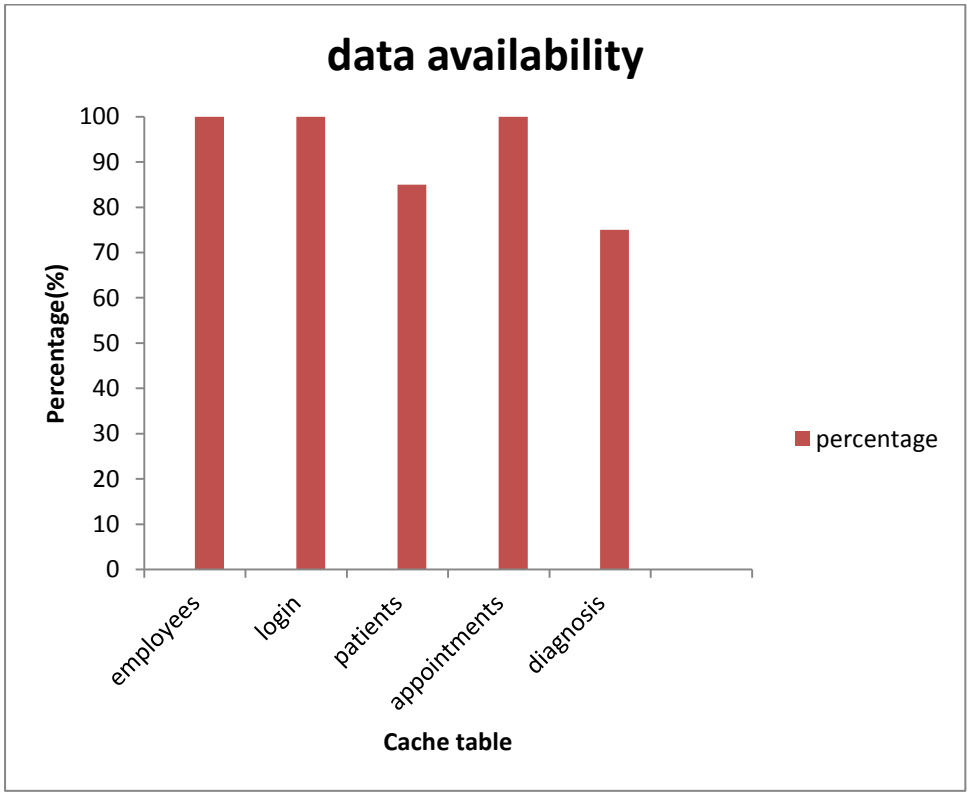



**Figure 5-4: data availability graph**

### 5.4.2 Experiment Two : Performance Under Read Operation

#### 5.4.2.1 Response Time

For the second experiment, we examined the amount of time the cache would take to respond to read requests under a certain load from concurrent users. The recorded results are shown in table 5-2. Figure 5-5 demonstrates the performance of the cache with regards to an increasing number of requests for read operations. The graph shows that the time it takes for the cache to fetch requested data increases with an increasing number of requests.

| No. of requests | Response Time (ms) |
|---|---|
| 50 | 124.5 |
| 100 | 130.3 |
| 150 | 138.1 |
| 200 | 164.9 |
| 250 | 182.7 |
| 300 | 194.4 |
| 350 | 202.3 |
| 400 | 217.6 |
| 450 | 230.0 |
| 500 | 228.4 |

**Table 5-2: response time for read operations**

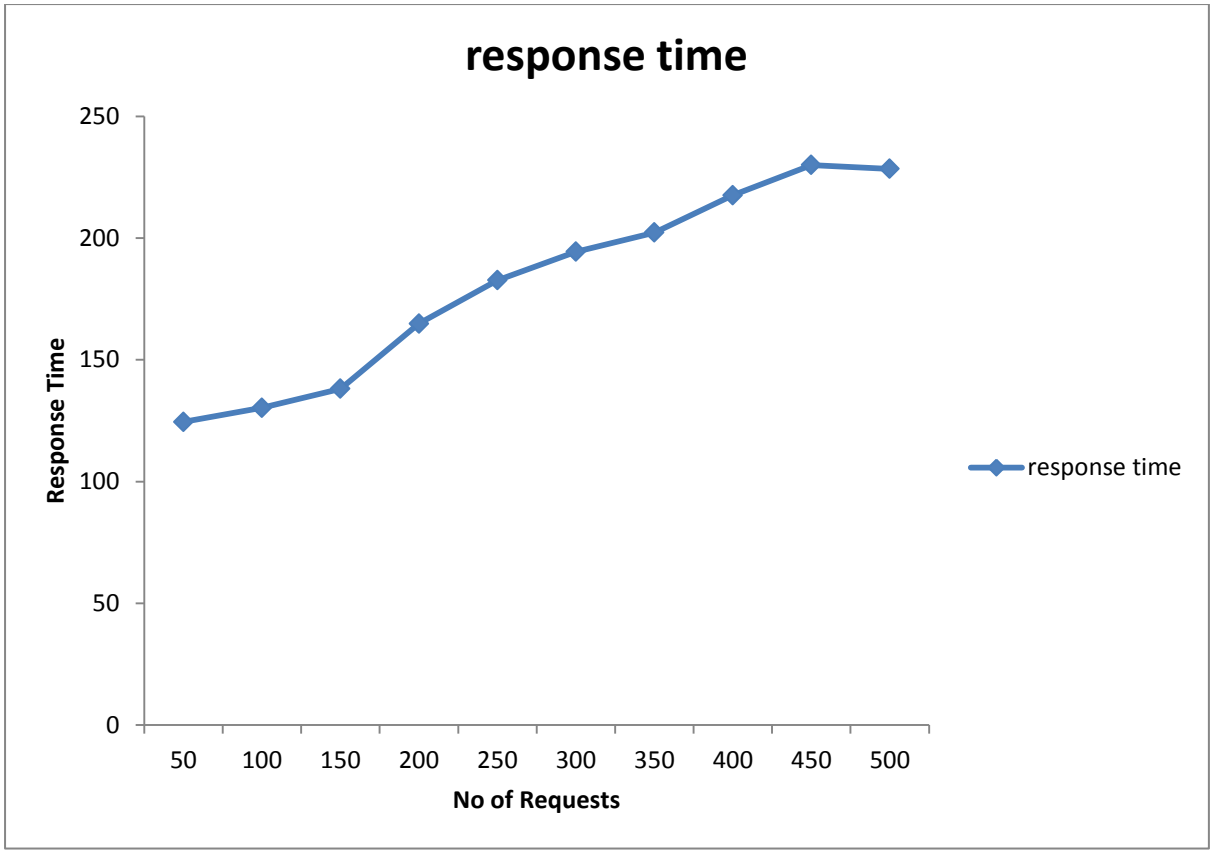



**Figure 5-5: response time vs. No of requests graph**

### 5.4.2.2 Throughput

Table 5-3 shows the recordings for throughput for read operations; the resulting graph is shown in Figure 5-6. The graph shows the increase in throughput as the number of requests increases. Upon reaching 400 requests, throughput starts to decrease.

| No. of requests | throughput |
|---|---|
| 50 | 82.3 |
| 100 | 96.0 |
| 150 | 108.6 |
| 200 | 122.2 |
| 250 | 141.7 |
| 300 | 158.2 |
| 350 | 164.5 |
| 400 | 173.3 |
| 450 | 156.9 |
| 500 | 147.0 |

