Data Synchronization in a Network-Volatile Mobile Ecosystem

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Wen Fu

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Today, it is a major issue for mobile applications to maintain a replica state of the server on mobile devices. This creates the need to keep data on both the server and the mobile. In such cases, when the data changes on the server, the new state of the data has to be updated on the mobile in order to maintain a consistent view of the data flow. However, mobile devices communicate over wireless mediums (.e.g., Bluetooth, Wi-Fi, 3.5G/4G, etc.) which can experience intermittent connectivity. The volatility of the network is also influenced by low-bandwidth. The direct effects of these issues are high latency and inconsistency issues between the data on the mobile clients and the remote servers. In this work, I present a detail review on the topic of data synchronization in mobile networks. Then, a generic architecture called MobiQ is proposed which can keep working in an offline mode to record local modifications and can synchronize with the remote servers when connectivity is restored. This is achieved through the proposal of an efficient synchronization protocol which combines different synchronization and replication strategies. Moreover, the MobiQ framework provides a secured environment to work with data. The implemented architecture is designed and tested in mobile questionnaire system and the result is encouraging.

Acknowledgements

Foremost, I would like to thank my supervisor, Dr. Ralph Deters for his support, guidance, patience and financial assistance for my two years of study. Besides my supervisor, I would like to thank the rest of my thesis committees: Dr. Julita Vassileva, Dr. Eric Neufeld, and Dr. Chris Zhang for their valuable advices. Also, I would like to thank Ms. Gwen Lancaster, Graduate Correspondent at the Department of Computer Science, who has been very helpful throughout my study at the University of Saskatchewan. Finally, I would like to thank my parents, Jianguang and Shengyu, for their love and faith in me.

Contents

List of Tables

List of Figures

List of Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
BASE	Basically Available, Soft state, Eventual consistency
DS	Data Synchronization
DP	Data Protection
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
NoSQL	Not Only Structured Query Language
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SP	Synchronization Protocol
SQL	Structured Query Language

Chapter 1

Introduction

Today, data is typically collected from user inputs and activities through the web. An example is the web-based questionnaire, which is the process of collecting data through answering electronic sets of questions on the web [22]. According to Evans and Mathur [9], it was estimated that one third of all types of questionnaires will be web-based surveys in the USA as of 2006. Furthermore, enterprises are shifting focus to the web since it has the power to reach vast majority of users. This potential is now being extended to mobile computing, which is becoming pervasive and ubiquitous.

The advancement in modern mobile devices in terms of storage and support for diverse wireless networks (e.g., Wi-Fi and 3.5/4G) provides new opportunities for client-server interactions. The new mobile-based architecture not only inherits design flexibility and real-time access but also improves mobility from the web-based perspective. This enhances remote accessibility of information on the go without any perceived delay.

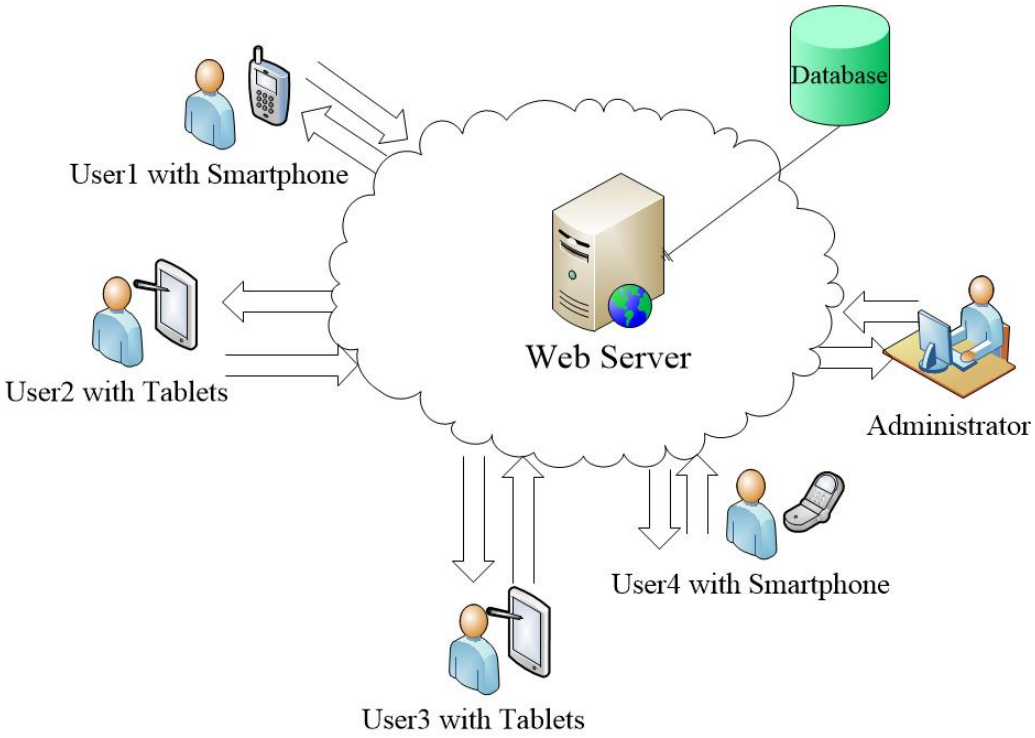

Figure 1.1: The main scenario of a mobile questionnaire system

In the Figure1.1, the architectural design of a mobile questionnaire system is illustrated. In order to clearly explain this concept, I describe the mobile questionnaire scenario. In the Figure1.1, there are several people at different locations using the mobile questionnaire system through their mobile devices. User1 and User4 use smartphones while User2 and User3 use tablets. All of them directly download specific questionnaires from the server, locally work with these data, and finally upload results back to the server. Then, the server processes the uploaded results and stores them into its database. Additionally, there is an administrator who is responsible for managing the database in this system, so the administrator will create new questionnaires, update existing questionnaire, delete redundant information and also collect submitted results from users. The goal is to provide the latest available data to serve requests from clients. Under the circumstances, this system aims to provide an environment that mobile clients can pull and push data anywhere at anytime. However, the communication between mobile clients and the server is over the wireless communication network which is unreliable [20]. Also, the data in the mobile questionnaire system always contains personal information that need to be highly secured. Therefore, this research proposed a 2-tire distributed mobile architecture (i.e. client-server) that can enable users to complete their questionnaires effectively. The proposed system further enables the mobile input to be synchronized efficiently in the network volatile environment. To achieve this, methodologies such as mobile caching and server side pushing are adopted to support offline use and the update management and propagation.

The remaining sections of the thesis are structured as follows:

- Chapter 2 discuss the definition of problems
- Chapter 3 reviews the related technologies and researches
- Chapter 4 explain detail architecture and design
- Chapter 5 explain detail implementation
- Chapter 6 discuses the experiment and results
- Chapter 7 discuses the contributions
- Chapter 8 discusses the future work

Chapter 2

Problem Definition

With the improvement in wireless technology and portable devices, the client and server application has been migrating from the desktop application architecture to the mobile environment. Mobility has been greatly improved, but it also implies that users may access data from different locations and even have to stay connected while on the move. Under this condition, unreliable wireless network may cause connection failure or even terminate applications. Additionally, how to securely deal with data on the mobile environment is another issue. Thus, the next two sub-sections will detail these two difficulties in a mobile environment.

2.1 Unreliable Wireless Network

The wireless network is characterized by intermittent connectivity. It means the connection can be temporally terminated between the clients (i.e. the mobile) and the server, which is mostly the back-end. Due to the variability, the wireless network is considered as unreliable since the reliability of constantly staying connected is threatened. This can subsequently hinder data accessibility on the server from the mobile node (pulling) or sending data to the server.

Challenge 1. Connection Loss

The intermittent loss of connectivity occasionally causes the mobile devices to be partially or totally disconnected from the distributed network [19]. During this period, any request from the mobile clients will fail to be delivered to the server or other nodes that are connected to the architecture. As a result, the mobile application may freeze or even force to shut down. To avoid this situation, the mobile application should still be responsive and have computation ability during the disconnected period.

Challenge 2. Data Synchronization Issue

Agrawal [1] pointed that any mobile application which provides data access has to deal with the inherent problem of data synchronization. In any client-server mobile application, the mobile clients have to retrieve data from the server and may store them into a local storage on each mobile device. This is called caching. On the other hand, the server needs to save real-time updates from the mobile clients. Therefore, mobile clients and a server should have the consistent state. For this reason, if the related data changes on one side, the other sides will have to be updated. Also, this synchronization process has to work seamlessly. However, due to the unstable connection, the real-time synchronization can

be interrupted and cause inconsistency between the data on the mobile clients and the server. Thus, a good mobile application framework should have its own customized synchronization and replication process to deal with the unreliable network.

Challenge 3. Bandwidth

In mobile wireless networks, the bandwidth is limited. Moreover, the bandwidth is affected/influenced when several people share the same network simultaneously. To effectively communicate, mobile architectures are designed to support seamless exchanges of data between the clients and the backend server. Therefore, the minimization of the data/messages across the system can improve the bandwidth usage.

2.2 Security Risks

In distributed mobile systems, mobile clients are sometimes expected to frequently retrieve data from the server and locally store the data to deal with the unreliable connection. Therefore, the security risks and privacy invasion issue are increased. In my work, I focus on the next two challenges.

Challenge 1. Security Issue on the Mobile Device

The mobile clients need to download data into the local storage and also locally save the newly created data before uploading, so that the data will live in the client storage for a certain time. If the device is lost through some unforeseen circumstance, other people will easily access the data and then the private information will be leaked. In order to protect user's private information, the system needs to secure the data on the mobile devices.

Challenge 2. Security Issue during the Transmission

Most mobile applications consume web services over the HTTP transport protocol. However, the default setting for every data in the HTTP message is plain text. If attackers intercept such HTTP requests, they can easily read these messages. So data in the HTTP transport protocol are unsure. As most data in the mobile questionnaire system is private information, the system should protect its information during the transmission.

2.3 Research Goal

This work proposes a mobile questionnaire framework that is a client and server model application to overcome the above highlighted challenges. The client component is a mobile application which is running on the mobile devices; and the server component is hosted remotely to provide no downtime services. The connection between the two components relies on the wireless network. However, it does not mean this application has to be limited because of unreliable connection. Also, the system has to be designed to be highly secured and all data has to be fully protected. Therefore, the following sub-goals and features will be provided in the proposed work.

Goal 1. To support work when the connectivity is unstable

- Online and offline mode
- Efficient synchronization protocol

Goal 2. To ensure data security in the system

- Protect data on the mobile devices
- Protect data during the transmission channel

Chapter 3

Literature Review

In order to overcome the challenges of a mobile questionnaire system, this chapter reviews related researches and technologies in the following fields.

- Cloud Computing in Mobile Environment
- Web Service
- Data Store
- Data Synchronization
- Data Replication
- Data Security
- Online Questionnaire System

3.1 Cloud Computing in Mobile Environments

This section will discuss two macro trends, which are: cloud computing and mobile computing. The concepts of merging the two technologies in order to meet some transactional requirements is known as the mobile cloud computing. This area is fairly new and the details are discussed next.

3.1.1 Cloud Computing

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configuration computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [21] Cloud computing users can easily request network-oriented services anywhere and anytime without worrying about constraints and management of physical resources (e.g., servers, network, etc.). Depending on the business model of the cloud service provider, consumers only pay for what they use. The cloud architecture is categorized into three main groups namely:

- Software as a Service (SaaS)

  This service enables the provider's application to run on the cloud infrastructure. On the other side, consumers can access the application via program application interfaces on various client devices (e.g., browsers, mobile Apps, etc.). Moreover, accessing software and applications from a cloud provider through APIs means the consumers are shielded from the complexities of the physical architecture, which sometimes follow complex logics [21][16]. Examples are: Salesforce.com, Social media, etc.

- Platform as a Service (PaaS)

  This service allows consumers to easily deploy their applications into the cloud infrastructure where the providers have already provided supported programming languages, libraries, services, and helper tools. Identical to software as a service, consumers don't manage or control the network, servers, operating systems or storage of the cloud infrastructure. A good example of PaaS is Google App Engines (GAE). It is a cloud platform which provides services to develop and host web applications or backend services with the Google infrastructure in the cloud [21][26].

- Infrastructure as a Service (IaaS)

  This service is also called Hardware as a Service (HaaS). It is the lowest level of all cloud services. This service facilitates users to work directly with the virtual machine. So consumers are able to customized the virtual machine to run arbitrary softwares or operating systems. Infrastructure as a service can also include virtualized databases, networks, and servers.

3.1.2 Mobile Computing

Mobile computing allows users to share data, voice and video from a computer or any type of mobile device that supports wireless connection to other computing machines [37]. Mobile computing comprises three major components.

- Mobile Communication

  The communication in mobile computing refers to data/message transmission between two components in a wireless network environment. The communication should be reliable and seamless. As mobile communication relies on wireless networks that experience instability (in comparison to wired networks), transmission protocols, services, bandwidth and portals are necessary to facilitate and support uninterrupted data transmission in the mobile environment. Moreover, the format of the data across the various components during mobile communication needs to be predefined. This ensues that there is no parse error while mobile devices retrieve data [37].

- Mobile Hardware

  Mobile hardware includes any type of device that can receive or request data via wireless links. Some examples are portable laptops, smartphones, tablets and so on. These devices contain receptor medium

which can search available signals and connect to them. Also, the receiving and sending functionalities can work simultaneously [37].

- Mobile Software

  Mobile software is an application which is running on a mobile device. In general, there are two types of mobile applications.

  - Native Applications

    Native applications (also known as, resident applications) are application software that only works for specific hardware, so they have better performance. For the same reason, they are not easily portable to other platforms. Additionally, native applications are fat clients, because they locally process the presentation and business logic with data downloaded from the backend servers. As the downloaded data is always stored into the local storage, they occupy most of the available resource space [17].

  - Web Application

    Mobile web-based application rely on the connection between clients (i.e., the mobile device) and servers. And most computations are done on the backend server. Compared to the native application, web applications have higher latency but they have no issues with cross platform deployment. So web applications are powerful alternative to native applications [17].

3.1.3 Mobile Cloud Computing

In the review of cloud computing, we know that cloud computing is a group of servers in the network to provide powerful computation and large storage to serve clients. Whether the client devices have poor performance or not, as long as they can pull and push data to the servers, the servers will respond with a result. Further, we know that in mobile computing, mobile devices are able to exchange input and output data with remote devices. So the advantage of cloud computing can be merged into mobile computing. The research community has named this new combination of cloud computing and mobile technology as mobile cloud computing [25]. Qureshi et al [25] categorized the solutions of mobile cloud computing into two groups as general purpose mobile cloud computing and application specific mobile cloud computing.

- General Purpose Mobile Cloud Computing (GPMCC)

  The general purpose mobile cloud computing helps improve mobile device performance. Although the performance of mobile devices has increased by each device generation, the constraints are still there such as limited processing capability, storage and battery life. In GPMCC, offloading computation from the mobile device to the clouds is the primary approach. When the clients and servers are well connected, the large data and complicated computation will be moved and processed in the cloud [25].

