Performance evaluation of embedded mobile databases: RDBMS VS NoSQL

D.M. Nilanka Chathuri Dissanayaka

129153K

Faculty of Information Technology

University of Moratuwa

March 2015

Declaration

We declare that this thesis is our own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Name of Student   Signature of Student

Nilanka Chathuri Dissanayaka

Date:

Supervised by

Senior Lecturer

Mr. Saminda Premarathna   Signature of Supervisor

Date:

Dedication

In dedication of

My loving mother

Who is strength of my life.

Acknowledgement

Immeasurable appreciation and deepest gratitude for help and support extended to the following persons who is one way or another have contributed in make this research success.

Mr. Saminda Premarathne, Senior Lecturer, University of Moratuwa as supervisor for his support, guidance ,valuable comments and suggestions to conduct this research successfully. Rather than being research adviser he helped all the students to develop knowledge in many ways.

I would also like to thank all the lecturers and staff of the faculty of Information technology, University of Moratuwa for their support and assistance through-out the course.

I would also like to thank head of the department and staff members of Information technology department in Hutchison Telecommunication Lanka (pvt) Ltd for their valuable support for completed my research.

I would like to express many thanks to all the colleagues and others for the each and every support given to make this a success.

Abstract

Most embedded mobile databases are customized to perform requirement of the application. Considering embedded databases due to it doesn't have database administrator they are really robust and failing percentage is very less. Performance characteristics of the embedded database is need to match exactly with the performance characteristics of the application it's resides in, there can be no poverty.

Hence embedded database in android mobile phones and tablets has vast responsibility to cater high performance to end users without disappointing their expectations. Therefore aim and objective of this research is evaluated which embedded database is perform well in mobile environment.

Analyzing current available embedded databases in mobile devices it can observe Relational databases management systems are ruling at the top. However recent huge popularity of NoSQL databases for cloud computing has inspired initiative non-relational data types the essence of big data flooding in organizations and large datacenters.

To evaluate performance of mobile embedded database locally installed SQLite and CouchDB on android environment and integrate android application to which may interact with both databases. I have designed the database which may suitable for both databases and schema will be populated by a FeildGenerator class that used an Xorshift random number generator and arrays of predefined data to generate reasonable tables as efficiency as possible large datasets.

While executing predefined benchmarking queries, benchmark matrices have been evaluated. Once the benchmark metrics calculate , able to demonstrate from RDBMS vs NoSQL which database is performing well on mobile embedded environment.

Table of Contents

List of Tables

List of Figures

Chapter 01

Introduction

1.1 Introduction

Considering most demanding and fast growing technological devices in present world without any debating android mobile phones and tablets are in the top rank. Along with evolution of telecommunication technology from 2G, 3G to 4G mobile devices technology also been rapidly growing up. Due to the fact of android is open source it has massive advantage of integrate with open source embedded databases and bring market share high.

Mobile phones and tablets are well capable to do a large amount of processing without influencing of cloud services. Most mobile applications are data driven and their performance mostly depends on data availability. With rapidly development of the devices and systems support for mobile computing, the embedded database management system becomes more and more important.

1.2 Background and Motivation

As more users adopt Wi-Fi enabled laptops with increasingly capable mobile devices, the need for mobile application is increasing. Applications like Email, Navigation, CRM (Customer Relationship Management) are already targeted mobile devices. Middleware infrastructure like application server and workflow services are becoming mobile-aware. According to Microsoft Cooperation's Nori [1] reasons for such mobility trends are:

- More employees are mobile. Email and offline access is becoming pervasive.

- Mobile usage is broadening. Mobile usage is already prevalent in certain vertical domain like E Commerce, Healthcare, Insurance and Field Services.
- Mobile applications are more than just browser windows – more and more applications now run natively on mobile devices.

Data management and access on mobile devices is central to mobile applications. As mobile applications achieve widespread adoption in the enterprise, mobile and embedded DBMS need to support such applications become an important part of the IT infrastructure. And as these applications grow more discounted and sophisticated with increasing data sizes the need for rich data processing capability increases.