**Table 5-3: Throughput for read operations**

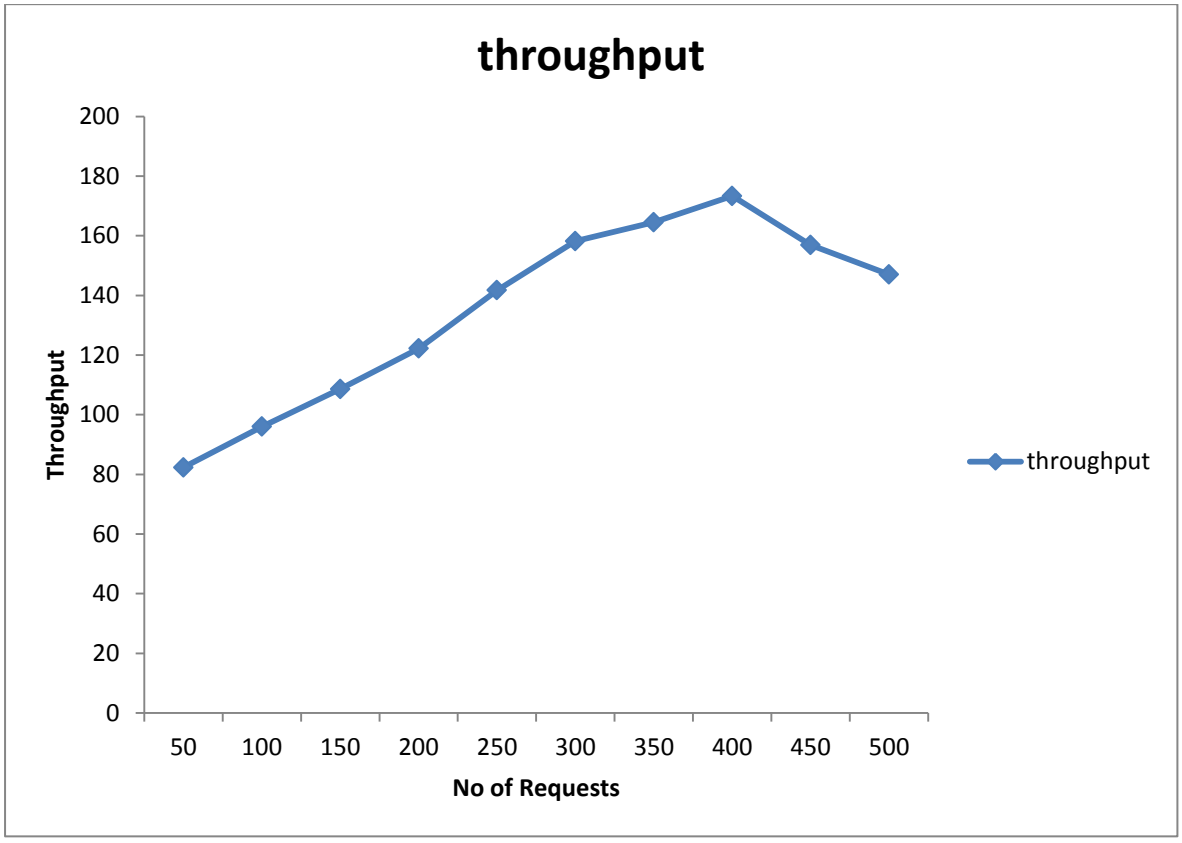



**Figure 5-6: throughput vs. No. of requests graph for read operations**

### 5.4.3 Experiment Three : Cache Performance Under Write Operations

We also measured how the system performs when write operations are issued by a different number of users. Table 5-4 and Figure 5-7 show in tabular form and graphically the response time of the system, while table 5-5 and Figure 5-8 show throughput, respectively. Both the response time and throughput increase with the increase in the number of user requests.

#### 5.4.3.1 Response Time

Response time recordings under write operations are shown in Table 5-4, with the resulting graph in Figure 5-7.

| No. of requests | Response time (ms) |
|---|---|
| 50 | 140.9 |
| 100 | 205.6 |
| 150 | 337.3 |
| 200 | 481.3 |
| 250 | 638.0 |
| 300 | 722.7 |
| 350 | 842.2 |
| 400 | 1173.0 |
| 450 | 1432.2 |
| 500 | 1693.4 |

**Table 5-4 : Response time for write operations**

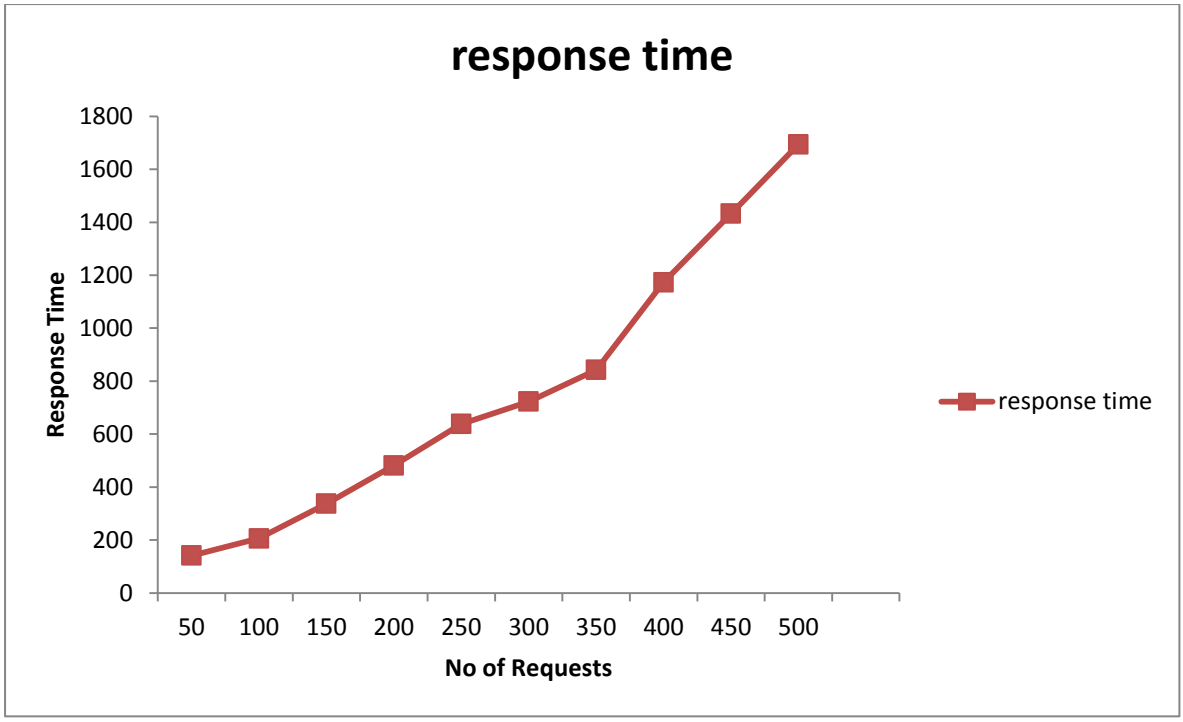



**Figure 5-7: response time vs. No. of requests graph for write operations**

### 5.4.3.2 Throughput

The cache's throughput under an increasing number of requests for write operations was also measured as shown in Table 5-4. Figure 5-8 shows the resulting graph from these recordings

| No of requests | Throughput |
|---|---|
| 50 | 70.6 |
| 100 | 89.2 |
| 150 | 110.2 |
| 200 | 133.9 |
| 250 | 192.1 |
| 300 | 236.5 |
| 350 | 298.4 |
| 400 | 340.8 |
| 450 | 382.3 |
| 500 | 404.1 |