- Application Specific Mobile Cloud Computing

  The application specific mobile cloud computing does not only offload heavy tasks to the backend cloud,

but also provides powerful computation to other devices. Serendipitys [29] is a framework that enables mobile devices to use other mobile devices' computational resource to increase their own computing ability and decrease their own energy use.

Different solutions of mobile cloud computing improve the performance issues of mobile devices, but at the same time, there are lots of potential issues that affect the smooth delivery of mobile cloud computing services.

- Resource Limitation of Mobile Devices

  This is inherent issue in mobile cloud computing. Generally mobile devices (e.g., phones) have less computation power, limited storage capacity and battery constraints as compared to desktop computers.

- Network Bandwidth

  Another issue is limited or fluctuating bandwidth in the mobile cloud network. In some geographical areas, mobile devices can encounter poor signal reception which leads to lower bandwidth [25].

- Intermittent Connectivity

  In mobile cloud computing, seamless connectivity is impossible due to the unreliable wireless connection. The applications experience temporary loss of connectivity sometimes due to user mobility.

- Security Concerns

  Mobile applications mostly require to protect user data. In mobile cloud computing, the security problems are divided into the client side and server side. Additionally the transmission channel also needs to be highly secure.

In a nutshell, mobile cloud computing has changed traditional mobile computing. The general purpose mobile cloud computing and application specific mobile cloud computing both can increase the performance of mobile devices. Although potential issues do trouble mobile cloud computing, it is likely that many can be alleviated through good design.

3.1.4 Google Cloud Platform

Basically Google cloud platform is "Google" because it relies on the same infrastructure that Google currently uses for its own website and applications. Google cloud platform is composed of five main components.

- Google App Engine (GAE)

  It is a cloud platform to deploy and host web and mobile applications. It uses Google's large-scale server infrastructure with 99.95% uptime service level agreement. With GAE, developers do not need to be concerned with physical issues such as CPU, storage and bandwidth. According to the application usage, these demands will automatically increase or decrease as needed [30].

- Google Cloud Datastore

  It is a non-relational database that supports automatic scale up to massive data. And it is a schema-less database that does not require pre-defined data columns [31].

- Google Cloud SQL

  It is a relational database that offers fully managed functionalities for MySQL database. In order to get high availability and durability, all the entire dataset is replicated in multiple locations [32].

- Google Big Query

  It is a service to help analyze massive amounts of data. Based on Google's infrastructure, Google big query is a high speed data analyzer, and there are different user interfaces to interact with queries' results [33].

- Google Compute Engine

  It is an infrastructure as a service that allows clients to create virtual machines (VMs). And these VMs can run large scale computing tasks as efficiently, quickly and consistently as possible [34].

  This research proposes a framework, called MobiQ, to provide anytime, and anywhere access services. In order to achieve this requirement, the server side of MobiQ should be accessible and responsive all the time. The Google app engine with Google's infrastructure guarantees high availability and durability. Therefore, the server side of MobiQ hosts on the Google app engine. Also, MobiQ uses Google cloud data store as database because of high scalability. The reason why MobiQ chooses NoSQL database other SQL will be explained in section 3.3 on Data Store.

3.2 Web Services

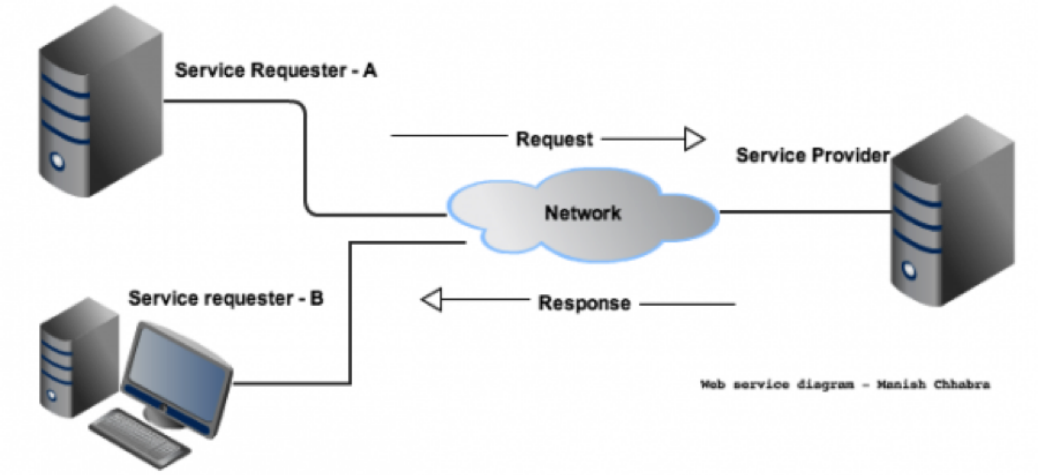

Figure 3.1: The architecture of web service[7]

A web service is a method of communication between clients and servers over the World Wide Web and the figure 3.1 illustrates the architecture of web service. The W3C defines web service as a [13] "software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language). Other systems

interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards". In general, two types of web services exist: Simple Object Access Protocol (SOAP) and the Representational State Transfer (REST).

3.2.1 SOAP

The Figure 3.2 represents the architecture of SOAP-based Web service that consists of three components.

- Service Provider

  It creates and deploys SOAP based Web services. Additionally, it publishes services using the WSDL through service registry such as UDDI.

- Service Registry

  It registers and categorizes published services and also enables the search functionality.

- Service Request

  It uses the service registry to discover the WSDL-described web services. In the description, it gets information about methods and bind information. Based on this information, it calls the service provider.

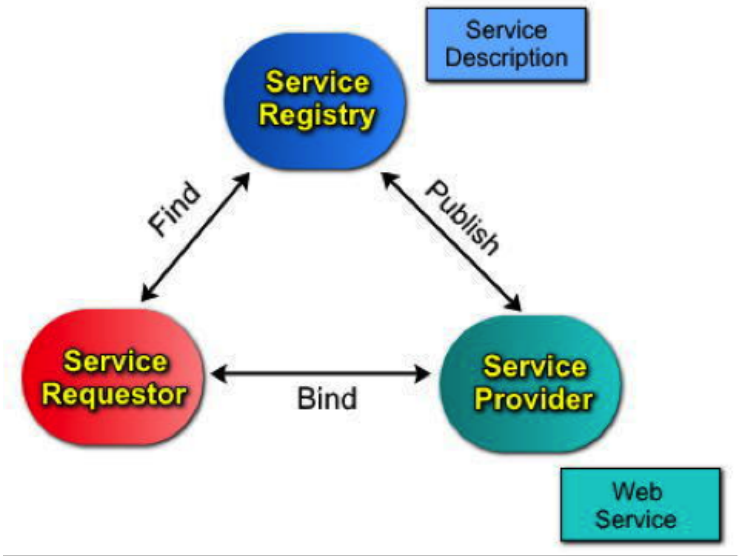

Figure 3.2: The architecture of SOAP based web service[5]

The communication between these three components is based on the SOAP message. The message uses the XML format which commonly exchanged over HTTP, even though other transport protocols such as FTP, TCP or UDP are also possible [3] [40].

3.2.2 REST

Figure 3.3 shows the architecture of the Representational State Transfer (REST) Web service. REST uses the client-server architecture. The client sends requests to the server and the server responds to each request. Like SOAP based web service, REST is independent of the transport protocol, but it is normally over the HTTP. REST relies on four design principles.

- Addressability

  In the REST, each service has unique URI for the identification of the main entities (resources). The source to the resource is linked with these URIs.

- Uniform Interface

  In the REST, the set of operations depends on the application protocol. For HTTP transport protocol, the standard HTTP operations (GET, PUT, DELETE, and POST) are used.

- Representation Oriented

  In the REST, any service is represented according to client requests. For HTTP, the client can specify the represented type (XML, JSON, or YAML) though the Content-Type header.

- Stateless

  In the REST, stateless means the client session data will not be stored on the server. Each request contains all the information that the server needs, and the server never relies on history to answer a request.

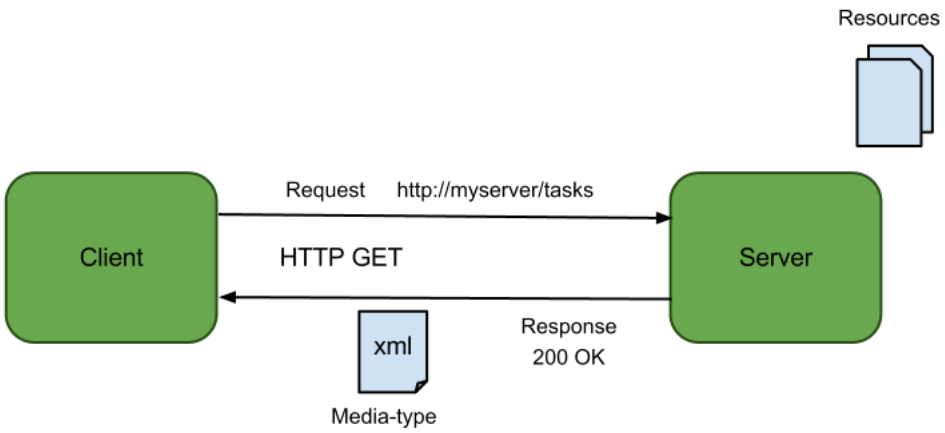

Figure 3.3: The architecture of REST based web service[2]

In a nutshell, REST web service is addressable through specific URI and can be represented in different data formats. Comparing to other Web services, REST is simple, interoperable, and flexible.

3.2.3 SOAP for Mobile

As many web applications are built with SOAP based web service, the assumption that SOAP based web service also is the best choice for mobile applications is made. However, SOAP is designed before the explosion of mobile application. There are some potential problems.

- Limited library-support

  According to DuVander [8], he pointed out that the support for the SOAP-based API is decreasing. This means, the developer interest and the business interest from services providers (e.g., cloud services) are going down.

- Heavy-weighted

  The soap message is based on a large chunk of XML data. Upadhyaya indicated [38] "mostly XML tags are not all necessary for mobile clients". Mobile applications, which have been built and tested using the SOAP standard, also prove to consume high bandwidth and experienced intolerable latency [20] [40].

3.2.4 REST for Mobile

In REST, due to unique URIs, the client can simply invoke services. As a result, it is easier to consume in a mobile application. There are key features for REST in mobile application.

- Simplicity

  The architecture and implementation is simple in REST. Moreover, REST can operate with thin clients to different services types (Java, PHP, Python, .NET and so on).

- Cacheable

  REST is well suited for caching. The request is expressed in the URL. Network caching infrastructure can provide correct response [15].

- Scalable

  By the definition of REST, servers don't manage client session. Due to this design principle, the server is easier to scale [20].

- Efficient

  The message of REST web service supports in different formats such as JSON, XML or YAML. Comparing to SOAP based web service, the message is restricted in the XML format. The client may need other steps to parse messages which cost extra memory and power.

3.2.5 Performance Comparison

SOAP and REST are two major technologies to build web services. In order to choose high efficiency web service technology for mobile applications, the performance of each web service technology needs to be considered. From Potti's experiments [23], the results pointed out that REST has more scalability, interoperability and performance compared to SOAP. Additionally, Belqasmi et al. also highlighted [3] that SOAP takes 10 times longer to process same requests than REST in a mobile environment. In conclusion, REST web service is more efficient for mobile applications.

3.2.6 Summary

In a mobile environment, the mobile client is designed to be thin client. It means there are lots of communications between clients and servers. Therefore, the web service in a mobile environment must be designed to be simple, flexible, and efficient. REST can satisfy all the requirements. In my proposed framework, the cloud server is a REST-based web services for the mobile clients.

3.3 Data Store

For decades, SQL was the most successful database for applications. However, recently there is another option for choosing a database, which is the NoSQL that stands for not only SQL. The "no" in this term does not mean to reject SQL. It means to compensate for the limitation of the SQL system. In order to appropriately choose data store for the proposed framework, this section reviews several fundamental design rules of data store.

3.3.1 ACID

ACID is a set of properties in the traditional database transactions. In order to provide reliable services , these four properties must be guaranteed.

- Atomicity

  All of operations in the transaction will complete. If one of them fails, the entire transaction fails .

- Consistency

  According to Pritcheit[24], "the database will be in a consistent state when the transaction begins and ends". This property ensures that any transaction will bring the database from one valid state to another.

- Isolation

  There is no communication between transactions, therefore transactions cannot see each other during the processing time.

- Durability

  After completing a transaction, the operation cannot be rolled back.

In a nutshell, traditional database can protect data transaction in a single node by following ACID proprieties. However, availability and performance are impaired.

3.3.2 CAP Theorem

In 2000, the CAP theorem which is shown in figure3.4 (Consistency, Availability and Partitioning Tolerance) was firstly introduced by Erick Brewer who is a professor at the University of California, Berkeley [4].

- Consistency

  Regardless of locations, the data should be the same in the cluster - " what you write is what you read ". Additionally, the order of operations must follow.

- Availability

  A service should be available even if a node in the cluster goes down. Moreover, each request has to have a response to indicate if it is successful or failed.

- Partition tolerance

  The cluster can be partitioned into several groups of nodes, which cannot communicate with each other. As a result, the data in some parts of the cluster keeps getting updated, but others are unsynchronized.

It is impossible to achieve all three guarantees together for web services. The tradeoff must be made among the CAP. There are three possible combinations.

- CA (Consistency and Availability)

  With this option, the data must always be consistent in the cluster. As long as the node is online, the data can be updated from others and keep multiple consistent copies. Unsynchronized data only happen, when working a partition between two nodes. In conclusion, this is a good design when servers are in the same data centre [10].

- CP (Consistency and Partition tolerance)

  With this option, the data is synchronized in the cluster. And it supports partition tolerance when all nodes are available. However, if one node goes down, the synchronization breaks down.

- AP (Availability and Partition tolerance)

  With this option, all nodes stay online but data divided into different groups which may not communicate with each other. In this case, data consistency cannot be guaranteed. However, the data will try to synchronize with others when they are available to communicate with each other. Finally it will achieve consistency through gradual replication.