**Table 5-5: Throughput for write operations**

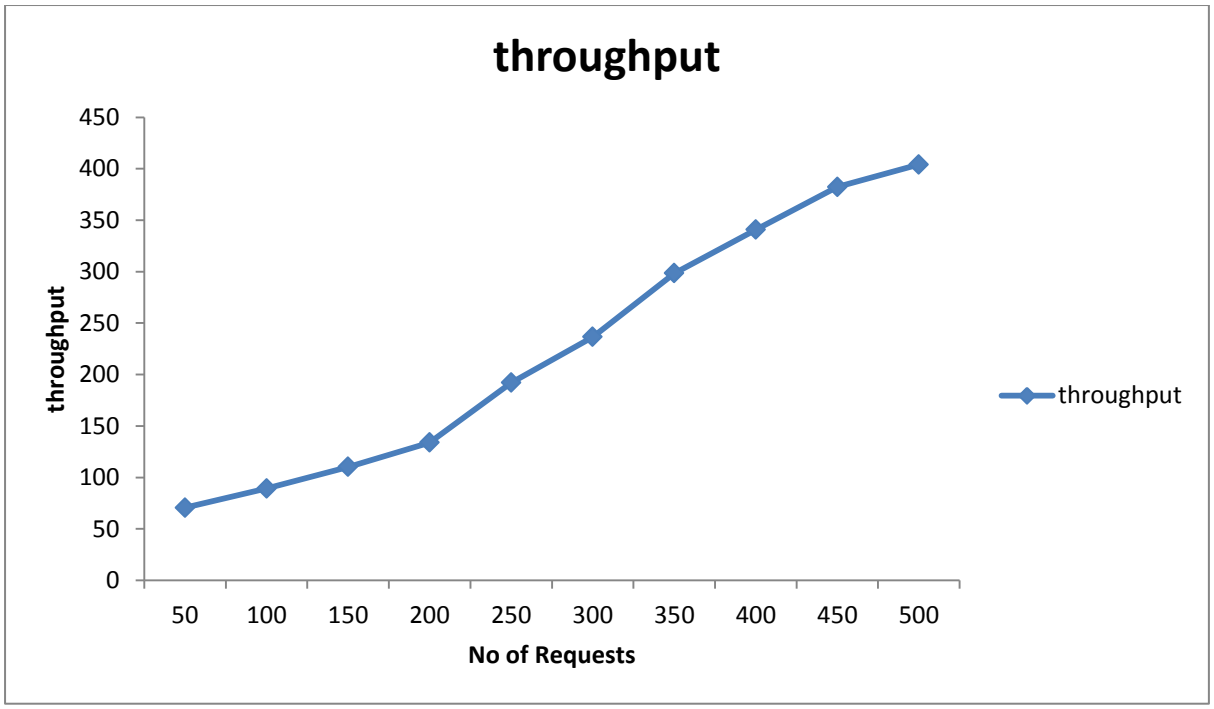



**Figure 5-8: throughput vs. No. of requests graph for write operations**

## 5.5 Discussion of Results

Figure 5-4 shows the availability of data in each cache table. Employees, Login and Appointments tables maintained 100% hit rate. This means that the three tables were able to maintain data availability to mobile clients during disconnection all the time. Following were the Patients and Diagnosis tables, with 85% and 75% hit rates respectively. This indicates their failure to provide requested data 15% and 25% of the times. This was due to randomly selecting requests, resulting in selection of the requests for the data that was not going to be needed in the near future and hence was not pre-fetched. Taking the average hit rate, the cache was able to provide requested data to users 93% of the time. Nonetheless, this still worked in favour of the disconnected clients since according to (Li & Levine 2012; Hui & Haiyan 2016), the availability of a storage facility should be at least 90%. This shows the effectiveness of the proposed cache towards temporarily providing data to users.

The response times of the proposed cache for read and write operations are shown in Figure 5-5 and Figure 5-7, respectively. These figures follow the same trend of an increase in response time with an increasing number of requests. For read operations, the minimum response time is 124.5ms, which resulted from 50 requests, and the maximum is 230.0 corresponding to 450 requests. Similarly for write operations, the minimum response time obtained was 140.9 for 50 requests, and the maximum value was 230.0 for 500 requests. The direct proportionality behaviour between response time and the independent factor in a cache storage system is explained by (Amiri et al. 2003; Rashid et al. 2016) as denoting the ability of a system to scale well, thus improving overall performance and availability of the cache.

Similarly, Figure 5-6 and Figure 5-8 present throughputs for the cache model for read and write operations. These figures also show an increase in throughput related to the increasing number of requests to be processed. The minimum and maximum values for read requests

were 82.3 for 50 requests and 173.3 for 400 requests respectively. From 400 onwards, the cache shows an unexpected decrease in throughput. For cache systems, this may be due to conflicts caused by a very large number of data access operations , resulting in the blockage or abortion of the transactions being executed, thus yielding poor throughput performance (Kounev & Buchmann 2002). Nonetheless, this decrease does not necessarily impact the AMC environment negatively since the disconnections are assumed to last for shorter periods, hence not a larger number of requests is expected. For write requests, the minimum value obtained was 70.6 for 50 requests, and the maximum value was 404.1 for 500 requests. This behaviour is also reported to work in favour of the overall improved system performance (Wu & Reddy 2010).

Due to these results which show the proposed model as achieving a high rate of data availability whilst performing well, it can, therefore, be concluded that the model is well suited for ad hoc clouds, mainly in critical scenarios such as mobile healthcare.

### 5.6 Summary

In this chapter, the prototype of the mHealthcare scenario was evaluated to observe its usability during disconnections. Experiments were conducted to evaluate the cache's data availability rate and its performance under a different number of requests issued. The results indicated that the cache is able to maintain data availability to disconnected mobile clients most of the time. For performance evaluations, the cache was found to be able to scale with the increasing number of requests. Therefore, it can be concluded that the proposed model not only maintains data availability during disconnections but also efficiently handles the increasing number of requests received, thus improving the performance of the whole system.

# CHAPTER SIX

# CONCLUSION AND FUTURE DIRECTIONS

## 6.1 Introduction

This research work tried to address the Internet disconnection challenge faced by mobile clouds due to the unreliability of wireless connections. These disconnections result in mobile clients not being able to access services from cloud servers. This is not tolerable to most mobile cloud computing applications where instant access to data is required most of the times. However if local storage could be provided for these occasional occurrences, the applications would then be able to function smoothly in the face of disconnections. Hence this work observed a need to provide such local storage, where during connectivity outages, the mobile node in a particular community consisting of users who share similar interests would then form an ad hoc mobile cloud. This local storage would then be made available within that formed mobile cloud.

Therefore, this work strived towards formulating a model that could be used in such an environment. The prototype of the model was developed and its performance and availability were tested. The requirements for the infrastructureless mobile cloud to be integrated with GUISET in the provisioning of the proposed cache were also outlined. The remainder of this chapter presents the conclusions that were drawn based on the results obtained. Limitations of this study and possible future extensions are also highlighted.

## 6.2 Conclusion

Caching is one of the widely used storage mechanisms in different computing paradigms, mainly due to its ability to provide fast access to data. This is because the data is stored locally on the computing system, which minimises latency, thus improving the system's

performance. Apart from this advantage, caching is also being used to support disconnected clients. This is achieved through the storage of data locally, which will later be used to service clients' requests during Internet disconnections. This work follows the latter scenario.

Section 1.5 of chapter 1 presented the four objectives of this work. In this section, a brief explanation of how they were achieved is presented.

**OBJECTIVE I**: To document why the current caching approaches to tolerate faults in DBaaS are not adequate for ad hoc Mobile Cloud

In this work, a cache model that will ensure data availability to occasionally disconnected mobile cloud computing applications through an Ad hoc mobile cloud-powered GUISET environment was proposed. To achieve this, the GUISET environment and its integration requirements with the ad hoc mobile cloud were presented in Chapter 2. Due to insufficiency of the current caching mechanism to support ad hoc mobile clouds during Internet disconnections, the literature was surveyed in Chapter 3 to find out about this insufficiency. The permanent storage of data in a mobile device's memory, resulting in only a small amount of data being stored due to memory scarcity in mobile devices, was found to be amongst the main reasons behind this lack of support. The conduction of this literature survey also provided an answer to the first research question "Why is the current DBaaS handling of offline data necessary to maintain smooth running of clients during connectivity outage not adequate in ad hoc mobile cloud?"

**OBJECTIVE II**: Use the information gathered from objective I to specify what is required to make DBaaS make provision for tolerating faults such as connectivity outage.

Based on the literature reviewed, a caching model to host DBaaS data during Internet disconnections in an Ad hoc mobile cloud-powered GUISET environment was proposed.

Unlike persisting data in the device's storage memory, the proposed cache is deployed on a mobile device only at runtime, that is, when the Internet disconnection situation strikes. Moreover, due to the fact that some of the devices within the network might not have enough space to host the cache, the proposed cache is shared amongst the mobile devices in the network, so as to support all clients regardless of their resource limitations. Consequently, the cache model and its aforementioned specification answered the second research question of "How can we ensure that DBaaS model(s) tolerate(s) the data requirements of Clients with regards to data availability in an ad hoc Mobile Cloud, in cases of Internet disconnections?"
"

**OBJECTIVE III**: Design a caching model with regard to the fault tolerance specification, which will support Clients which have to operate using an ad hoc mobile cloud.