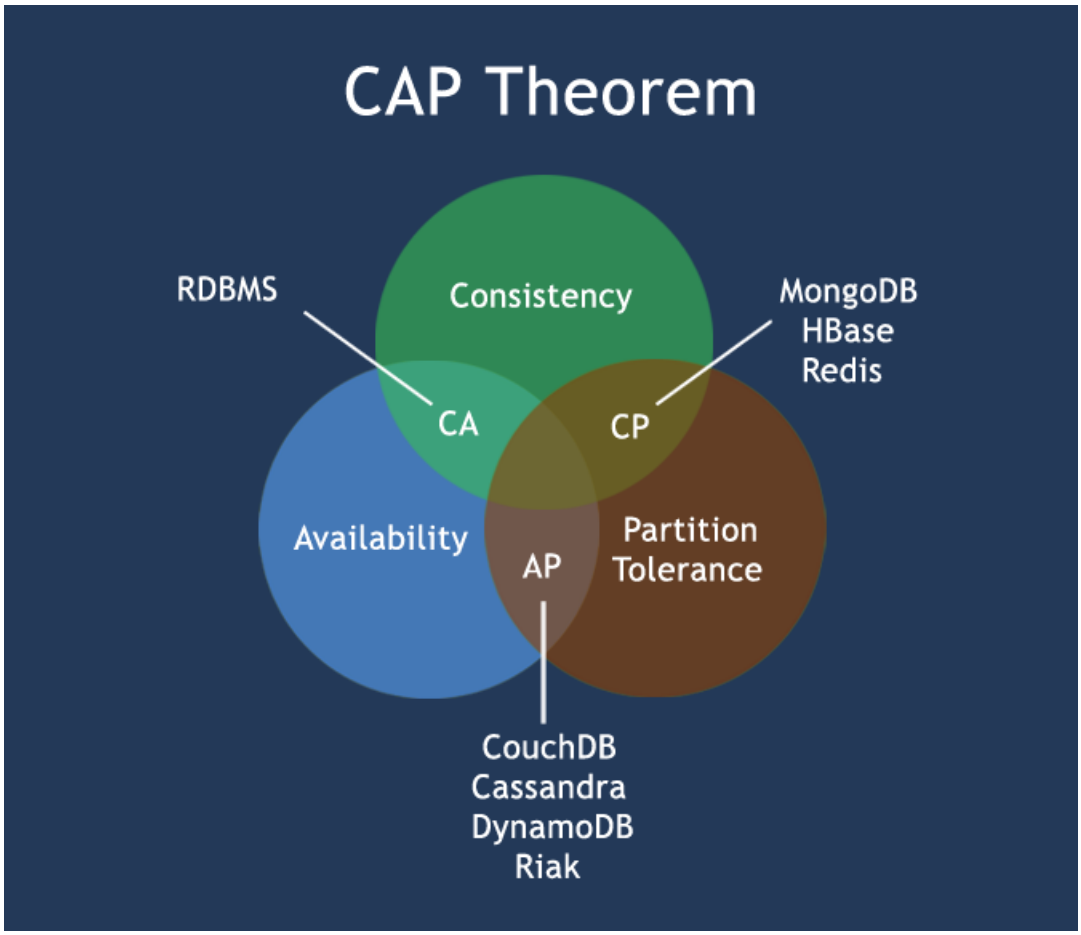

Figure 3.4: CAP Theorem[39]

According to Gilbert and Lynch[10], "the CAP theorem is one example of a fundamental trade-off between safety and liveness." Due to specification of each application, the trade-off also is designed to be different. In mobile wireless network, the intermittent network is common and leads to unpredictable message lose. Under this circumstance, partition tolerance becomes a given. Hence, applications need choose one more option between Consistency and Availability [18].

3.3.3 BASE

BASE [28] is another design philosophy for database and it is absolutely opposed to ACID

- Basically Available

  This option is to guarantee availability. However, the data may be inconsistent in different nodes.

- Soft State

  Due to eventual consistency, the state of the system can be updated even if currently no modification is made

- Eventual Consistency

  The system finally will be eventually consistent. But it may take short or long time to complete the whole synchronization process.

The BASE is a design to trade off consistency in CAP for availability. Consistency does not require synchronization after each update, but the data will be eventually consistent. Comparing to the ACID rule, the BASE improves availability and performance.

3.3.4 NoSQL (not only SQL)

NoSQL is a non-relational database management system. Comparing to SQL, NoSQL has different design principles. It is based on the CAP theorem and BASE rules to provide high availability and performance in database management systems. As NoSQL follows BASE rules, it drops the Atomicity, Consistency and Durability in ACID. Thus it improves scalability in traditional database.

3.3.5 Difference between NoSQL and SQL

NoSQL and SQL are two database management systems. Differences between them are as follows.

- Design Principles

  SQL

  follows ACID rules to build strong consistency and high durability database management system. However, in order to provide consistency in distributed system, it is difficult to scale up [14].

  NoSQL

  follows BASE rules to design database management system. To increase scalability and performance in the distributed system, it replaces strong consistency with eventual consistency. Also introduces MVCC (Multi-Version concurrency control) as another method to keep data consistent. However, different implementations in the application layer may cause inconsistency even if the data is supposed to be consistent [14].

- Data Model

  SQL uses relational data model which provides structured and secured data in the system. However, due to no overlap in data presentation, the system process queries not efficiently. Moreover, the relational data model is possible to create errors in the distributed system [14].

  NoSQL uses a different approach to manage data. Generally, there are three main solutions.

  – Document-oriented. Data is stored as documents such as JSOn or XML formats [14].

- Graph-based. Data is stored as nodes and edges. And there are plain and efficient user interfaces to manage relations. This data model is widely used in social networking sites [14].
- Key-value. Data is stored in schema-less way. Each data has unique key to look up.

- Scalability

  SQL Consistency of ACID means that the same data must be identical. If one node changes, others must be updated. However, the data across different servers in the distributed system cannot ensue the immediately update. Therefore, ACID rule limits scalability in SQL.

  NoSQL are generally designed to work with distributed data. NoSQL follows BASE rules instead of ACID, so they sacrifice consistency and in return it increase scalability.

3.3.6 Summary

In the past, SQL is almost synonymous with databases. As a result, SQL is a one-size-fits-all solution for data persistence and retrieval. As time goes by, new web and mobile applications request more scalability and performance for data store. These new requirements have already challenged one-size-fits-all SQL database. This research proposes a mobile questionnaire system framework which is designed for high scalability. Through the review of difference between NoSQL and SQL, NoSQL is more scalable and flexible than SQL. Therefore, this proposed framework will take advantages of NoSQL.

3.4 Data Synchronization

In a distributed system, data can have many identical copies that are stored on machines located in the different physical places. These copies may be accessed and modified by individual owners. As a result, the original data may have different versions. In order to have a consistent state version for later use, the separated modifications should merge together. This process is defined as data synchronization [36].
In the mobile network, each mobile device can be represented as a node. The nodes intermittently connect to other nodes and maintain the consistency. In this situation, there are two possible synchronization relationships. One is single-device synchronization (1 : 1) and another is multi-devices synchronization (1 : n). In both two synchronization relationships, there must be one node that is always actively updated and send latest modifications to the others. In single-device synchronization, mobile devices only synchronize with the same machine such as personal computer or web server. The timestamps and flags are employed to identify different data. With this relationship, two devices only need to update modifications that are created after the last synchronization. On the other hands, multi-devices synchronization is much more complicated, because the synchronization needs to detect individual modification, put them together, and then eliminate conflicts [36] [1].

3.4.1 Data Synchronization Design Pattern

- Asynchronous Data Synchronization

  With this pattern, the mobile application synchronizes with backend servers without blocking user interfaces. And still runs with local data or caches. However, presented data in the devices may be inconsistent with the backend server during synchronization.

- Synchronous Data Synchronization

  It is a mechanism pattern. when a synchronization process starts, the mobile application has to wait until the synchronization is finished. Obviously, the user interface's thread is also blocked. But this pattern ensures that all data is highly consistent with the server.

- Partial Storage

  This pattern allows an application only to store the data as needed. Therefore, mobile devices decrease the usage of local storage and also optimize network bandwidth.

- Complete Storage

  In this pattern, a mobile application will download all data from servers even if it does not need it right away. Thus, this application has high availability to access data even if the connection is unavailable. However, this design may waste a great amount of local storage and bandwidth.

- Full Transfer

  In order to reduce the conflicts after synchronization, the full transfer pattern requires to upload and download all data for each synchronization request. Therefore, the traffic of duplicated data wastes bandwidth.

- Timestamp Transfer

  Timestamp transfer uses a timestamp as identifier to updates unsynchronized data as need, so it saves bandwidth from redundant data.

- Mathematical Transfer

  This pattern is similar to the timestamp transfer. Besides simple comparison in timestamp, there are a few significant mathematical calculations to detect unsynchronized data.

3.4.2 Data Synchronization in Practices

- Palm HotSync Protocol

  HotSync protocol is a synchronization protocol that helps programs to synchronize data with other devices or servers in Palm's OS. With this protocol, there are two synchronization models, Fast Sync and Slow Sync.

– Fast Sync

There is one prerequisite for this protocol. Mobile devices can only synchronize with the same machine that they synchronize with at first time. Under this condition, a personal computer or web server should host a database that stores the newest updated data. Each mobile client can use its own data timestamps or flags to compare with the last synchronized timestamps in the database. If its own timestamps is greater than last synchronized timestamp, it means that PCs' data is out of date. The timestamp of data only changes when new data is inserted, existed data is updated, or redundant data is deleted in the mobile client. To update modifications, there are several steps. Firstly, PC will check if there are new items created (insertion). Secondly, PC will detect if inconsistent data is stored in the mobile client (modification). Thirdly, PC will find out if the data has already been deleted in the client (deletion). Fourthly, PC will execute modifications on its own database and then update status flags. In a nutshell, this synchronization process relies on each flag of data which is stored in the client and server. As each synchronization evaluate last update flag, it is impossible to synchronize with different PCs [36].

– Slow Sync

There is no prerequisites to execute synchronization process. With Slow Sync, mobile devices can synchronize with multiple PCs or servers. In this situation, the synchronization flags is not as important as in the Fast Sync. During the synchronization, all data in the local storage will send to the PC.Therefore, this protocol is similar with backup data from mobile client to server [36].

In the mobile environment, the intermittent connectivity causes low bandwidth and long latency, so transmission data is designed to be as small as possible. Between Slow Sync and Fast Sync, less data transports in the Fast Sync protocol, because only modified data will sent to the PC from mobile devices. Therefore, this protocol saves bandwidths and improves latency, so it should be an efficient protocol in mobile application system.

3.5 Data Replication

Replication is a technique where data is copied from one server to another in distributed systems. When errors happen suddenly, the server can go down. If this server has already copied its data to others, the same services will still be available. Therefore, users will not be affected by the downtime of the server. The purpose of replication is to increase availability and performance of the servers [11] [36]. In the mobile environment, each device is considered as an individual server which needs to run application services with data from databases. With connection, each mobile client needs to update its database when a new modification happens at the server side. Also, the mobile needs to notify others to update when a new modification happens locally.

3.5.1 Replication in Database

Most servers contain a data store to save information. If many servers share only one database, each server needs to wait a long time to access the stored data. Also, all the servers will be unavailable because of failure of this database. Thus, in addition to the server replication technique, the database also needs to be copied [36]. In the mobile environment, the network is unreliable. When the connection is unavailable, mobile clients still need to live. Therefore, database replication is a strategy to provide necessary data on the client even if there is no connection between mobile devices and servers.

3.5.2 Replication Design Patten

- Single Master Replication

  With this pattern, there is only one primary replica that is stored on the master nodes. The rest of the nodes save the copy of primary replica called secondary replica. The node that stores the primary replica is responsible for updating all the transaction of this systems. And then sends the copy of the primary replica to others. Since only one primary replica can be modified, there are no conflicts in the distributed nodes. Furthermore, the secondary replica in the nodes also can be another primary replica for its subset system, but the primary node is exclusive in a distributed system.

  - Eager Replication in Single Master Replication

    It is also known as the synchronous technique pattern. When a copy changes, the rest is going to modify. This pattern ensures that the consistency of replica in the system. However, due to updates in all copies, it not only reduces performance but also increases transactions. With eager replication, online nodes always provide the newest data of the system when others make request. However, offline nodes may provide out of date data because modifications can happen during disconnected periods. In some systems, they are required with data consistency, so they stop to write new modifications when one node disconnect. In these systems, all data must be consistent in each copy. On the other hand, some systems are more available than the previous system. They allow changes when some nodes are disconnected and only update lived nodes. And when disconnected nodes are available again, they take extra steps to synchronize. Frankly, even if all node connect all the time, they still may meet dead lock or fatal errors because of increased transaction [11]

  - Lazy Replication in Single Master Replication

    This replication pattern is also known as an asynchronous technique. When a transaction happens, the primary replica will update itself and then send new replica to the rest. Each replica contains a timestamp to specify versions. When the rest is received new data and ready to update, firstly they compare the timestamp of incoming data with the local version. If the new one is greater than

the local one, the update starts. But if the local one is greater than the incoming one, the update stops and the new replica is abandoned. Comparing with Eager Replication, this pattern have better performance in response time. Additionally, it does not deal with reconciliation failures and conflicts [11].

- Multi Master Replication

  Different from a single master replication, each node be a master node containing a primary replica. And this replica can modify in any network condition even if this node is disconnected. In this situation, it is possible to cause conflicts with multiple modifications. In order to avoid errors, the systems have to implement specific mechanisms.

  – Eager Synchronization in Multi Master Replication

    This pattern is similar with eager synchronization in single master replication. But a system can have more than one master node.

  – Lazy Synchronization in Multi Master Replication

    This pattern allows arbitrary nodes to update local data. When a transaction is received, the local data will be updated, so there is possibility for conflict when two updates are working with the same data. In order to avoid data lose, timestamps is a common strategy to detect and reconcile conflicts [11].

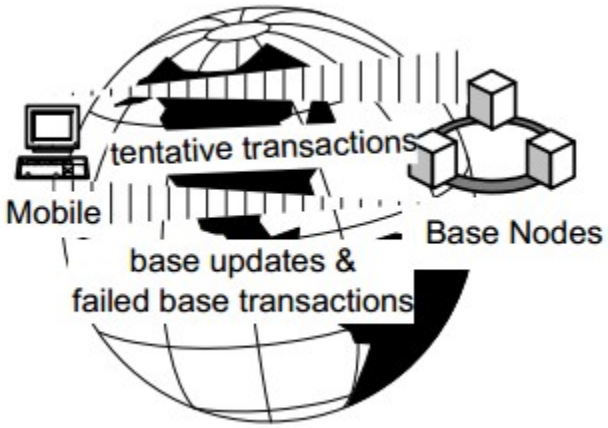

Figure 3.5: Two-tier Replication Architecture[11]

- Two-Tire Replication

  Two-tier replication is a replication solution for mobile applications [11]. In Figure3.5, there are several terms to describe the structure.

  – Mobile Node

    This node is disconnected for a while so that some modifications can be made offline, and these modifications will have to be updated with the online nodes

– Base Node

  This node is always connected to others. Therefore, it has the latest version of replicas.

– Master Version

  This version of data is the newest. However, it is possible to be outdated when it is in the mobile node.

– Tentative Version

  This version of data is modified when the node is disconnected. Thus it is only in the mobile node.

– Base Transaction

  This transaction works in the online nodes. Also, only one of the nodes is unsynchronized.

– Tentative Transaction

  This transaction executes locally when the mobile node is disconnected.

This replication pattern is designed to synchronize with nodes which experience unpredicted disconnections. When this node gets connection again, it has to synchronize with others through several steps as follows. Firstly, the mobile nodes abandon tentative versions of data. Secondly, mobile nodes need to submit changes to all base nodes. Thirdly, mobile nodes need to wait for base nodes, because they need to synchronize with tentative transactions and reconcile conflicts. Fourthly, mobile nodes get copies of the newest replica and install them. Fifthly, mobile nodes will get responses of each update if it is successful or failed. Sixthly, mobile nodes change the status to base nodes. For base nodes, they also need to go through some steps. Firstly, base nodes need to the newest data to mobile nodes. Secondly, base nodes will get all tentative transition from mobile nodes. Thirdly, base nodes will execute all transaction with committed order. And also reconcile all conflicts. Fourthly, base nodes will send correct data to mobile nodes.

In the mobile environment, the application system is looking for a replication pattern that can satisfy availability, scalability, mobility, serializability and convergence. The eager and lazy replication can provide a safest transactional replication and no reconciliation problems. However, there are other problems. First, all nodes has to connect to network. Without connection, there is no way to update. Secondly, the probability of dead lock or error is increased by new joins. Thirdly, lazy synchronization in multi master replication cannot provide ACID serializability. On the other hand, two-tier can satisfy all requirements of mobile applications. Therefore, it should be a good replication pattern for mobile applications [11].

3.6 Data Security

Most mobile applications have to load data from the local storage which can contain sensitive and private data. Therefore, there should be ways to protect data which live in the local storage.

- Encryption is a way to transform data into secured format. And only can be consumed by a person who is intended to consume the data.
- Hashing is a method to create a fixed-length number generated from another number or string. With hashing methods, outputs are not possible to go back to inputs.

In the mobile environment, the energy consumption is extremely important. When considering energy efficiency, algorithms of encryption and hashing have different performances. According to Damasevicius, et al. [12], they concluded "the most energy-efficiency of hash function on a mobile device is SV1 for cryptographic applications, and crc 16 for non-cryptographic application ". And for encryption algorithms Salama et al. [27] stated that Blowfish has better performance than AES, DES, 3DES, RC6 and RC2.

3.7 Online Questionnaire System

Online questionnaire system is a tool that collect data through computers. With online surveys, researchers or business companies can get feedbacks from customers faster and more accuractely. Currently, there are lots of different online tools out there.
The SurveyMonkey is one of the most popular online questionnaire systems. With SurveyMonkey, users build customized questionnaires, send them to specific or general groups, and then analyze results with powerful tools. Currently, the SurveyMonkey only has a web based application. And it does not support offline work.

3.8 Conclusion

Mobile cloud computing consists of advantages of mobile computing and cloud computing. The application of mobile cloud computing increases a mobile device's computation capacity, decrease its energy consumption and also balances its space in the local storage. These improvements greatly benefit the performance of mobile applications. Because of high scalability and availability, a NoSQL database is a good choice to save data in the cloud server. Also, the schema free and simple query features of a NoSQL database benefit for developing. From reviewing data synchronization and replication, each pattern has its own advantages and disadvantages. However, the centralized server architecture, the fast sync and two tire replication are a combination for providing high efficient protocol to keep data consistent. Therefore, this research will propose a mobile questionnaire system that takes advantage of all these benefits.

In summary, the current research indicates:

- Connection Loss
  - Due to the connection may temporally loss, mobile applications need to cache necessary data in the client side.
  - Mobile applications should provide offline storage capability. When the connection goes down, all changes need to save into the local storage. Later, all changes should synchronize with the backend server.
- Data Synchronization Issue
  - The intermittent connectivity can interrupt the real-time synchronization. Therefore, all mobile clients need to firstly update changes in local database and then synchronize with the backend server when the connection restores. The two-tire replication strategy allows mobile clients read, write and update the local database while disconnected with the network. And later synchronize all changes with the backend
- Bandwidth
  - The REST-based web services provide light weight data communication to minimize the bandwidth usage.
  - The data synchronization protocol use timestamps to classify synchronized and need to synchronize data. And only pass data that does not synchronize between the clients and server.
- Security Issues on the Mobile Devices
  - Mobile applications never store passwords in the plain text.
  - Mobile applications need encrypt all sensitive data.
- Security Issues during the Transmission
  - All data should encrypt before sending and decrypt before using.
- Availability and Scalability
  - NoSQL database provide high availability and scalability.
  - Mobile Cloud Computing increases application mobility and scalability.

However, there are still open questions related to mobile questionnaire system :

- How to caching enough data for disconnecting use ?
- How to synchronize with multiple servers ?

Table 3.1: Categorizes and Lists All the Reviewed Research

Cloud computing	• Definition of Cloud computing [21] • Different services of cloud computing (SaaS, IaaS, Paas) [21]
Mobile cloud computing	• Taking advantages from cloud computing and mobile computing [25] • Offloading computations to the cloud server [25] • Providing computations to help other devices in distributed systems [25]
Web service	• The definition of SOAP and REST web service [13] [3] • The mobile platform has limited libraries to consume SOAP services [40] • The SOAP message contains unnecessary data for mobile clients [38] • The REST web service has much better performance than SOAP web service in mobile environment [23] [3]
Data store	• The definition of ACID, BASE and CAP theorem • From the CAP theorem, the three requirements, consistency, availability and partition-tolerance, can be simultaneously guaranteed by two of three[10] • Differences in NoSQL and SQL[14]

Data Synchronization	• Different design patterns in data synchronization • Examples in Fast Sync and Slow Sync[36]
Data Replication	• Different design patterns in data replication • Details of Two-tire replication [11]
Data Security	• Different methods to secure data • Energy efficient algorithm of encryption and hashing in mobile devices[27][12]
Online Questionnaire System	• Review the SurveryMonkey web-based questionnaire system

Chapter 4

Design and Architecture

This chapter discusses the architectural design of the MobiQ framework (Figure 4.1). The architecture is proposed to enforce efficient synchronization of data in volatile network environments. Due to the intermittent network disconnections, mobile applications need efficient synchronization and replication protocols which should consume minimal bandwidth to keep data consistent between the cloud server and mobile clients. In addition, all data should be secured while saving in the local database and passing in the transmission channel. I aim is to advance on the previously proposed two-tire replication concept to build a synchronization protocol and ensure consistent data between mobile clients and the cloud servers.

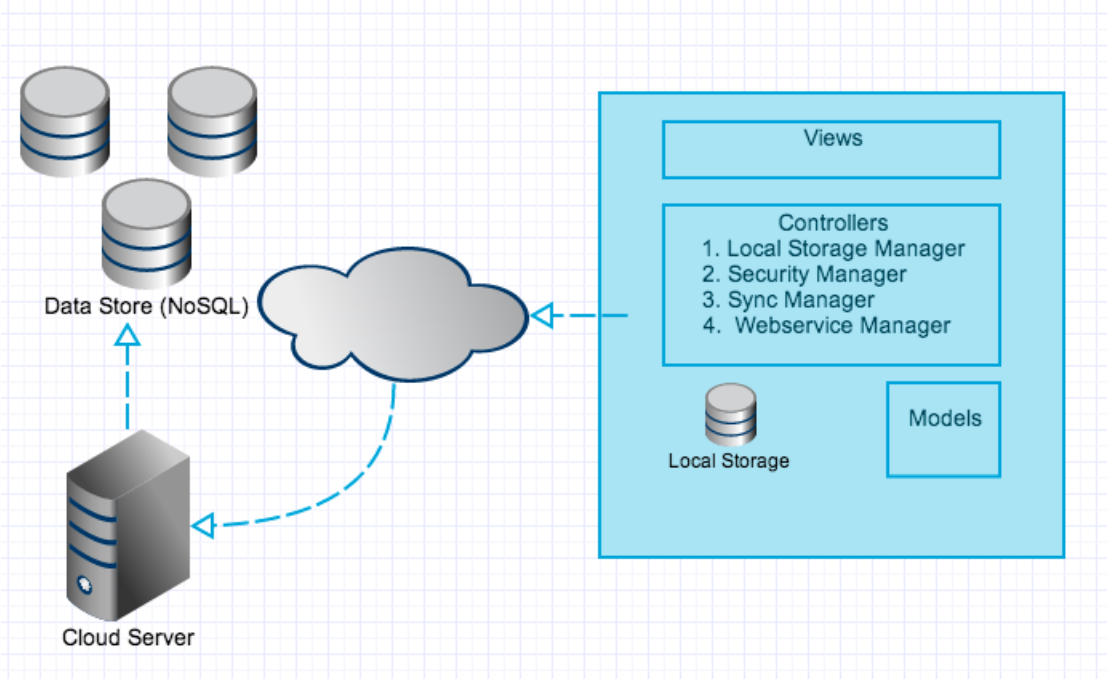

Figure 4.1: The proposed architecture design of MobiQ

The solutions of the MobiQ framework for challenges are discussed in the Chapter 2 are listed below :

- Connection Loss
  - The MobiQ mobile client provides a lightweight database to store necessary cache data.

  - All data in the local database can be accessed and modified in anytime.

- Data Synchronization Issue

  - The MobiQ mobile clients can read, write and update local data in anytime. And all new created changes will synchronize with the server if the connection is available.

- Bandwidth

  - The MobiQ mobile client communicate with the backend through lightweight REST protocol.

  - Each data in the MobiQ mobile client has a property to indicate the synchronize status. And only unsynchronized data will upload to the cloud server.

- Security Issues on the Mobile Devices

  - Any password in the MobiQ mobile client is hashed by the SHA256 hash algorithm before saving into the local storage.

  - All sensitive data in the local database is encrypted.

- Security Issues during the Transmission

  - The MobiQ framework uses highly secured HTTPs protocol to exchange data

- Availability and Scalability

  - The server side of the MobiQ framework is hosted on the Google App Engine to increase availability.

  - The data store of the MobiQ cloud server uses NoSQL database to increase scalability.

4.1 Example Scenario

To understand the architecture and functionalities of MobiQ framework, I will use the example of e-Questionnaire system throughout this chapter. It is a client-server based mobile questionnaire application that helps medical professors in conducting examinations by mobile devices. Before the examination, an administrator will create and download exam questions and check lists to the mobile devices. During the exam, students will answer exam questions with mobile devices. And then they will be asked to diagnose patients. At the same time, professors will evaluate the student's performance with checklists. After the exam, the student's answers and the result of check lists will be uploaded to the backend server. Next, I will introduce these three roles, Administrator, Student, Professor, in details.

Administrator

Figure 4.2: Administrator in e-Questionnaire system

The Administrator is responsible for creating questions and checklists for students and professors through web interfaces. As show in the figure 4.2, the administrator will download necessary data into the mobile devices of students and professors before the examination's date.

Student

Figure 4.3: Student in e-Questionnaire system

Before starting any exam, student must go through an authentication step. A student has to provide his username and password pair after which the student can start his exam. Different types of questions are presented in the examination such as question answer, multiple choices and true or false. When a student finishes the exam, he can upload his answer back to the server if the wireless connection is available. Otherwise, all data will be saved into local data storage. Later, the administrator will manually upload them.

Professor

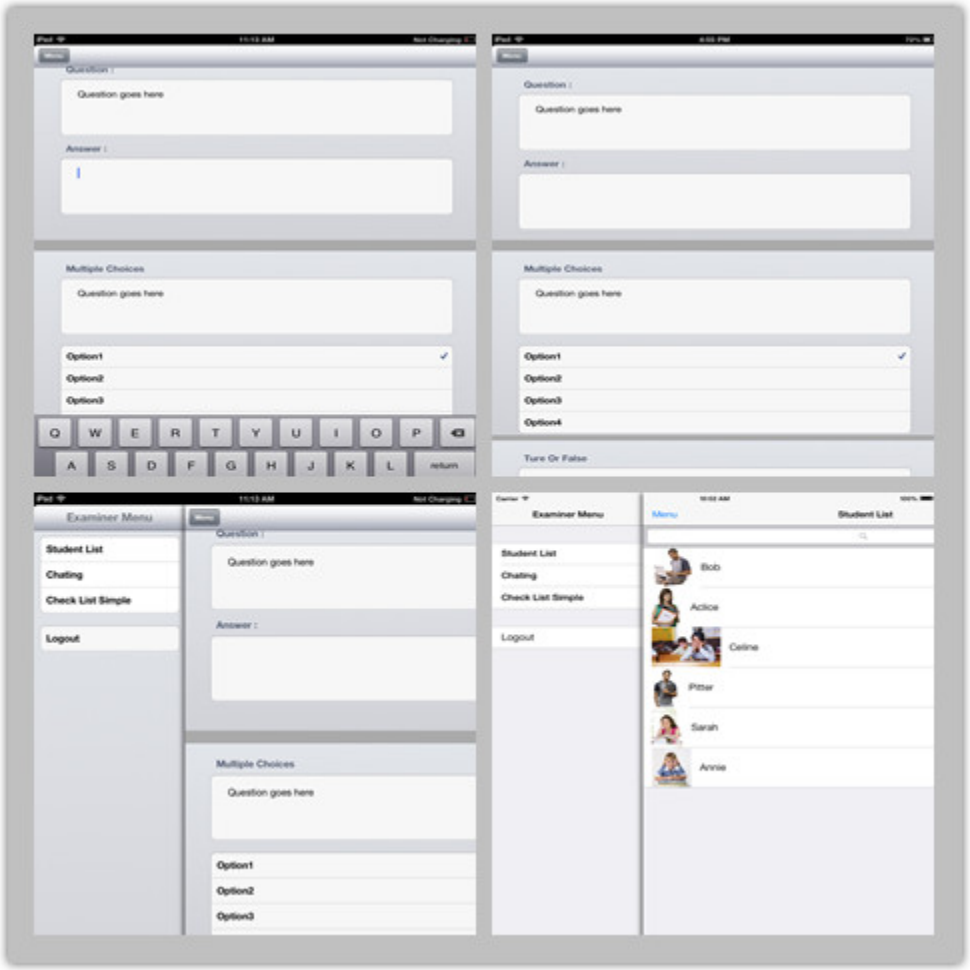

Figure 4.4: Professor in e-Questionnaire system

The Professor will evaluate students' performances when they are diagnosing patients, and give marks to each student according to requirements of the checklist. Similar to the student, a professor also needs to authenticate first. As shown in Figure 4.4, when a professor is logged-in, he has full permissions to access all student information and checklists. Also, a professor can chat with administrators to report real-time status. All new data will be saved into the local database first. When the professor is logged-off, all the data is automatically sent to the backend server if the connection is possible. Otherwise, the administrator will upload all data when the connection is available again.

4.2 Architecture Overview

The aim of the MobiQ framework is to provide a reliable system to consume data. It consists of two main components: the clients and the server.

Client

Due to the processing lack of power in mobile devices, the clients of MobiQ are thin clients that do not compute intensive tasks. They download data from the cloud server and users can read and write downloaded data or locally create new data. If there is updated or newly created data, all changes will be uploaded to the server. There are four manager classes in the mobiQ's client.

- SecurityManager
- LocalStorageManager
- SyncManager
- WebServicesManager

The detail of each manager will be explained in the implementation chapter.