The design for the proposed cache model for tolerating the Internet outage fault was carried out in Chapter 4 in accordance with the specification presented by objective II. A number of factors were taken into consideration:

I. The properties of a mobile ad hoc network.
II. The prefetching mechanism for periodically hoarding data to prepare for Internet disconnections.
III. The synchronization mechanism to propagate locally made changes to the cloud database.

As aforementioned, the model was proposed and designed to ensure DBaaS data availability for mobile cloud applications, in ad hoc mobile cloud environments. Hence the model was designed for disconnected mobile cloud applications, whilst conforming to mobile ad hoc network standards.

**OBJECTIVE IV**: Evaluate the data availability of the fault-tolerant caching model from objective III in an ad hoc Mobile Cloud application Scenario e.g. m-Healthcare.

To demonstrate the usability of the proposed cache model, a prototype of an mHealthcare system was created based on the cache design presented in Chapter 4, with regard to the prototyping research method. Rate of data availability and the performance of the prototype were evaluated and presented in Chapter 5. Analysis of the results showed that the model maintains data availability more than 90% of the time during disconnections, that is, more than 90% of the issued requests were successfully serviced by the cache, which is what the literature classifies as best performance in cache systems. This analysis indicates the suitability of the proposed cache model in hosting DBaaS data during Internet disconnections, allowing the continual functioning of clients. Furthermore, the usage of a random sampling method to select proof of concept data in a cache produced more effective results in drawing conclusions on the usability of the cache.

Moreover, the model showed that it could perform scalably with an increasing number of requests. More specifically, a direct proportionality relationship was observed between the increasing number of concurrent requests against response time and throughput respectively. This scalability relationship between the performance metrics and the number of requests make the cache most suitable for small environments where an increase in the number of concurrent data requests issued on a cache is expected. In essence, the model was observed to perform scalably towards maintaining data availability to disconnected mobile clients in mobile clouds.

### 6.3 Limitations and future work

This work developed a caching model for providing data to mobile disconnected users. Despite the better performance shown by this model, there are some limitations to be

considered. These limitations include the fact that we only considered the shared-cache approach to the provisioning of data. Mobile ad hoc networks also use one cooperation approach where mobile devices search each other's internal caches to enhance the data availability rate. Moreover, to fully support the disconnected operation approach, the synchronisation state between the cache and the cloud server after disconnections should be implemented. Therefore in future, we intend to support the sharing of data within the mobile device's local caches and to fully support the disconnected operation in ad hoc mobile cloud-powered GUISET environments.

# References


Adamczyk, P. et al., 2011. REST and Web Services : In Theory and in Practice.

Agarwal, R. & Motwani, D.M., 2009. Survey of clustering algorithms for MANET. *International Journal on Computer Science and Engineering*, 1(2), pp.98–104.

Ahmed, M. et al., 2012. An Advanced Survey on Cloud Computing and State-of-the-art Research Issues. *IJCSI International Journal of Computer Science Issues*, 9(1), pp.201–207.

Akingbesote, A.O. et al., 2014. Performance Modeling of Proposed GUISET Middleware for Mobile Healthcare Services in E-Marketplaces. *Journal of Applied Mathematics*, p.9.

Allamaraju, S., 2007. *RESTful Web Services Cookbook*, O'Reilly Media.

Alshahwan, F., Carrez, F. & Moessner, K., 2012. Providing and Evaluating the Mobile Web Service Distribution Mechanisms Using Fuzzy Logic. *Journal Of Software*, 7(7), pp.1473–1487.

Alshahwan, F. & Moessner, K., 2010. Providing SOAP Web Services and RESTful Web Services from Mobile Hosts. In *Proceedings of the 2010 fifth international conference on internet and web applications and services*. pp. 174–179.

Alshahwan, F., Moessner, K. & Carrez, F., 2010. Evaluation of Distributed SOAP and RESTful Mobile Web Services. In *International Journal on Advances in Networks and Services*. pp. 447–461.

Aminzadeh, N. et al., 2015. Simulation Modelling Practice and Theory Mobile storage augmentation in mobile cloud computing : Taxonomy , approaches , and open issues. *Simulation Modelling Practice and Theory*, 50, pp.96–108. Available at: http://dx.doi.org/10.1016/j.simpat.2014.05.009.

Amiri, K. et al., 2003. DBProxy : A dynamic data cache for Web applications. In *19th International Conference on Data Engineering*. pp. 821–831.

Arora, I. & Gupta, A., 2012. Cloud Databases: A Paradigm Shift in Databases. *International Journal of Computer Science*, 9(4), pp.77–83. Available at: http://www.ijcsi.org/papers/IJCSI-9-4-3-77-83.pdf.

Avancha, S., Baxi, A. & Kotz, D., 2012. Privacy in Mobile Technology for Personal Healthcare. *ACM Computer Surveys*, 45(1), pp.1–54.

Basu, P., Little, T.D. & Khan, N., 2001. A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks. In *2001 International Conference on Distributed Computing Systems Workshop*. pp. 413–418.

Belqasmi, F. & Glitho, R., 2011. RESTful Web Services for Service Provisioning in Next-Generation Networks : A Survey. *IEEE Communications Magazine*, (December), pp.66–73.

Bentaleb, A., Boubetra, A. & Harous, S., 2013. Survey of Clustering Schemes in Mobile Ad hoc Networks. *Communications and Network*, 5(2B), pp.8–14.

Carroll, A. & Heiser, G., 2010. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*. pp. 1–14. Available at: http://portal.acm.org/citation.cfm?id=1855861.

Chand, N., Joshi, R.C. & Misra, M., 2007. Cooperative caching in mobile ad hoc networks based on data utility. *Mobile Information Systems*, 3(1), pp.19–37.

Chang, C., Ling, S. & Krishnaswamy, S., 2011. ProMWS : Proactive Mobile Web Service Provision Using Context-Awareness. In *8th IEEE International Workshop on Managing Ubiquitous Communications and Services*. pp. 69–74.

Chang, S. & Curtis, D., 2002. An Approach to Disconnected Operation in an Object-Oriented Database. In *Proceedings of the Third International Conference on Mobile Data Management*. pp. 19–26.

Chappell, D. & Jewell, T., 2002. *Java Web Services*.

Chauhan, N., Awasthi, L.K. & Chand, N., 2009. Cooperative Data Caching with Prefetching in Mobile Ad-Hoc Networks. In *First Asian Himalayas International Conference on Internet*. pp. 1–4.

Chauhan, N., Awasthi, L.K. & Chand, N., 2011. Data Caching with Intelligent Prefetching in Mobile Ad Hoc Networks. In *International Conference on Communication Systems and Network Technologies*. pp. 71–75.

Cheluvaraju, B. et al., 2011. Anticipatory Retrieval and Caching of Data for Mobile Devices in Variable-Bandwidth Environments. In *IEEE International Systems Conference*. pp. 531–537.

Chun, B. et al., 2004. Selfish Caching in Distributed Systems : A Game-Theoretic Analysis. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. pp. 21–30.

Cossio, M.L.T. et al., 2012. *Virtualization:A manager's guide*, Available at: http://www.ncbi.nlm.nih.gov/pubmed/15003161.

Curbera, F. et al., 2002. Unraveling the Web Services Web:An Introduction to SOAP, WSDL, and UDDI. *IEEE INTERNET COMPUTING*, pp.86–93.

Demers, a. et al., 1994. The Bayou Architecture: support for data sharing among mobile users. In *Workshop on Mobile Computing Systems and Applications*.

Denko, M.K., 2007. Cooperative Data Caching and Prefetching in Wireless Ad Hoc Networks. *International Journal of Business Data Communications and Networking*, 3(1), pp.1–15.

Dev, D. & Baishnab, K.L., 2014. A Review and Research towards Mobile Cloud Computing. In *2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. pp. 252–256.

Dinh, H.T. et al., 2011. A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches. In *Wireless Communications and Mobile Computing*.

Dorn, C. & Dustdar, S., 2006. Achieving Web Service Continuity in Ubiquitous Mobile Networks : the SRR-WS Framework. In *4th International Workshop on Ubiquitous Mobile Information and collaboration Systems*. pp. 954–968.

Doukas, C., Pliakas, T. & Maglogiannis, I., 2010. Mobile healthcare information management utilizing Cloud Computing and Android OS. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC'10*. pp. 1037–1040.

Dubey, A. & Wagle, D., 2007. Delivering software as a service. *The McKinsey Quarterly*, pp.1–12.

Ekabua, O.O. & Adigun, M., 2010. GUISET LogOn: Design and Implementation of GUISET-driven Authorization Framework. In *The First International Conference on Cloud Computing, GRIDs, and Virtualization*.

Elbashir, K. & Deters, R., 2005. Transparent Caching for Nomadic WS Clients. In *Proceedings of the IEEE International Conference on Web Services*. pp. 177–184.

Fakude, S.C. et al., 2015. On Supporting Disconnected Operation in Ad hoc Mobile Cloud Computing. In *Proceedings of the Southern Africa Telecommunication Networks and Applications Conference*.

Fathi, A. & Taheri, D. hasan, 2010. Enhance Topology Control Protocol(ECEC) to Conserve Energy based clustering in Wireless Ad Hoc Networks. In *2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*. pp. 356–360.

Fathy, M. et al., 2008. Some Enhanced Cache Replacement Policies for Reducing Power in Mobile Devices. In *2008 Internatioal Symposium on Telecommunications*. pp. 230–234.

Feng, Y. et al., 2014. On Data Caching for Mobile Clouds. In *International Conference on Computer Communications and Networks*. pp. 275–284.

Fernando, N., Loke, S.W. & Rahayu, W., 2013. Mobile cloud computing : A survey. *Future Generation Computer Systems*, 29(1), pp.84–106. Available at: http://dx.doi.org/10.1016/j.future.2012.05.023.

Fiedler, M., Cabanilla, C. & Lê-Quôc, A., 2013. *The Top 5 AWS EC2 Performance Problems*,

Forman, G.H. & Zahorjan, J., 1994. The Challenges of Mobile Computing. *IEEE Journals and magazines*, 27(4), pp.38–47.

Forrester Consulting, 2012. *Database-As-A-Service Saves Money , Improves IT Productivity , And Speeds Application Development*, Available at: https://www.vmware.com/files/pdf/vfabric/VMWARE_Forrester_TLP_Final_10_5.pdf [Accessed September 26, 2015].

Foster, I. et al., 2008. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop 2008 (GCE '08)*. pp. 1–10. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4738445.

Fredrich, T., 2012. *RESTful Service Best Practices Recommendations for Creating Web Services*,

Gagnaire, M. et al., 2012. Downtime statistics of current cloud solutions. In *International Working Group on Cloud Computing Resiliency*. pp. 2–3.

Gao, J. et al., 2012. DIPLOMA : Consistent and Coherent Shared Memory over Mobile Phones. In *IEEE 30th international conference on computer design*. pp. 371–378.

Gruber, R. et al., 1994. Disconnected Operation in the Thor Object-Oriented Database System. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*. pp. 51–56.

Gupta, A.K., 2008. Challenges of Mobile Computing. In *Proceedings of 2nd National Conference on Challenges & Opportunities in Information Technology,*.

Hakan Hacıg¨um ¨us, B.I. & Mehrotra, S., 2002. Providing Database as a Service. In *Proceedings of the 18th International Conference on Data Engineering*. pp. 29–38.

Hamad, H., Saad, M. & Abed, R., 2010. Performance Evaluation of RESTful Web Services. *International Arab Journal of e-Technology*, 1(3), pp.72–78.

Han, J. et al., 2011. Survey on NoSQL Database. In *6th International Conference on Pervasive Computing and Applications (ICPCA)*. pp. 363–366.

Hong-qing, G. & Yan-jie, Z., 2010. System design of cloud computing based on Mobile Learning. In *Knowledge Acquisition and Modeling (KAM), 2010 3rd International Symposium on*. pp. 239–242.

Houngbadji, T. & Pierre, S., 2010. Distributed Data Sharing in Mobile Ad Hoc Networks. In *2010 Australasian Telecommunication Networks and Applications Conference*. pp. 1–6.

Huebscher, M.C. & McCann, J. a, 2008. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys*, 40(3), pp.1–28. Available at: http://portal.acm.org/citation.cfm?doid=1380584.1380585.

Huerta-Canepa, G. & Lee, D., 2010. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing Services Social Networks and Beyond MCS 10*. pp. 1–5. Available at: http://portal.acm.org/citation.cfm?doid=1810931.1810937.

Hui, Y & Haiyan, H., 2016. A Study on Data Storage System Based on High-Availability Open Virtual Experiment Platform, *2016 International Conference on Robots & Intelligent System*

Hussain, K. et al., 2013. Efficient Cluster Head Selection Algorithm for MANET. *Journal of Computer Networks and Communications*, 2013(Article ID 723913), pp.1–8.

Hussien, N.S., Sulaiman, S. & Shamsuddin, S.M., 2013. Review on Pre-fetching for Mobile Cloud Computing. In *IEEE Conference on e-Learning, e-Management and e-Services*. pp. 130–135.

Jansen, W. & Grance, T., 2011. *Guidelines on Security and Privacy in Public Cloud Computing*, Available at: http://csrc.nist.gov/publications/nistpubs/800-144/SP800-144.pdf.

Jia, W. et al., 2011. SDSM : A Secure Data Service Mechanism in Mobile Cloud Computing. In *The First International Workshop on Security in Computers, Networking and Communications*. pp. 1060–1065.

Jiang, Z. & Kleinrock, L., 1998. Web Prefetching in a Mobile Environment. In *IEEE Personal Communications*. pp. 25–34.

Jouppi, N.P., 1993. Cache Write Policies and Performance. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*. pp. 191–201.

Joy, P.T. & Jacob, K.P., 2012. Cooperative Caching Techniques for Mobile Ad hoc Networks. In *International Conference on Data Science & Engineering*. pp. 175–180.

Kasemi, S., Zamanifar, K. & Nematbakhsh, N., 2014. Shared Cache in Mobile Cloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(10), pp.35–38.

Katsaros, G., Kübert, R. & Gallizo, G., 2011. Building a service-oriented monitoring framework with REST and Nagios. In *2011 IEEE International Conference on Services Computing*. pp. 426–431.

Khandekar, A.A. & Mane, S.B., 2015. Analyzing Different Cache Replacement Policies on. In *2015 International Conference on Industrial Instrumentation and Control*. pp. 709–712.

Kiani, S.L. et al., 2012. Context caches in the Clouds. *Journal of Cloud Computing: Advances, Systems and Applications*, 1(7), pp.1–13.

Kim, S., 2006. Synchronization in an Embedded DBMS Environment. *IJCSNS International Journal of Computer Science and Network Security*, 6(7), pp.30–35.

Kistler, J.J., 1993. *Disconnected Operation in a Distributed File System*.

Kistler, J.J. & Satyanarayanan, M., 1992. Disconnected operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1), pp.3–25.

Kohn, L.T., Corrigan, J. & Donaldson, M.S., 2000. *To err is human: building a safer health system*, Washington, DC. Available at: http://books.google.com/books?hl=en&lr=&id=JInZiZnUyicC&oi=fnd&pg=PA1&dq=To+Err+Is+Human+Building+a+Safer+Health+System&ots=hqCQPF8GN6&sig=0EvrocvHDOzSwyo4ZtoqTgq9t2w.

Kolb, S. & Wirtz, G., 2014. Towards application portability in platform as a service. *Proceedings - IEEE 8th International Symposium on Service Oriented System Engineering, SOSE 2014*, pp.218–229.

Konanki, P. & Butt, A.R., 2007. On Supporting Disconnected Operation in Grid Computing. In *Poster in IEEE International Conference on High Performance Computing*.

Koukoumidis, E. et al., 2011. Pocket Cloudlets. In *Proceedings of the 16th international conference on Architectural support for programming languages and operating systems*. pp. 171–184.

Kounev, S. & Buchmann, A., 2002. Improving Data Access of J2EE Applications by Exploiting Asynchronous Messaging and Caching Services. In *Proceedings of the 28th international conference on Very Large Data Bases*. pp. 574–585.