Server

To ensure accessibility at anywhere and anytime, the server of MobiQ is hosted in the cloud and NoSQL data store is provided to save the data. Also, to increase the performance on the server side, the memcache mechanism [35] is implemented. There are two managers in the mobiQ's server.

- WebStorageManager
- AuthenticationManager

The detail of each manager will be explained in the implementation chapter.

4.3 Client Side Architecture

The architecture of MobiQ's client follows the classic Model-View-Controller (MVC) design pattern. According to three different roles of objects in the client, it separates display and processing functionalities. Also, each object can be easily reused. The three roles are:

Model

The objects that are used to store application data and define how to manipulate them. For example, the Question, Answer, Checklist, Student, Professor classes are Models in the e-Questionnaire system.

View

The views are the visualized user interfaces of applications. Users interact with them to control Models and Controllers. For example, QuestionView, CheckListView, LoginView classes are Views in the e-Questionnaire system.

Controller

The objects that are core parts of the applications. According to different inputs from users, they load data from the model and represent it with views. In the e-Questionnaire system, security manager, local storage manager, sync manager and web services manager classes car Controller class.

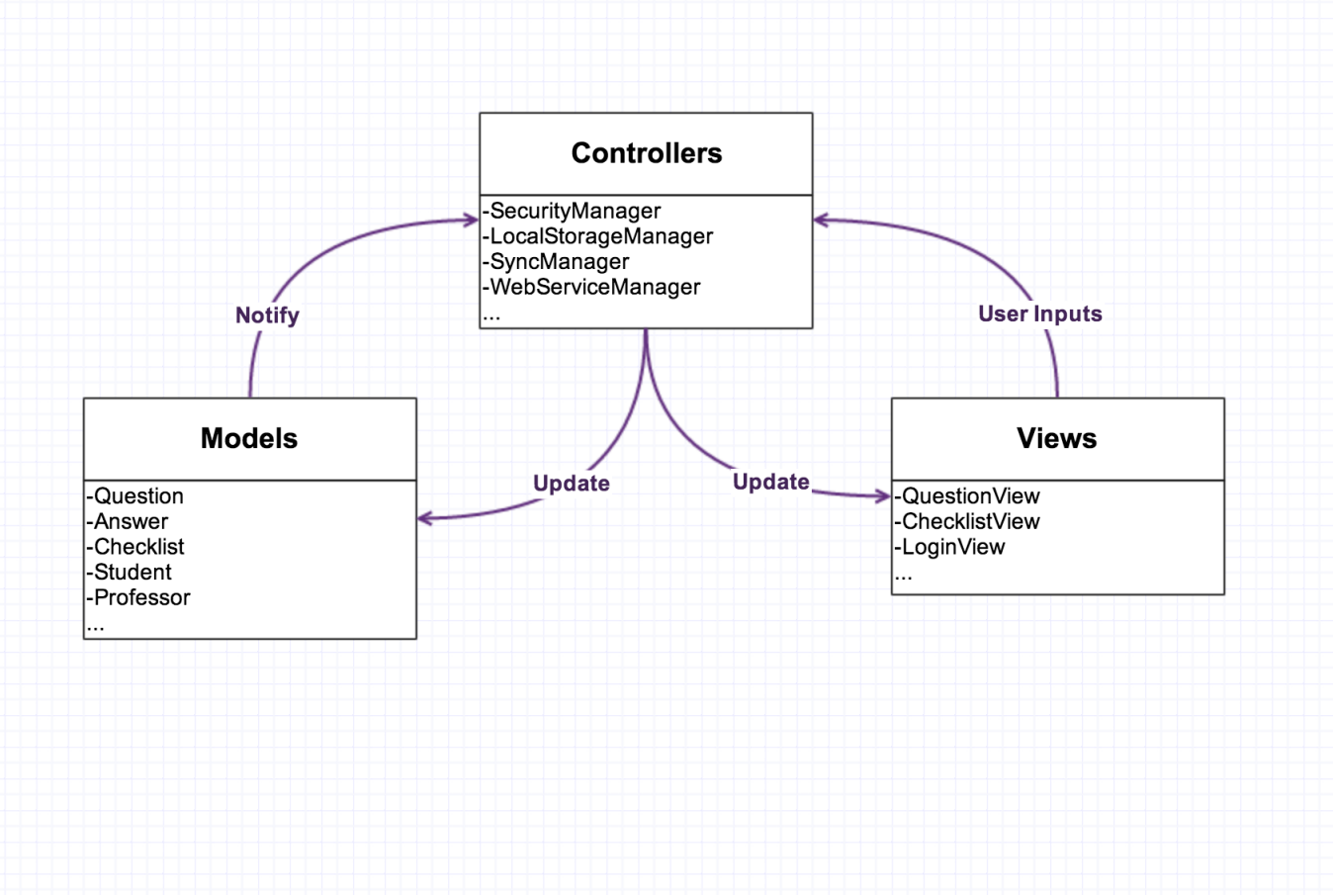

Figure 4.5: MVC in e-Questionnaire System

As show in Figure 4.5, Model classes can notify Controllers when a resource's data is changed. Then the Controllers will update the Views to show the new data. On the other hand, when Views received user's requests, they will ask Controllers to update the Models if necessary or retrieve any other requested data.

4.3.1 Connection Loss

Since the wireless connection is not always stable, the mobile client needs to work in an offline situation. To handle this problem, it is important to:

- Detect if network is available

  The connection status is not always available. Therefore, WebServiceManager has to check out the status of the connection before sending data. If the connection is available, the WebServiceManager makes a request to retrieve data. Otherwise, it will cancel this request. This steps can guarantee robustness of the client application.

- Offline storage

  During the disconnected period, any data cannot be sent to the cloud server. The offline storage is a solution to save data in the local storage. When the network is connected again, the data can be synchronized with the backend server. Also, the offline storage is a performance booster. Users can save a large chuck of data into the device. And then they can quickly access these data instead of waiting for a long time to download again and again.

e-Questionnaire System Example Reference

When the e-Questionnaire system starts, the LocalStrageManager will create a table for Question and Checklist model classes if it uses a SQL database. When administrators select an examination, the Question and Checklist will download from the cloud server and save into the local storage. Then, these data can be accessed in whatever connection status. Also, even if the connection is available, the same Question and Checklist will not download again. It saves time and bandwidth. When students start their examination, they will create Answer objects for each Question object. These objects would be back-up to the cloud server when they are created. However, due to the connection problem, they may not be sent to the backend immediately. The local storage is another place to store them. Therefore, all created Answer objects will not loss. This is very important to the e-Questionnaire system because no students want to do the same tests twice. Later, administrator will collect data and save into the cloud server.

4.3.2 Data Synchronization

Data synchronization is a process to ensure that the client and server have the same data. However, the client and server cannot notify modifications in time during disconnected period. Hence, the MobiQ framework proposed a protocol which is based on the two-tire replication and fast sync strategies. This protocol contains next a few steps to synchronize data.

- All data in the client and server side has a field to save last synchronized timestamps. After disconnecting, mobile clients will check timestamps for all downloaded data and then send them to the backend

server.

- When the cloud server receives a request, it will search all timestamps of the same object requested by the client. And then it will compare the two timestamps from clients and the server. If the timestamps are the same, it means there are no updates on this object. And if the timestamps are not the same, it means there are updates on this object. In the MobiQ system, the server always has the highest priority. Therefore, the system will give up updates from the client and mark this object as unsynchronized object. Finally, the server will send the latest version of the unsynchronized data object back to the clients.
- After getting the response from the server, the client gets a list of unsynchronized object and will use those to update its local data.
- When the newly created objects are related to unsynchronized objects, the client will find them out and delete them. This is to ensure that no unexpected error will happen later.
- After deleting, the rest of the newly created data will be sent to the cloud server.

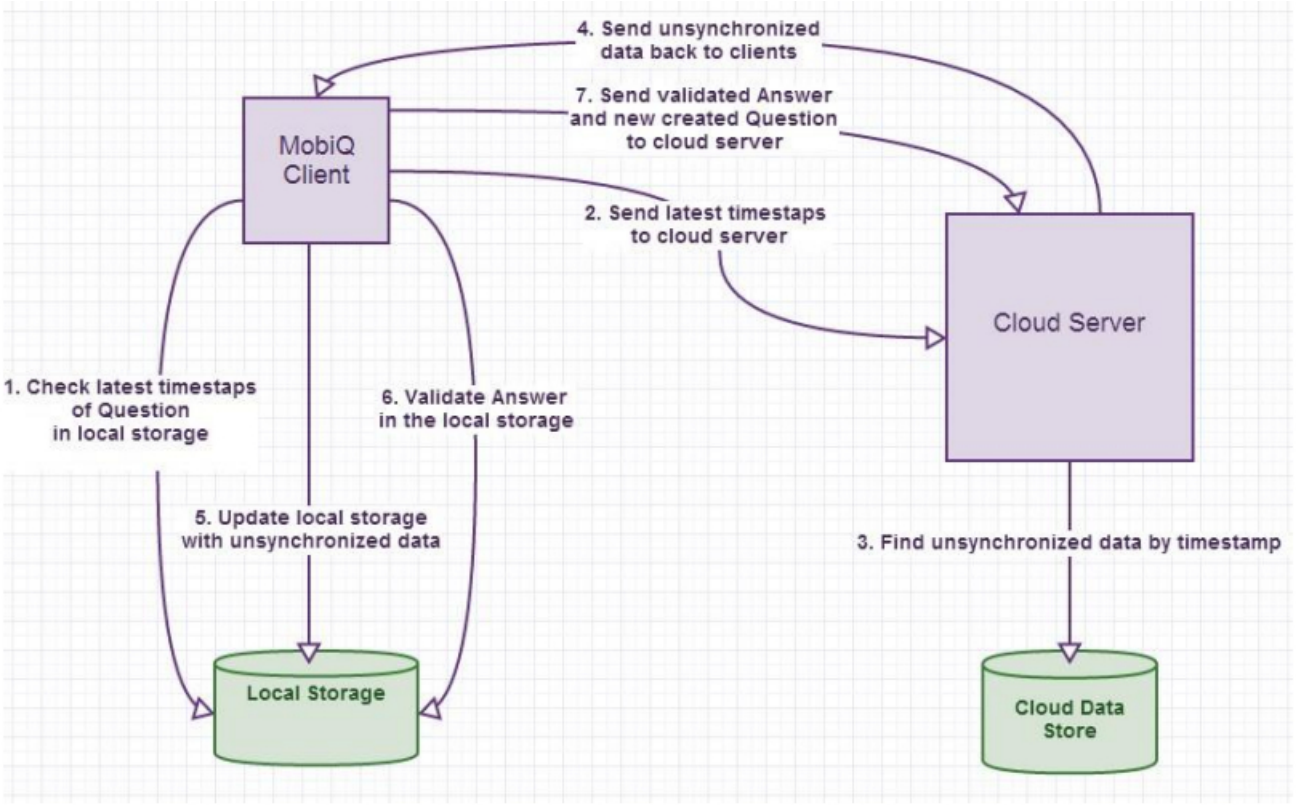

Figure 4.6: The synchronization protocol of e-Questionnaire system

e-Questionnaire System Example Reference

The Figure 4.6 explains synchronization steps of e-Questionnaire system. For example, there are three

questions in the client's local database. Users can access them anytime and anywhere. When the connection breaks down, the server side updates the second question. And at the same time user answers all three questions. When connection is available again, the client tries to synchronize its data with the backend. The mobile application will send the id and last synchronized timestamps of three questions to the server. Then the server will check them. And it will find the second question has already been updated, so it will send the newest version of second question to the client. After getting the new data version, the client application will immediately use the new data instead of the old second question. And then check if there is an Answer object related to the second question. Because users have already answered all questions, there is an answer for second question. Thus, this answer will be deleted. Finally, the application will send the answer of first and second questions to the server side.

4.3.3 Wireless Network Bandwidth

In the mobile environment, the bandwidth is extremely valuable. Thus the MobiQ framework tries to minimize the usage of bandwidth in two ways.

- Only necessary data will pass between the mobile client and server.
- No repeat downloads for the same data

e-Questionnaire System Example Reference

When synchronizes with the cloud server, the mobile client only sends the object id and last update timestamps. It will not send the whole objects. In this way, only updated data will pass between the mobile client and server during synchronization. Also, the client will save all downloaded data into the local database. Therefore, there is no need to download the same data for each request time. Following these rules, the MobiQ framework can save much consumption of bandwidth.

4.3.4 Local Data Security

Security is one of the most important aspects of developing applications. Users always expect the system can keep their data safely. Due to the intermittent connectivity loss, the MobiQ questionnaire system has to save data into the local system. The system proposed two different solutions to secure data.

- Hashing

  Directly storing a password is a bad idea. There are lots of potential reasons to leak data from mobile devices. For instance, if mobile devices are lost, attackers can easily get user's password from the local file system. Therefore, applications should never directly save password. In MobiQ framework, each mobile client has an unique string that is a fixed-length number or string. And it will add to any password string to enhance the security of the password. This new created string can prevent dictionary attack where someone has a list of passwords and go through each one against the password.

After getting new created string, it will be computed by SHA256 which is a secure and one-way hash algorithm so that the password should convert to random unreadable code. Later, this unreadable code will save into the local storage and use to authenticate users during disconnected period.

- Encryption

  Due to one-way hash algorithm, hashed data cannot be reversed. For some objects, they need to be represented again. Therefore, hashing data cannot work for all situations. In MobiQ framework, it provides encryption methods to secure data that should be reconverted.

e-Questionnaire System Example Reference

In the e-Questionnaire system, the students's username and password will download into mobile devices. When the application receives passwords from the backed server, the system will directly hash passwords and store into the local file system. Later, if the connection breaks down and in the same students are trying to login, MobiQ will hash users' inputs with the same hashing algorithm. And only comparing between two hashed password. Because of the same algorithm, the same input can get the same output. Under this condition, if devices are stolen or lost, hackers only can get hashed password which is not the plain value in the clear text. On the other hands, students has to answer questions with mobile devices. The Answer object of this system will be encrypted by the SecurityManager because they may contain private information. When the system prepares to pass Answer objects from the client to server, the SecurityManager will decrypt data and then these data will pass to the cloud server.

4.3.5 Transmission Security

The client and server model application has to frequently pass data. However, the common protocol, HTTP, will not provide security feature. All data is plain text during transferring. In MobiQ questionnaire system, it will use a secure HTTPs protocol to communicate between clients and the cloud server. With this protocol, all data will automatically encryption before entering the transmission channel and also automatically decryption after exiting the channels. Therefore, even if someone can catch data, they are just unreadable strings.

4.4 Server Architecture

Similar with the client side, the server's architecture also follows MVC design pattern.

Model

The model on the server is used to backup the client data.

View

In the MobiQ framework, there are two types of views. One is web interfaces where administrators can create and manage data. Another one is Restful APIs that is exposed to mobile client to retrieve and store data on the cloud server.

Controller

To decouple the relationship between Model and View, the controller will be a medium to notify any change between them. Also, the controller will provide functionalities to increase the performance of MobiQ server application.

4.4.1 Anytime Access

To satisfy the feature, anywhere and anytime accessibility, the server application of MobiQ will be hosted on the cloud. According to Harris [6], "Google blogged this morning about a new no-planned downtime for Google Apps, a promise it's able to make because of its globally distributed infrastructure estimated at more than 1 million servers. Google's expansive infrastructure gives it multiple options for migrating workloads during planned downtime." Therefore, the MobiQ's cloud server will be hosted on the Google app engine. Also, the data store will use Google's no SQL database, data store, to save data on the cloud.

Chapter 5

Implementation of the Mobile Questionnaire System

This section describes the implementation of the mobile questionnaire system. I will separately give details of the client and server side of the mobile questionnaire system.

5.1 The mobile client of the mobile questionnaire system

On the client side,the mobile questionnaire system follows the classic Model-View-Controller(MVC) architecture. I proposed a client view component that facilities interaction with the system, a model component which is driven by a local storage, and the controller coordinates the activities between the views and the model. The upcoming details explain the various components.

5.1.1 Models of the mobile questionnaire system's client

The model component is designed following the questionnaire standard. Generally, there are three basic objects in the client such as Question, Answer and User objects.

Question object

```
public class Question
{
        [PrimaryKey, AutoIncrement]
        public int local_ID { set; get;}
        public int object_ID { set; get;}
        public DateTime createdAt { set; get;}
        public DateTime updatedAt { set; get;}
        public string question_context { set; get;}
        public string option_1 { set; get;}
        public string option_2 { set; get;}
        public bool syncStatus { set; get;}
}
```

1. **local_ID**, which is a unique id for each question object on the local database.

2. **object_ID**, which is a unique id for each question object on the system.
3. **createdAt**, which is a timestamp to record the created time of the question object
4. **updatedAt**, which is a timestamp to record the latest update time of the question object
5. **question_context**, which is a string variable to store the question context.
6. **option_1**, which is a string variable to store the fist option.
7. **option_2**,which is a string variable to store the second option
8. **syncStatus**, which is a flag status to determine if the current object has already synchronized with the cloud server.

In the question object, the object_ID, createdAt, updatedAt, question, option_1 and option_2 attributes are created by the cloud server. The updatedAt attributes is updated when new modifications take place. The syncStatus are created by mobile clients. The default value of the syncStatus attribute is false and it will change to true after the first synchronization with the server.

Answer object

```
public class Answer
{
        [PrimaryKey, AutoIncrement]
        public int local_ID { set; get;}
        public int object_ID { set; get;}
        public int question_ID { set; get;}
        public DateTime createdAt { set; get;}
        public DateTime updatedAt { set; get;}
        public string answer { set; get;}
}
```

1. **local_ID**, which is a unique id for each answer object on the local database.
2. **object_ID**, which is a unique id for each answer object on the system.
3. **createdAt**, which is a timestamp to record the created time of the answer object.
4. **updatedAt**, which is a timestamp to record the latest update time of the answer object.
5. **answer**, which is a string variable to store answers.
6. **question_ID**, which is a string variable to store the id of an answered question.

User object

```
public class User
{
        [PrimaryKey, AutoIncrement]
        public int Id { set; get;}
        public string username { set; get;}
        public string passworld_hased { set; get;}
        public bool login_status { set; get;}

}
```

1. **Id**, which is a unique id for each user object on the local database.
2. **username**, which is a string variable to store the user name of user.
3. **password_hased**, which is a string variable to store the password of an user's password.
4. **login_status**, which is a bool variable to store the current user's login status.

In the user object, the value of password_hased attributed is created by the local security manager.

5.1.2 Controllers of the mobile questionnaire system's client

There are four singleton controllers that are implemented on the client side of mobile questionnaire system. These controllers are discussed as follows

Security manager

The security manager is used to protect data on the local database. There are three core functions.

1. The hash function

```
public string SHA256 (string password)
{
        SHA256Managed crypt = new SHA256Managed ();
        string hash = string.Empty;
        byte[] crypto = crypt.ComputeHash (Encoding.UTF8.GetBytes(password),
                        s0, Encoding.UTF8.GetByteCount(password));
        foreach(byte bit in crypto)
        {
```

```
                hash += bit.ToString ("x2");
            }
            return hash;
    }
```

The hash function will convert given plain texts into another unreadable texts. Also, the input cannot be converted back to the original text. This function will be called when the client download User objects from the server. Because the User object always contains the private password information. Additionally, the same hash function will use to help the mobile questionnaire system to authenticate user in the offline mode. Due to no connection, it is impossible to connect to the cloud server to authenticate. The mobile client will take the user's input to execute the same hash function which is used in the User object. And compare the hashed string, if they are the same, the user can go to the next page. Otherwise, the authentication is failed.

2. The encryption function

```
public string Encrypt (string plainText)
{
        if (plainText == null)
        {
                throw new ArgumentNullException ("plain text is null");
        }
        var data = Encoding.Unicode.GetBytes (plainText);
        try
        {
                byte[] encrpyted =
                 ProtectedData.Protect (data,key,DataProtectionScope.CurrentUser);
                return encrpyted.ToString ();

        }catch(CryptographicException e)
        {
                Console.Out.WriteLine (e.ToString());
                return null;
        }
}
```

3. The dencryption function

```
public string Decrypt (string plainText)
```

```
{
        if (plainText == null)
        {
                throw new ArgumentNullException ("plain text is null");
        }
        var data = Encoding.Unicode.GetBytes (plainText);
        try
        {
                byte[] encrpyted =
                ProtectedData.Unprotect (data,key,DataProtectionScope.CurrentUser);
                return encrpyted.ToString ();
        }catch(CryptographicException e)
        {
                Console.Out.WriteLine (e.ToString());
        }

}
```

These two functions which are paired will use to protect other data such as answer and question object. Each object encrypt before saving into the local database. And also decrypt before displaying on the mobile screen.

Localstorage manager

Due to compare the performance of NoSQL and SQL database in the mobile devices, I implement two different types of databases, SQLite and CouchDB Lite to execute CRUD operations.

Sync manager

The sync manager ensures consistent data between the local storage and the server's datastore, so it needs to communicate with the local storage manager to load data and the web request manager to push and pull data from the server. It follows the seven steps which is described in the chapter 4 to keep data consistent between mobile clients and the cloud server.

Webrquest manager

The web request manager uses secured hypertext transfer protocol to provide the communication between the client side and server side.

5.2 The cloud server of the mobile questionnaire system

In the mobile questionnaire system, the server is hosted on the Google App Engine (GAE) because of its high availability and scalability. The Google High Replication Datastore, a NoSQL data store management system, is used to manage the data store on the cloud. On the GAE, it provides four runtime environments such as Python, Java, Go and PHP. Using the Python environment, I define objects that can be interacted with based on the code snippet below:

```
class Quetion (db.Model):
    question = db.StringProperty()
    option_1 = db.StringProperty()
    option_2 = db.StringProperty()
    createdAt = db.DateTimeProperty(auto_now_add = true)
    updatedAt = db.DateTimeProperty(auto_now = true)
    syncStatus = db.StringProperty()
    object_Id = db.IntegerProperty()

class Answer (db.Model):
    questionId = db.IntegerProperty()
    answer = db.StringProperty()
    createdAt = db.DateTimeProperty(auto_now_add = true)
    updatedAt = db.DateTimeProperty(auto_now = true)
    objectId = db.IntegerProperty()
```

This codes creates the data on the GAE and because the auto_now_add is equal to the Boolean value true, the createdAt property value is set to the first time the model is stored in the data store and it will not be changed. Next, because the auto_now is true, the updatedAt property value is set to the current time whenever the model instance is stored in data store. It means the newest update time will be recorded in this property. The format of the transferred data between the mobile clients and the cloud server is JSON. When the cloud server receives requests to create a resource (e.g., questions) from the mobile clients, it will get related data from JSON objects, create the resource and then save the resource to the data store. The flowing code represents the details.

```
class CreateQuestion(webapp2.RequestHandler):
    def get(self):
        q = self.request.get('question')
        o1 = self.request.get('option1')
        o2 = self.request.get('option2')
        s = 'YES'
```

```
question = Quetion (
    option_1 = o1,
    option_2 = o2,
    quesiton = q,
    syncStatus = s,
    object_Id = Q_OBJ_ID
)
quetion.put()
Q_OBJ_ID = Q_OBJ_ID +1
```

In this code, Q_OBJ_ID is a global variable which is used to create unique ids for the resources. And the mobile questionnaire system will not manually set value for createdAt and updatedAt properties. When the resource is created, the Google High Replication Datastore will automatically set values for them.

5.3 Conclusion

In this chapter, I explains specific model and controllers used in the mobile questionnaire system. Section 5.1, I introduces the core models and controllers in the mobile clients. The question and answer are core object in the system. For the controllers, the security manager will secured data, the local storage manager will persistently save data, the sync manager is keep the consistent data between mobile clients and the cloud server and the web request manager help communication between two side. Section 5.2, I give the detail code of the model in the Google App Engine with Python environment.

Chapter 6

Experiments

This chapter evaluates the functionalities and performance of the mobile questionnaire system in the network volatile environment. In line with the research goal, the following factors are evaluated: offline capability, the performance comparison of NoSQL and SQL database on mobile devices, data Synchronization protocol(DS), performance of the Synchronization Protocol (SP), and Data Protection (DP).

6.1 Experiment Setup

As mentioned in the design chapter, the the mobile questionnaire system is a client-server application. The following systems are used in the whole experiment process.

The Mobile Client

In order to test the system's performance and functionalities, the mobile application runs on the iOS system and various devices.

- iPhone 5s

Hardware	**Specification**
Chips	A7 chips with 64-bit architecture
Capacity	16GB
Wireless	802.11a/b/g/n Wi-Fi (802.11n 2.4GHz and 5GHz)
Model	ME452CH/A
System	iOS 7 (7.1.2)

Table 6.1: Hardware Specification of iPhone 5s

- iPod 5th Generation

Hardware	Specification
Chips	A5 chips
Capacity	32GB
Wireless	802.11a/b/g/n Wi-Fi (802.11n 2.4GHz and 5GHz)
Model	MD720C/A
System	iOS 7 (7.1.1)

Table 6.2: Hardware Specification of iPod 5th Generation

The Server Side

The server is implemented as a Python Web Application. And is host on the Google App Engine. Thus the server application builds and runs on the Google's infrastructure.

Runtimes	Python (2.7)
Framework	Webapp2
Database	App Engine Datastore (schemaless NoSQL datastore)

Table 6.3: Runtime environment of the server application

6.2 Offline Capability

Description

Because mobile networks are not reliable, mobile clients may temporally lose connection. In a disconnection period, the client of mobile questionnaire system should still access previously downloaded question objects from its local storage. There are two Offline Mode (OM) experiments to test the offline capability.

Experiments

1. While being offline, the clients of mobile questionnaire system can still access previously downloaded resources from the local storage.

2. While being offline, users can create response (e.g., answer objects) for downloaded question objects and store the answer objects in the local storage.

Results

In the OM experiments, I evaluated the functionality of mobile questionnaire system in the disconnection period. From the table 6.5, the client application can display previously downloaded objects on the device's screen. Also, the new answer objects also can be created and updated. Therefore, these results show that the

	Read	Write	Update
Local questions (OM 1)	Yes	No	No
Local answers (OM 2)	Yes	Yes	Yes

Table 6.4: Results of offline capability experiments

client mobile application of the questionnaire system has the ability to work in offline mode with previously download objects.

6.3 NoSQL and SQL database on mobile devices

Description

The MobiQ framework uses the local storage to save data and its performance depends on data availability. In this experiment, I evaluate two types of data storage models on the mobile devices. SQLite is a relational database and Couchbase Lite is a JSON Document NoSQL database. And I create all CRUD operations of the database to evaluate performance on the mobile device.

Experiments

1. Insert Benchmark. In this experiment, I create arbitrate numbers of questions and then save them into the local databases.

2. Update Benchmark. In this experiment, I modify the data which has already existed in the device's database. Firstly, I select them from the local database. Secondly, I sign the new value for them. Finally, I save the data back to the local database.

3. Delete Benchmark. In this experiment, I give the specific ids to find the records and then delete the objects in the local database.