Kovachev, D. et al., 2010. Mobile Community Cloud Computing: Emerges and Evolves. In *Proceedings - IEEE International Conference on Mobile Data Management*. pp. 393–395.

Kovachev, D., Cao, Y. & Klamma, R., 2011. Mobile Cloud Computing : A Comparison of Application Models. *Information Systems Journal*, (4), pp.14–23. Available at: http://arxiv.org/abs/1107.4940.

Kulkarni, G., Sutar, R. & Gambhir, J., 2012. Cloud Computing-Infrastructure As Service- Amazon Ec2. *International Journal of Engineering Research and Applications*, 2(1), pp.117–125.

Kumar, K. & Lu, Y.H., 2010. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4), pp.51–56.

Li, H. & Hua, X.-S., 2010. Melog: mobile experience sharing through automatic multimedia blogging. In *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing*. pp. 19–24.

Li, Y. & Levine, C., 2012. Extending TPC-E to measure availability in database systems. In *Proceedings of the Third TPC Technology conference on Topics in Performance Evaluation, Measurement and Characterization*. pp. 111–122.

Li, Z., Wang, C. & Xu, R., 2001. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*. pp. 238–246. Available at: http://dl.acm.org/citation.cfm?id=502257.

Lim, S., Lee, B. & Kim, J., 2013. Voluntary Disconnected Operations for Energy Efficient Mobile Devices in Pervasive Computing Environments. *Intelligent Automation & Soft Computing*, 19(1), pp.39–49.

Lin, T.-H. et al., 2014. Using a database as a service for providing electronic health records. In *IEEE-EMBS International Conference on Biomedical and Health Informatics*. pp. 9–12.

Liu, F. et al., 2013. Gearing resource-poor mobile devices with powerful clouds: Architectures, challenges, and applications. *IEEE Wireless Communications*, 20(3), pp.14–21.

Lopez, J., Oppliger, R. & Pernul, G., 2004. Authentication and authorization infrastructures ( AAIs ): a comparative survey. *Computers & Security*, 23(7), pp.578–590.

M´arquez, J. et al., 2008. Exploring the Benefits of Caching and Prefetching in the Mobile Web. In *Second IFIP Symposium on Wireless Communications and Information Technology for Developing Countries*.

Ma, S.-P. et al., 2015. Framework for Enhancing Mobile Availability of RESTful Services. In *Mobile Networks and Applications*. Mobile Networks and Applications. Available at: http://dx.doi.org/10.1007/s11036-015-0655-7.

Marinos, A. & Briscoe, G., 2009. Community Cloud Computing. In *Proceedings of Cloud Computing: First International Conference (CloudCom)*. pp. 472–484.

Mayo, R.N. et al., 2003. Energy consumption in mobile devices: Why future systems need requirements-aware energy scale-down. *Proceedings of the Workshop on Power-Aware Computing Systems*, pp.26–40. Available at: http://www.springerlink.com/index/YL3N30H3581B94LJ.pdf.

Mell, P. & Grance, T., 2011. The NIST Definition of Cloud Computing. *U.S. National Institute of Standards and Technology*.

Memcached, 2016. Memcached. Available at: http://memcached.org/ [Accessed February 19, 2016].

Microsoft, 2012. Fault-tolerance in Windows Azure SQL Database. Available at: http://azure.microsoft.com/blog/2012/07/30/fault-tolerance-in-windows-azure-sql-database/ [Accessed November 12, 2015].

Milutinovic, V., 2000. Caching in Distributed Systems. *IEEE Concurrency*, 8(3), pp.14–15.

Mtibaa, A., Fahim, A. & Harras, K.A., 2013. Towards Resource Sharing in Mobile Device Clouds : Power Balancing Across Mobile Devices. In *Proceedings of the second edition of the MCC workshop on Mobile cloud computing*. pp. 51–56.

Neamtiu, I. et al., 2013. Improving Cloud Availability with On-the-fly Schema Updates. In *The 19th International Conference on Management of Data (COMAD),*.

Neto, M.D., 2014. A brief history of cloud computing. Available at: http://www.thoughtsoncloud.com/2014/03/a-brief-history-of-cloud-computing/ [Accessed October 19, 2015].

Ngai, E.C.H. & Lyu, M.R., 2004. Trust- and Clustering-Based Authentication Services in Mobile Ad Hoc Networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*. pp. 582–587.

Olken, F., 1993. *Random Sampling from Databases*.

Ollite, I. & Mohamudally, N., 2015a. Cost Efficient RESTful Services Caching for Mobile Devices. *International Journal of Software and Web Sciences*, 11(1), pp.96–101.

Ollite, I. & Mohamudally, N., 2015b. Performance Analysis of a 2-tier caching proxy system for mobile RESTful services. In *IEEE International Conference on Computer as a Tool (EUROCON)*. pp. 1–7.

Oracle, 2011. *Database as a Service: Reference Architecture – An Overview*,

Pallis, G., 2010. Cloud Computing The New Frontier of Internet Computing. *IEEE Internet Computing*, 14(5), pp.70–73.

Palpanas, T., Larson, P.-Å. & Goldstein, J., 2001. *Cache Management Policies for Semantic Caching*,

Papageorgiou, A. et al., 2011. Enhancing the Caching of Web Service Responses on Wireless Clients. In *IEEE International Conference on Web Services*. pp. 10–16.

Park, S., Ko, Y. & Kim, J., 2003. Disconnected Operation Service in Mobile Grid. In *1st International Conference on Service Oriented Computing*. pp. 499–513.

Pautasso, C., Leymann, F. & Zimmermann, O., 2008. RESTful Web Services vs . " Big " Web Services : Making the Right Architectural Decision. , pp.805–814.

Pendse, R. & Ksitta, H., 1999. Selective Prefetching: Prefetching when only required. In *42nd Midwest Symposium on Circuits and Systems*. pp. 866–869.

Pettersen, R. et al., 2014. Jovaku: Globally Distributed Caching for Cloud Database Services Using DNS. In *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. pp. 127–135. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6834954.

Piatek, M., Jackson, J. & Juola, P., 2008. Distributed Web Proxy Caching in a Local Network Environment. *The Student Research Competition*, pp.1–8. Available at: http://src.acm.org/subpages/papers/piatek.src.2004.pdf.

Pizette, L. & Cabot, T., 2012. Database as a Service: A Marketplace Assessment. Available at: http://hotels.emacromall.com/techpapers/Database as a Service - A Marketplace Assessment.pdf [Accessed October 23, 2015].

Podlipnig, S. & Böszörmenyi, L., 2003. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4), pp.374–398.

Polipo, 2016. Polipo. Available at: http://www.pps.univ-paris-diderot.fr/~jch/software/polipo/ [Accessed February 19, 2016].

Preetha, V. & Chitra, K., 2014. Clustering & Cluster Head Selection Techniques in Mobile Adhoc Networks. *International Journal of Innovative Research in Computer and Communication Engineering*, 2(7), pp.5151–5157.

Qi, H. & Gani, A., 2014. Research on Mobile Cloud Computing : Review , Trend and Perspectives. In *Proceedings of the Second International Conference on Digital Information and Communication Technology and its Applications*. pp. 195–202.

Qian, F. et al., 2012. Web Caching on Smartphones : Ideal vs . Reality Categories and Subject Descriptors. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services*.

Ramasubramanian, V. & Terry, D., 2004. *Caching XML Web Services to Support Disconnected Operation*,

Rashid, S.A. et al., 2016. Cache Persistence Aware Response Time Analysis for Fixed Priority Preemptive Systems. In *28th Euromicro Conference on Real-Time systems*.

Saemundsson, T. et al., 2014. Dynamic Performance Profiling of Cloud Caches. In *Proceedings of the ACM Symposium on Cloud Computing*. pp. 1–14.

Said, E.G., Omar, E.B. & Robert, L., 2009. Data Prefetching Algorithm in Mobile Environments. *European Journal of Scientific Research*, 28(3), pp.478–491.

Saleh, A. et al., 2012. A Systematic Review of Healthcare Applications for Smartphones. *BMC Medical Informatics and Decision Making*, 12(1), pp.1–31. Available at: BMC Medical Informatics and Decision Making.