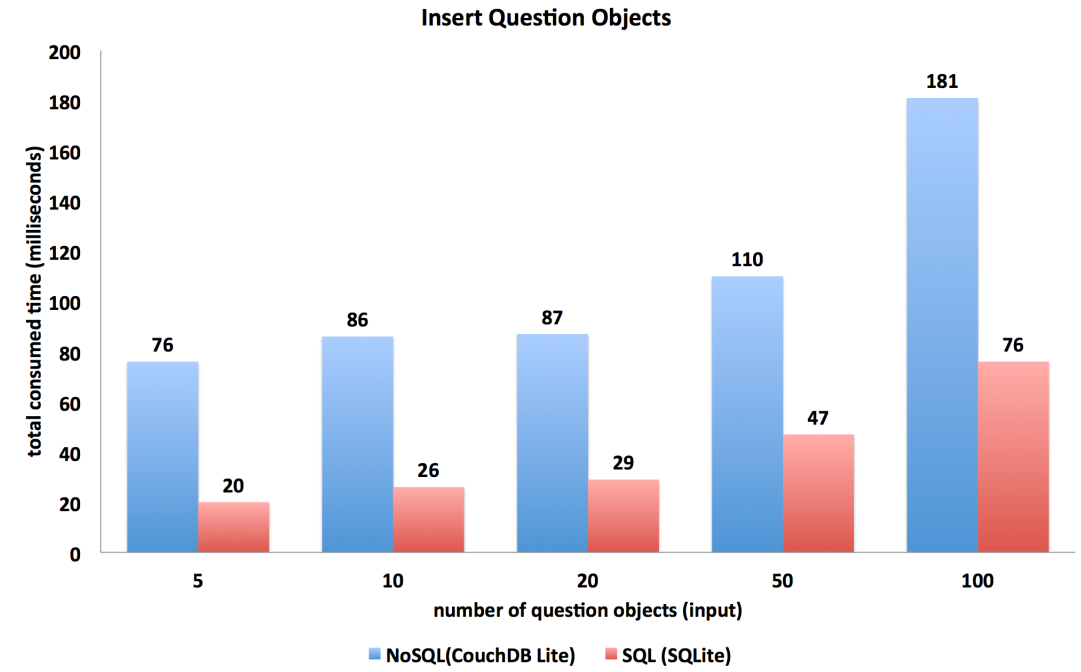

Figure 6.1: Insert objects into the NoSQL and SQL database of the mobile devices

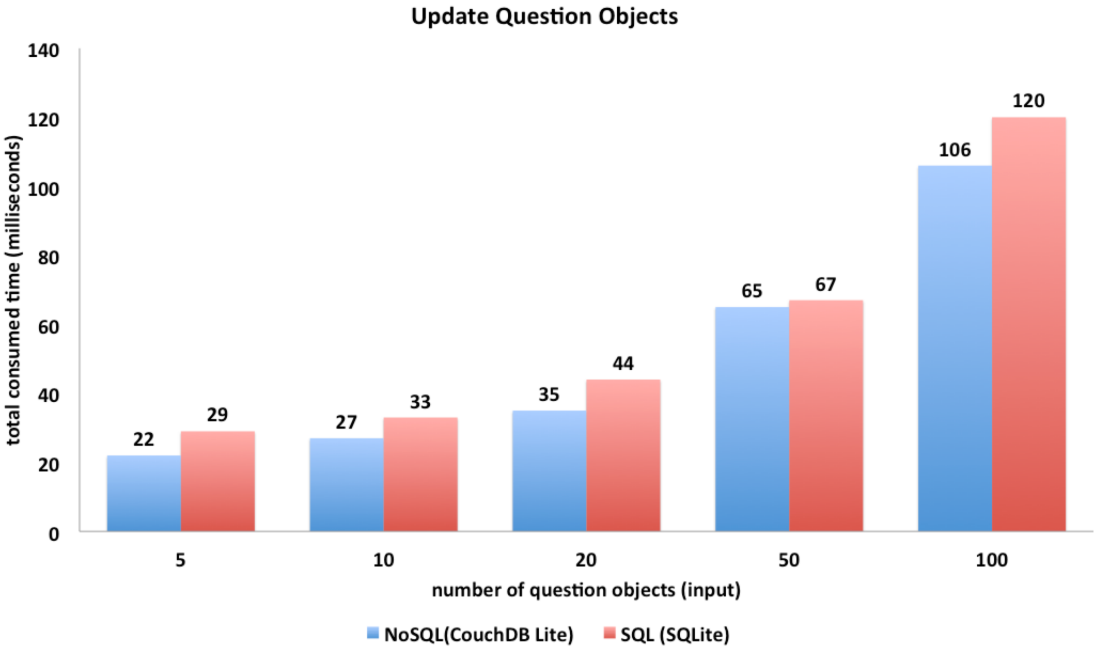

Figure 6.2: Update objects into the NoSQL and SQL database of the mobile devices

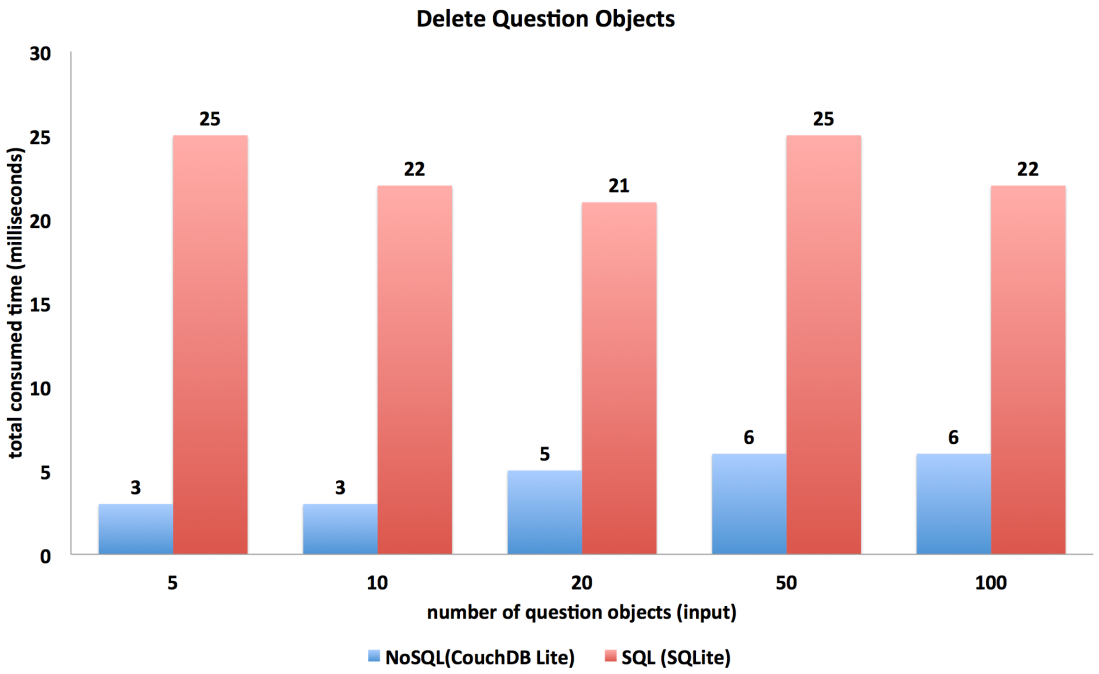

Figure 6.3: Delete objects into the NoSQL and SQL database of the mobile devices

Results

In this section, I set up necessary schemas and then run tests in the given inputs which are five, ten, twenty, fifty and one hundred objects. And I record the time window for each set. In the figure 6.1, the results show the insertion time for different sets of object. In the results, there is no difference among the given number of inputs. The SQLite database can use half time of CouchDB Lite to finish the given task. Indeed, for inserting five and ten objects, the SQLite only use one third time of CouchDB Lite. Thus, the SQLite definitely has better performance in the insertion task. In the figure6.2, I also give the five different input sets to test update performance between SQL and NoSQL database. To test update statement, I change one of fields in the existed object. Thus, the two types of database need firstly find the right object, secondly write new data to the object and thirdly save them back to the database. I record the time for all steps. In the results, we can see CouchDB Lite has a little better performance than SQLite. In the figure 6.3, the results show that deletion time for objects in the SQL and NoSQL database. In the results, the CouchDB Lite has the better performance. All in all, the CouchDB Lite has better performance in the operation which has to do the quires.

6.4 Data Synchronization (DS)

Description

The mobile questionnaire system should synchronize the local data with the cloud server and ensure the consistency of data between the database on the cloud servers and the local storages of the mobile devices.

As already posited, the mobile devices can disconnect from the cloud servers for a while and modifications can happen in that time. As a result, the data between the cloud servers and the mobile clients can be inconsistent. To avoid this, the data synchronization is necessary. There are five DS experiments to evaluate data consistency of the MobiQ framework that I outline below.

Experiments

1. In the disconnected period, the mobile clients can create answer objects (or response or queries). When the connection is restored, the mobile clients will have to synchronize with the cloud servers.

2. While being offline, the cloud servers can create new questions. When connectivity is restored, the mobile clients must get the newly created objects.

3. While being offline, the cloud servers can update their question objects. After connection is restored, the mobile clients must update the new version data.

4. While being offline, the cloud servers can delete question objects from their databases. When the connection is restored, the mobile clients should delete the same objects from their local storage.

5. The mobile clients should be able to delete local answer objects when related question objects are modified by the cloud servers While being offline.

Results

In the DS experiments, I evaluate the synchronization capability of the mobile questionnaire system. In these experiments, I simulate different modification scenarios in the mobile client and cloud server when there is loss of connectivity. In the results, the mobile client and cloud server still have consistent data after synchronizing. The results illustrate that the client-server system can update new modifications and download unsynchronized changes from the server in synchronization process. This process can ensure the consistent of data in the mobile client and cloud server in the mobile questionnaire system.

6.5 Performance of the Synchronization Protocol (SP)

Description

Because bandwidth is limited in wireless networks, the mobile questionnaire system should minimize bandwidth usage when communicating with the cloud server. There are three SP experiments to evaluate the performance of synchronization protocol in the mobile questionnaire system .

Experiments

1. While being offline, the mobile clients can synchronize with the server to get latest updates. Later, the mobile clients can be disconnected from the server. During this time, the servers can create new questions, and the mobile clients must synchronize with the servers to download only the new questions so as to ensure consistent view with the servers.

2. The mobile questionnaire system will use unsynchronized data to check out answer objects. If the related question of an answer objects is changed, this answer object will be deleted.

3. While being offline, the mobile client synchronizes with the servers to get the latest updates, and then the mobile clients disconnect with the server. During this period, the servers can update new questions. Next, the mobile clients must synchronize with the servers to only download new updates to maintain consistency with servers.

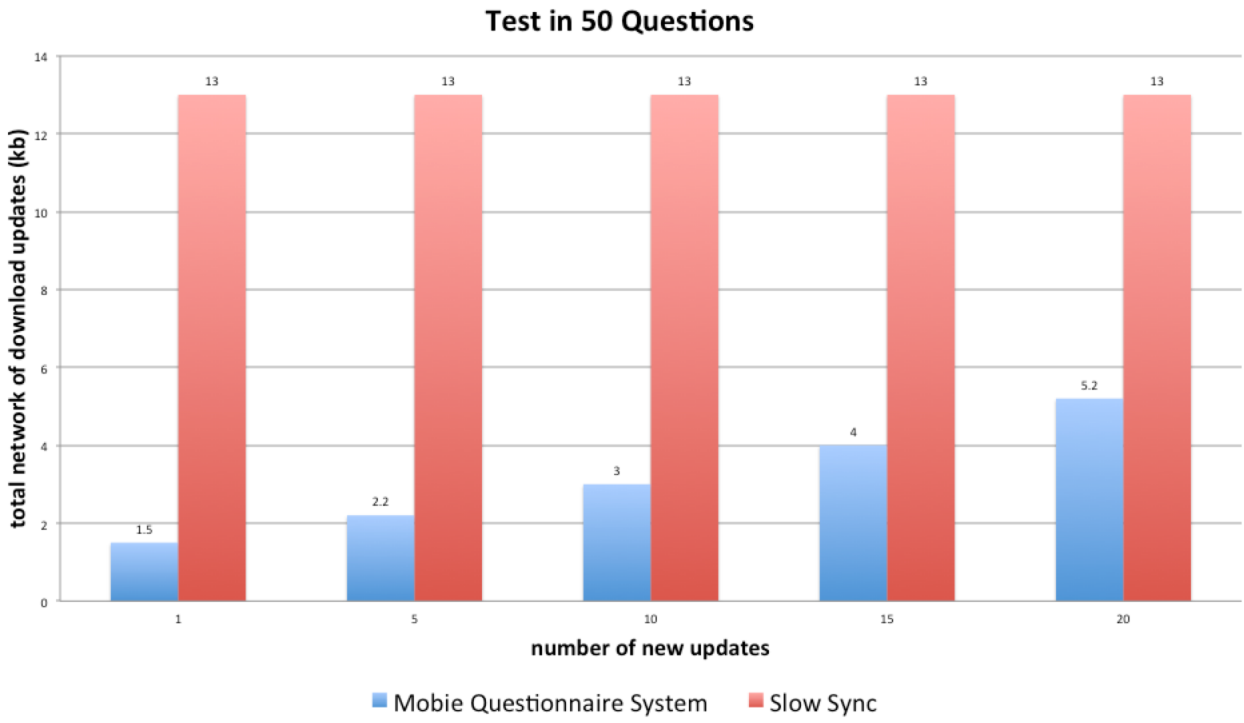

Figure 6.4: Synchronization with 50 question objects

Results

In the SP experiments, I analyzed the bandwidth usage of the mobile questionnaire system's synchronization protocol. From the figure 6.4, there are 50 questions stored on the cloud server for evaluating. For the basic strategy which is named Slow Sync, the application deletes all local records in the mobile devices and downloads all questions from the cloud server. No matter how many changes happened on the cloud server, this protocol downloads all 50 questions for every synchronization request. Thus, the Slow Sync consumes 13 kb for each synchronization process to download questions. On the other hand , the mobile questionnaire system aims not to download all related data every time. It only updates new modifications after the last syntonization. In this protocol, when only one updates happens in 50 questions, the mobile client of the questionnaire system use 1.5 kb to download the new update. Even for the update on the twenty question objects, the system only uses 5.2 kb for synchronization. The figure 6.5 presents the results for testing 100 questions for the performance of the synchronization protocol. In this test, there are 100 questions stored not

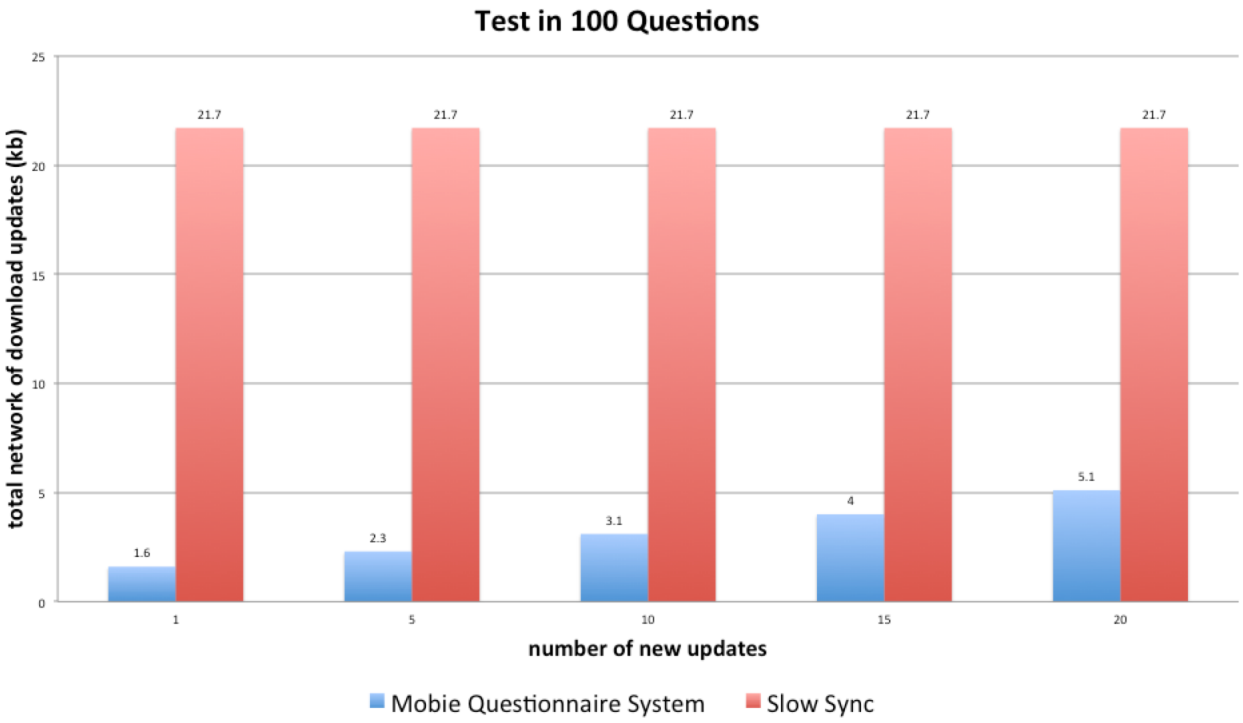

Figure 6.5: Synchronization with 100 question objects

the cloud server. Within the Slow Sync, it consumes 21.7 kb for each synchronization request. On the other hand, the Fast Sync of the mobile questionnaire system only takes 1.6 kb space for updating one change, 2.3 kb for updating five changes, 3.1 kb for updating ten changes, 4kb for updating fifteen changes and 5.1 kb for updating twenty changes. Comparing with Slow Sync, the mobile questionnaire system's synchronization protocol is more efficient.

6.6 Data Protection (DP)

Description

Since plain data in the mobile device is not secured, the mobile questionnaire system should secure sensitive and private data while in use. There are three DP experiments to evaluate the security of the mobile questionnaire system's data.

Experiments

- Because of intermittent loss of connectivity, users' name and password has to be stored into local storage of each device to provide offline authentication. After downloading into the device, this sensitive and private information should be hashed. When hacking tools are employed to look into these data, the characters that are visible should be meaningless. Moreover, these characters cannot convert back to the original data.

- Because of the intermittent loss of connectivity, private data such as users' answer or results of check list should not leak to other people. Hacking tools can be prevented from decoding the meaningless data, but these characters should convert back to original data when they reach the backend servers.
- The mobile questionnaire system framework is a client-server model application. It means that there are lots of communication between clients and servers. The data in the channel of transmission is also designed to securely. When some people use hacking tool to monitor data in the transmission channel, there is no clue to understand the transported message.

	Encryption	**Hashing**
Local questions	Yes	No
Local answers	Yes	Yes
Local password	No	Yes
Data in the transmission	Yes	No

Table 6.5: Results of data protection experiments

Results

In the DP experiments, I mock up different situations to test the data security in the mobile questionnaire system. Since support offline authentication, this system need to save data in the local database. Thus I push the user's password to the mobile client and then use hash function to protect data. The figure 6.6 is the screen shot of the database in the mobile device. I use hack tools to access local database files in the device. And then I look into the private password data. To test the data security, I also show the password in the plain text. After hashing, the password convert to unreadable string by the SHA256 hash algorithm. When the connection is not available, the system authenticate in the offline. It compares the passwords which have already hashed. For demonstrating, I insert the same User object two times to prove the same input can have the same outputs. All the inputs are not be reversed after hashing. On the other hand, some data need recreate when loading from the local database. So, I test the encryption functionality in the mobile questionnaire system. All Question and Answer objects can encrypt before saving and then decrypt before representing. Also, the secured hypertext transfer protocol use to transfer data between the mobile client and cloud server. Therefore, the all data automatically encrypt before entering the transmission channel and also

Table: User

	Id	username	passworld_plain	passworld_hased	login_status
1	1	abc123	12345	5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5	0
2	2	wef889	wef889	cff77138bef51c34ad86fe86ee6d5df418f84dcdacb8fa9ec161776872fcb863	0
3	3	wef889	wef889	cff77138bef51c34ad86fe86ee6d5df418f84dcdacb8fa9ec161776872fcb863	0

Figure 6.6: User's password in the local database

automatically decrypt after leaving from the transmission channel. In the results, the mobile questionnaire system can protect data in the mobile device and transmission channel.

6.7 Conclusion

Experiments in this chapter evaluated the mobile questionnaire system from functionalities and performance. The offline capability experiments prove the mobile questionnaire system can work in unstable wireless network environment. While being offline, the system can work with local data. Additionally, it can add newly created data into the system. In the second experiment, I compare the performance of two types of databases, SQL and NoSQL. The result shows that the NoSQL database have better performance than SQL in the operations which need to query existed data. And the SQL database has better performance for inserting new data. In the data synchronization experiments, I mock up different scenarios to test the data consistent after synchronization. The results show the mobile questionnaire system's synchronization protocol can keep data consistent between the mobile client and the cloud server. In the performance of the synchronization protocol experiments, I compare the synchronization protocol used in the mobile questionnaire system to the Slow Sync protocol. The results show the mobile questionnaire system's synchronization protocol more efficient than Slow Sync protocol. The last experiment focuses on data protection. I use hack tools to access the data in the local database. All data is secured by the secure manager o f the mobile questionnaire system. Therefore, these experiments prove that the mobile question system can work in the offline mode. After suffering disconnection, the system can use its efficient synchronization protocol to keep consistent data between the client and the cloud server. Also, all data is secured in the local database and transmission channel.

Chapter 7

Summary and Contribution

With the mobile shifting, the mobile device is small and portable. And it can provide services to help people to collect data from anywhere and anytime. However, the mobile device, which is different from the desktop computer, has some limitations such as loss connectivity, small bandwidth and poor security. To overcome these limitations, the research demonstrate a novel mobile architecture for the mobile questionnaire system.

The proposed mobile questionnaire system is under a client-server architecture. The mobile client is an iOS platform application. And the server is a Python web application hosting on the Google App Engine. The proposed client is a mobile application. The mobile client can work in the online and offline modes. When there is connection, the mobile client makes requests to the server to pull and push the newest data. On the other hand, the mobile can work with the previous downloaded data which is stored in the local database. During the disconnection period, the application still can read and write data with the local database. For example, users still can be authenticated with the proposed mobile questionnaire system when the connection breaks down. When the connection restores, all modifications will synchronize with the cloud server through the efficient synchronization protocol of the designed mobile questionnaire system. Therefore, data can always keep the consistency between the mobile clients and the cloud server. All steps of synchronization work with the RESTful WS interfaces of the the server. Additionally, the system provide its own police to secure data. The private and sensitive data will encrypt or hash before storing the local database. In this situation, if the device lose, other people is still cannot access the data.

The proposed server side of this system is a Python Web application. And it has been implemented and hosted on the Google App Engine. Google App Engine is the cloud platform provided by the Google. Therefore, the application on the Google App Engine shares the Google's infrastructures. The NoSQL database, data store of Google APP Engine, is used on the server side of the proposed mobile questionnaire system and it provides high scalability.

And the proposed mobile questionnaire system is evaluated by the following experiments.

- Offline capability

    1. Read data when the mobile device is offline

    2. Write data to the local storage when the mobile is offline

- Data synchronization
    1. The mobile client creates new data during the offline
    2. The cloud server creates new data during the offline
    3. The cloud server updates existed data during the offline
    4. The cloud server deletes existed data during the offline
    5. The mobile client automatically deletes answer for non-existed questions
- Performance of the Synchronization Protocol
    1. Only new data will be transferred
- Data Protection
    1. Sensitive and private data cannot readable

Apart from these experiments, I also compare the performance of NoSQL and SQL on the mobile device. This is the preparation for the future work.

In the conclusion, the proposed mobile questionnaire system ensures a questionnaire system can work in an offline mode. Thus, the system is not only to read data during the disconnection period but also can write data to the database. To keep the data consistence, the system also provides a high efficient two-tire synchronization protocol.

Chapter 8

Future Works

To achieve the better usability, there are few other desirable features to add into the proposed mobile questionnaire systems.

- **Peer to peer synchronization**

  The current state of the proposed mobile questionnaire uses a two-tie synchronization protocol to provide data consistency. All synchronization requests must firstly go to the centralized server and then the server pushes new data to the requested client. This synchronization process only works when the connection is restored. As the questionnaire or exam system have unique characters, they often happened in the same location or sharing the same wi-fi resource. If updates are created on the mobile clients, other users who share the same wi-fi resource can directly update their database. The synchronization can finish in the local area and also can be finished immediately. With this feature, only one copy needs to send the server to update the server's database. And other clients are not necessarily push and pull data from the centre server. Therefore, it increases the data availability and decreases the repeated work of the server.

- **NoSQL datastore**

  The current state of the proposed mobile questionnaire system uses a SQL based database on the clients. The format of data models which is structured before using. Therefore, the data schema is fixed and is difficult to change. On the other hand, the NoSQL database is not required to predefined schema before using. The NoSQL is using an unstructured approach to manage data model. For the mobile questionnaire system, the data model is always depends on the different requirement. Therefore, this system need a flexible way to manage uncertain data model.

- **Multiple media**

  The proposed questionnaire system employs plain texts to represent questions and answers. To increase the usability, the novel questionnaire system should add different data source to clearly represent questions and answers. For example, it can add videos to help users to understand the content of the questionnaire. The format of video streaming is different from the plain text. Thus, it should incorpo-

rate new deign that is compatible with the model to store.

- **Cross platform**

  In the proposed questionnaire system, the mobile client is only working with iOS platform on the mac's device. On the mobile market, there are many other operating systems. So, the client of the mobile questionnaire system must be working in the different mobile platforms.

In conclusion, this research involves many different areas of other researches. And they could raise some issues in the current work. Thus, this research, mobile questionnaire system, could still expand.

References

[1] Sachin Agarwal, David Starobinski, and Ari Trachtenberg. On the scalability of data synchronization protocols for PDAs and mobile devices. *IEEE Network*, 16:22–28, 2002.

[2] Vijaya Anand. Creating a rest service using asp.net web api. `http://www.prideparrot.com/blog/archive/2012/3creating_a_rest_service_using_asp_net_web_api`. Accessed Sept, 2013.

[3] Fatna Belqasmi, Jagdeep Singh, and R.Hs Suhib Younis Bani Melhemand Glitho. SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing. *IEEE Internet Computing*, 2012.

[4] Eric A. Brewer. Towards robust distributed systems. *Proceeding of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 7–10, 2000.

[5] Peter Brittenham. An overview of the web services inspection language. `http://www.ibm.com/developerworks/webservices/library/ws-wsilover/`. Accessed Sept, 2013.

[6] Greg Burd. Nosql. `https://www.usenix.org/legacy/publications/login/2011\-10/openpdfs/Burd.pdf`. Accessed Nov, 2013.

[7] Manish Chhabra. Rest and soap web services analogy. `http://blog.manishchhabra.com/2013/04/rest-and-soap-web-services-analogy/`. Accessed Sept, 2013.

[8] Adam DuVander. The next wave enterprises moving soap to rest. `http://blog.programmableweb.com/2012/03/22/the-next-wave-enterprises-moving-soap-to-rest/`. Accessed March, 2014.

[9] Joel Evans and Anil Mathur. The value of online surveys. *Internet Research*, 15:195–219, 2005.

[10] Seth Gilbert and Nancy A. Lynch. Perspectives on the cap theorem. `http://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf`.

[11] Jim Gray, Pat Helland, Patrick E. O'Neil, and Dennis Shasha. The Dangers of Replication and a Solution. *Sigmod Record*, 25:173–182, 1996.

[12] Jim Gray, Pat Helland, Patrick E. O'Neil, and Dennis Shasha. Energy Consumption of Hash Functions. *ELEKTRONIKA IR ELEKTROTECHNIK*, 18, 2012.

[13] Web Services Architecture Working Group. Web services glossary. `http://www.w3.org/TR/ws\-gloss/`. Accessed Sept, 2013.

[14] Derrick Harris. Infrastructure key to google's no-downtime guarantee. `http://gigaom.com/2011/01/14/infrastructure-key-to-googles-no-downtime-guarantee/`. Accessed Jan, 2014.

[15] Shomoyita Jamal and Ralph Deters. Using a Cloud-Hosted Proxy to support Mobile Consumers of RESTful Services. *Procedia Computer Science*, 5:625–632, 2011.

[16] Panagiotis Kalagiakos and Panagiotis Karampelas. Cloud Computing Learning. *5th International Conference on Application of Information and Communication Technology (AICT)*, pages 1–4, 2011.

[17] Dejan Kovachev, Yiwei Cao, and Ralf Klamma. Mobile Cloud Computing: A Comparison of Application Models. 2011.

[18] Richard Kwadzo Lomotey. Enabling mobile devices to host consumers and providers of web services. M.sc. thesis, University of Saskatchewan, March 2012.

[19] Richard Kwadzo Lomotey and R. Deters. Reliable Consumption of Web Services in a Mobile-Cloud Ecosystem Using REST. *IEEE First International Symposium on Service Oriented System Engineering (SOSE)*, pages 13–24, 2013.

[20] Richard Kwadzo Lomotey and Ralph Deters. SOPHRA: A Mobile Web Services Hosting Infrastructure in mHealth. *IEEE First International Conference on Mobile Services*, pages 88–95, 2012.

[21] Peter Mell and Timothy Grance. The nist definition of cloud computing recommendations of the national institute of standards and technology. National Institute of Standards and Technology, Information Technology Laboratory, 9 2011.

[22] Gay Helen Perkins. Will Libraries' Web-based Survey Methods Replace Existing Non-Electronic Survey Methods *Information Technology and Libraries*, pages pp. 123–126, Perkins, G. H.

[23] Pavan Kumar Potii. On the design of web services: Soap vs. rest. M.sc. thesis, University of North Florida, August 2011.

[24] Dan Pritcheit. BASE: An Acid Alternative. *ACM Queue*, pages 105–114, May 2008.

[25] Shahryar Shafique Qureshi, Toufeeq Ahmad, Khalid Rafique, and Shuja ul islam. Mobile cloud computingas future for mobile applications Implementatio methods and challenging issues. In *IEEE International Conference on Cloud Computing and Intelligence Systems*, 2011.

[26] M.N.O. Sadiku. Cloud Computing: Opportunities and Challenges. *IEEE Potentials*, pages 34–36, 2014.

[27] Diaa Salama, Hatem Abdual-Kader, and Mohiy Hadhoud. Studying the Effects of Most Common Encryption Algorithms. *International Arab Journal of e-Technology*, 2, 2001.

[28] Vatika Sharma and Meenu Dave. SQL and NoSQL Databases . *International Journal of Advanced Research in Computer Science and Software Engineering*, 2:20–27, 2012.

[29] Cong Shi, Vasileios Lakafosis, Mostafa H. Ammar, and Ellen W. Zegura. Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices. *MobiHoc'12*, pages 145–154, 2012.

[30] Google Developers Site. Google App Engine. `https://cloud.google.com/products/app-engine`. Accessed Nov, 2013.

[31] Google Developers Site. Google Cloud Datastore. `https://developers.google.com/datastore/docs/overview`. Accessed Nov, 2013.

[32] Google Developers Site. Google Cloud SQL. `https://developers.google.com/cloud-sql/docs/introduction`. Accessed Nov, 2013.

[33] Google Developers Site. Google Cloud SQL. `https://developers.google.com/cloud-sql/docs/introduction`. Accessed Nov, 2013.

[34] Google Developers Site. Google Compute Engine. `https://developers.google.com/compute/docs/`. Accessed Nov, 2013.

[35] Google Developers Site. Memcache Python API. `https://developers.google.com/appengine/docs/python/memcache/`. Accessed Nov, 2013.

[36] Alexander Stage. Synchronization and replication in the context of mobile applications. `http://www14.in.tum.de/konferenzen/Jass05/courses/6/Papers/11.pdf`. Accessed Jan, 2013.

[37] Tutorialspoint. Mobile computing brief overview. `http://www.tutorialspoint.com/mobile\_computing/mobile\_computing\_overview.htm`. Accessed Feb, 2013.

[38] Bipin Upadhyaya. Migration of SOAP-based Services to RESTful Services. *13th IEEE International Symposium on Web System Evolution*, pages 105–114, 2011.

[39] w3resource.com. Nosql. `http://www.w3resource.com/mongodb/nosql.php`. Accessed Sept, 2013.

[40] Qian Wang. Mobile cloud computing. M.sc. thesis, University of Saskatchewan, January 2011.