Satyanarayanan, M., 1996. Fundamental Challenges in Mobile Computing. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*.

Sen, J., 2010. A Robust and Efficient Node Authentication Protocol for Mobile Ad Hoc Networks. In *Proceedings of the 2nd International Conference on Computational Intelligence, Modelling and Simulation*. pp. 476–481.

Shaikh, A.A. & Gupta, G.K., 2014. M-Commerce Recommendation With Mobile Cloud Architecture. *International Journal of Application or Innovation in Engineering & Management*, 3(11), pp.347–351.

Shanmugam, R. & Maluk, M.M.A., 2012. Data Management in the Mobile Cloud Using Surrogate Object. *International Journal of Future Computer and Communication*, 1(2), pp.187–192.

Shehri, W. Al, 2013. CLOUD DATABASE DATABASE AS A SERVICE. *International Journal of Database Management Systems ( IJDMS )*, 5(2), pp.1–12.

Shiraz, M. et al., 2013. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *IEEE Communications Surveys and Tutorials*, 15(3), pp.1294–1313.

Shravanthi, C. & Guruprasad, H.S., 2014. MOBILE CLOUD COMPUTING AS FUTURE FOR MOBILE APPLICATIONS. *International Journal of Research in Engineering and Technology*, 3(5), pp.2319–2322.

Sihawu, S.A. et al., 2013. Dynamic Composition of Portal Interface for GUISET Services. In *International Conference on Adaptive Science and Technology*.

Singh, R. & Dutta, C., 2013. A Synchronization Algorithm of Mobile Database for Cloud Computing. *International Journal of Application or Innovation in Engineering & Management*, 2(3), pp.491–497.

Spillner, J. et al., 2013. RAFT-REST - A Client-side Framework for Reliable , Adaptive and Fault-Tolerant RESTful Service Consumption. In *Service-Oriented and Cloud Computing, Volume 8135 of the series Lecture Notes in Computer Science*. pp. 104–118.

SQLite, 2016a. About SQLite. Available at: https://www.sqlite.org/about.html [Accessed February 19, 2016].

SQLite, 2016b. SQLite Android Bindings. Available at: https://www.sqlite.org/android/doc/trunk/www/index.wiki [Accessed February 19, 2016].

Squid, 2016. Squid. Available at: http://www.squid-cache.org/ [Accessed February 19, 2016].

statistica, 2015. Global mobile retail commerce revenue 2018 _ Statistic. Available at: http://www.statista.com/statistics/324636/mobile-retail-commerce-revenue-worldwide/ [Accessed November 16, 2015].

Subramanian, K., 2011a. *Hybrid Clouds*, Available at: http://la.trendmicro.com/media/wp/hybrid-clouds-whitepaper-en.pdf.

Subramanian, K., 2011b. *Private Clouds*, Available at: http://la.trendmicro.com/media/wp/private-clouds-whitepaper-en.pdf.

Sudha, K. et al., 2015. A SURVEY ON ADVENT OF THE DBaaS AND ITS ENHANCEMENTS. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 4(1), pp.136–140.

Suguna, M. & Subathra, P., 2011. Establishment of Stable Certificate Chains for Authentication in Mobile Ad Hoc Networks. In *2011 International Conference on Recent Trends in Information Technology*. pp. 234–239.

Takase, T. & Tatsubori, M., 2004. Efficient Web Services Response Caching by Selecting Optimal Data Representation. In *Proceedings of the 24th International Conference on Distributed Computing Systems*. pp. 188–197.

Tang, W.-T.T.W.-T., Hu, C.-M.H.C.-M. & Hsu, C.-Y.H.C.-Y., 2010. A mobile phone based homecare management system on the cloud. *Biomedical Engineering and Informatics (BMEI), 2010 3rd International Conference on*, 6, pp.2442–2445.

Tere, G.M., Mudholkar, R.R. & Jadhav, B.T., 2014. Improving Performance of RESTful Web Services. In *International Conference on Advances in Engineering & Technology*. pp. 12–16.

Terry, D.B. & Ramasubramanian, V., 2003. *Caching XML Web Services for Mobility*,

Thomas, D., 2009. Cloud Computing – Benefits and Challenges ! *Journal Of Object Technology*, 8(3), pp.37–41.

Tiwari, R., Buse, S. & Herstatt, C., 2007. From electronic to mobile commerce : Technology convergence enables innovative business services FROM ELECTRONIC TO MOBILE COMMERCE : Available at: https://www.researchgate.net/publication/228635590_From_electronic_to_mobile_commerce_Technology_convergence_enables_innovative_business_services.

Tolba, F.D., Magoni, D. & Lorenz, P., 2007. Connectivity , Energy and Mobility Driven Clustering Algorithm for Mobile Ad Hoc Networks. In *IEEE Global Telecommunications Conference*. pp. 2786–2790.

Vanderwiel, S.P. & Lilja, D.J., 1999. Data Prefetch Mechanisms. *ACM Computing Surveys*, 32(2), pp.174–199.

Varshney, U., 2007. Pervasive healthcare and wireless health monitoring. *Mobile Networks and Applications*, 12(2-3), pp.113–127.

Vemulapalli, C., Madria, S.K. & Linderman, M., 2013. Pre-distribution scheme for data sharing in mobile cloud computing. In *Proceedings of the first international workshop on Mobile cloud computing & networking*. pp. 11–18.

Wagh, K. & Thool, R., 2014. MOBILE WEB SERVICE PROVISIONING AND PERFORMANCE EVALUATION OF MOBILE HOST. *International Journal on Web Service Computing (IJWSC)*, 5(2), pp.1–10.

Wang, J., 1999. A survey of web caching schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29(5), pp.36–46.

Wang, X., Wang, B. & Huang, J., 2011. Cloud computing and its key techniques. *2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE),*, 2, pp.404–410.

Wang, Y., Chen, I. & Wang, D., 2015. A Survey of Mobile Cloud Computing Applications : Perspectives and Challenges. *Wireless Personal Communications*, 80(4), pp.1–29.

Weiqing, Z., Yafei, S. & Lijuan, D., 2010. Improving computer basis teaching through mobile communication and cloud computing technology. *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, 1, pp.V1–452–V1–454.

Wu, B., Wu, J. & Cardei, M., 2007. A Survey of Key Management in Mobile Ad Hoc Networks Bing Wu. In *Handbook of research on wireless security*. pp. 1–23.

Wu, X. & Reddy, A.L.N., 2010. Exploiting concurrency to improve latency and throughput in a hybrid storage system. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. pp. 14–23.

Xiaodong, M. et al., 2010. Research on a Certificate-based Authentication For Mobile Ad hoc. In *201O Second International Conference on Communication Systems, Networks and Applications*. pp. 44–47.

Xiaoyan Yang, Tiejun Pan & Jingjing Shen, 2010. On 3G mobile E-commerce platform based on Cloud Computing. *2010 3rd IEEE International Conference on Ubi-Media Computing*, pp.198–201. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5544470.

Yin, L. & Cao, G., 2006. Supporting Cooperative Caching in ad hoc networks. *IEEE Transactions in Mobile Computing*, 5(1), pp.77–89.

Zahran, M., 2007. Cache Replacement Policy Revisited. In *WDDD held in conjunction with ISCA*. pp. 1–8.

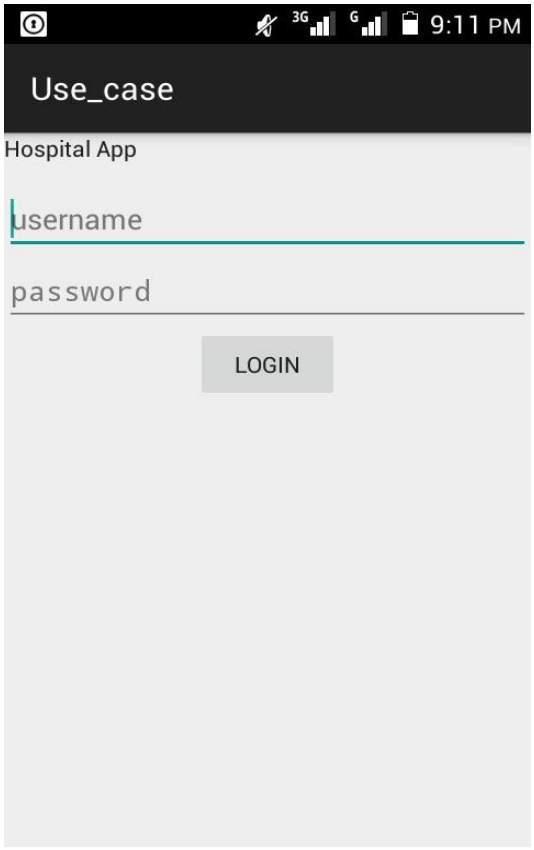
Zebchuk, J., Makineni, S. & Newell, D., 2008. Re-Examining Cache Replacement Policies. In *IEEE International Conference on Computer Design*. pp. 671–678.

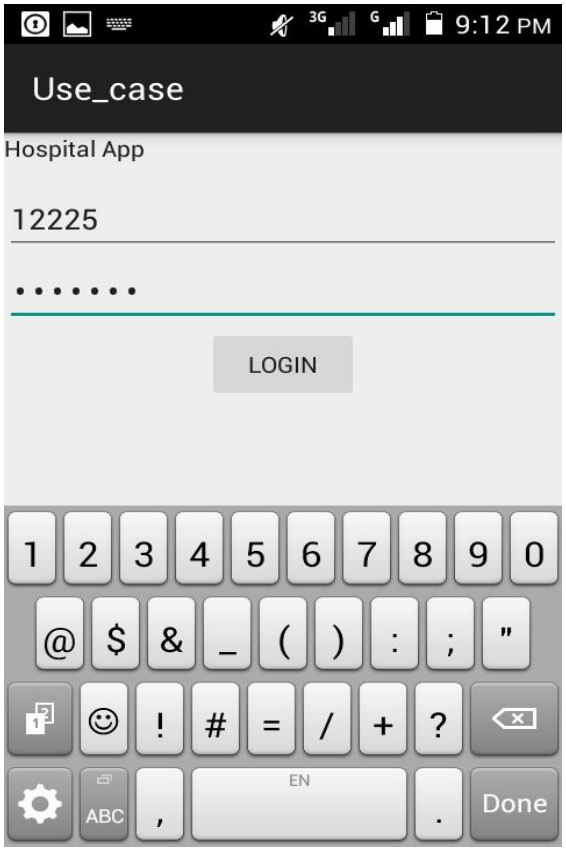
Zhang, Q., Cheng, L. & Boutaba, R., 2010. Cloud computing : state-of-the-art and research challenges. *Journal of Internet Services and Applications*, (1), pp.7–18.

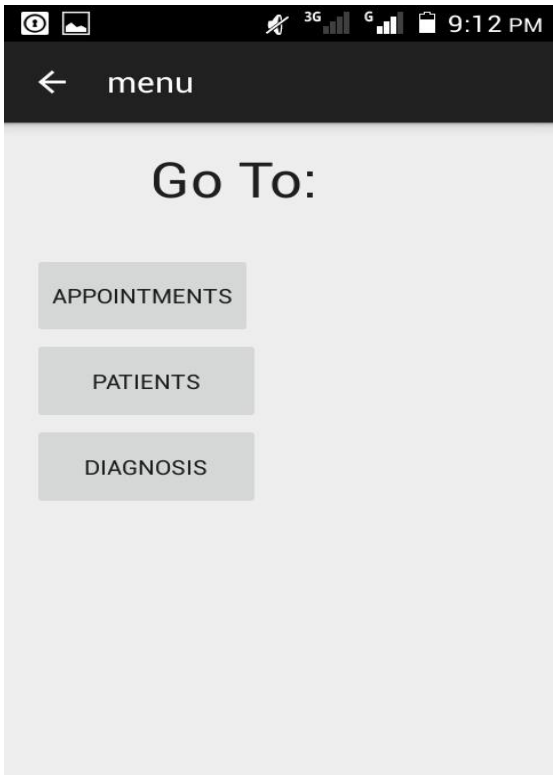
Zhang, X. & Chen, L., 2011. Fault Tolerance Study for Durable Storage on the Cloud. In *International Conference on Cloud and Service Computing*.

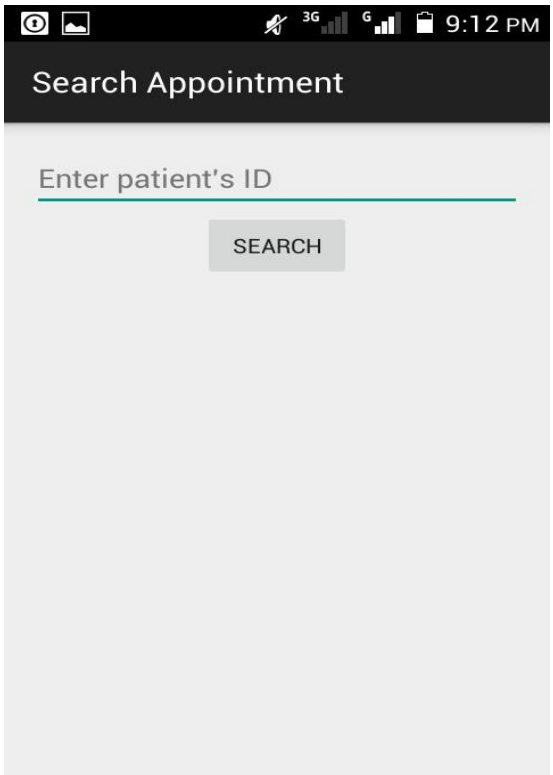
Zhu, T. et al., 2012. Saving Cash by Using Less Cache. In *USENIX Workshop on Hot Topics in Cloud Computing*. pp. 2–6.

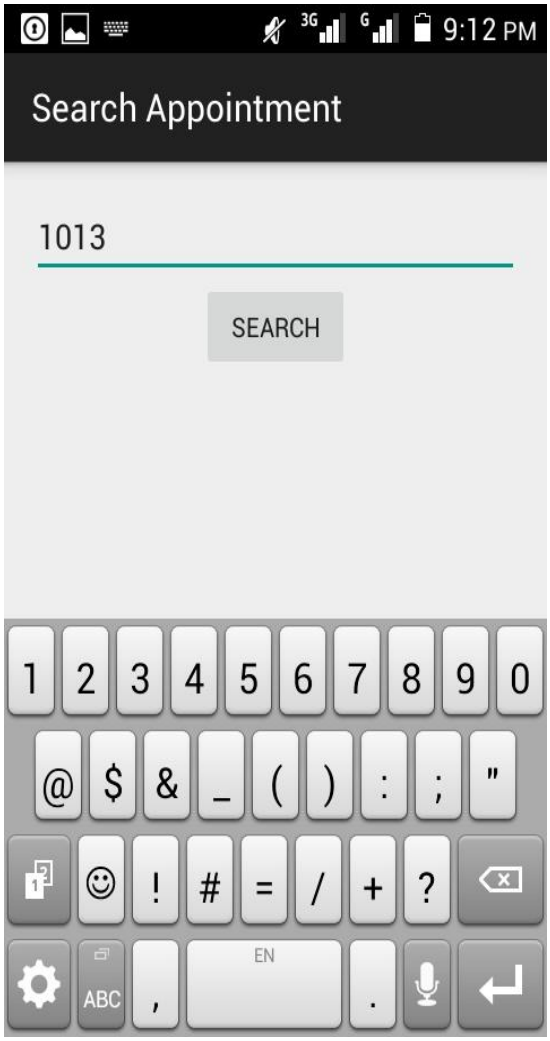
## Appendix A: Mobile Application Implementation




9:12 PM
Search Appointment
1013
SEARCH
1 2 3 4 5 6 7 8 9 0
@ $ & _ ( ) : ; "
! # = / + ?
ABC
EN


9:12 PM
Appointments
Patient ID: 1013
Patient Name: Luyanda
Patient Surname: Mkhize
Appointment reason: wound cleaning
Time: 12:30
Date: 2015-12-04
Attendant: Betty Johnson
Occupation: Nurse
OPTIONS