Analyzing current available embedded databases in mobile devices it can observe Relational databases management systems are ruling at the top. However recent huge popularity of NoSQL databases for cloud computing has inspired initiative non-relational data types the essence of big data flooding in organizations and large datacenters. NoSQL databases claim to deliver fast performance than legacy RDBMS system in various use cases, most notably those involving big data. While this is oftentimes the case but not all NoSQL db are created alike where performance is concerned [2].

However there are very less research has been done testing how a NoSQL databases performs on the limited hardware like mobile devices.

1.3 Aim and Objective

Aim:

Aim of this research is identify most suitable embedded mobile database by considering performance evaluation of RDBMS Vs NoSQL DB. Mainly performance evaluate by comparing locally installed NoSQL DB and RDBMS.

Objectives:

Study seeks answers for following research questions

1. What is the proper approach for performance evaluation of mobile embedded database?
2. What tool and techniques are appropriate for the select approach?
3. What aspects giving better performance comparing RDBMS Vs NoSQL DB?
4. Which mobile embedded database is performing well?

1.4 Structure of dissertation

Chapter 02 describes the Present tendency of the mobile embedded databases. Chapter 03 on technology enrichment of embedded mobile db. Chapter 04 on Approach to benchmark the mobile embedded db. Design of benchmarking mobile embedded db will preset on chapter 05. Chapter 06 on Implementation process of experiment and chapter 07 will present the evaluation. Final conclusion may can obtain from chapter 08.

Chapter 02

Literature Review

2.1 Introduction

Thorough out the IT industry there are several stranded benchmarks available for databases, mobile devises and embedded databases. Some of these benchmarks are according to IEEE stranded and advocate to benchmarking. We may obtain several bench marks for RDBMS, NoSQL databases. However theses benchmarks are available and targeting only individual component at a time such as benchmarking databases, mobile devices and embedded databases separately.

In general, there are no standard bench marks for mobile based data stores. There is some recent work in proposing evaluation of SQL based embedded databases on mobile phones. However, there is very less research has done to which compares locally installed NoSQL databases vs RDBMS.

2.2 Prominent databases trends and benchmarking.

Currently there are more than 190 different types of databases available in the world wide [3]. There are lots of new good designed database are entering to the industry providing good horizontal scalability for simple read/write database operations distributed over many servers. According to Cattell [4] in contrast, traditional database products have comparatively little or no ability to scale horizontally on these applications. One of the famous trends in new designed databases is “NoSQL” data stores. The definition of NoSQL, which stands for “Not Only SQL” or “Not Relational”.

A key feature for NoSQL system is “ shared nothing” horizontal scaling – replicating and partitioniing data over many server. This allows them to support a large number of simple read/write operations per second.This simple operation load is traditionally called OLTP (online transcation processing), but it is also common in modern web application.

Since Relational databases are dominant in exposed databases it has been hugely influence in embedded databases in mobile devices. NoSQL databases are finding significant and growing industry use in big data and real-time web applications.

For database evaluations there are some stranded benchmarks currently available such as follows

- Benchmark for large data center level systems : Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) style benchmarks[4][5]
- Benchmark for SQL based enterprise applications : Transaction processing performance council (TPC) [6]
- Benchmark for NoSQL based cloud serving system : Yahoo Cloud Serving Benchmark (YCSB)[8]

In year 2013 Difallah etc all [5] have been conducted OLTP bench mark for relational databases to contribute to better repeatability and easier comparison of results for evaluating RDBMS performance. They present OLTP –Bench, an extensible, open source testbed for benchmarking DBMSs using a diverse set of workloads. The testbed was capable of driving relational DBMSs via standard interfaces, tightly controlling the transaction mixture, request rate and access distribution of the work load and automatically gathering a rich set of performance and resource statistics. In addition to testbed they have implemented 15 bench marks in OLTP -Bench. In this research they have introduced an automated and extensible framework to setup, run an alyze the results of RDBMS performance experiments with controlled and repeatable settings. Also introduced real datasets and synthetic generators, along with their accompanying workloads, that span OLTP/Web applications, all implemented in the same framework .But in this research they have not touch the embedded RDBMS.

One of the most siginificant research conduct about scalable SQL and NoSQL Datastores by Cattle in 2011 [4]. He examine a number of SQL and NoSQL data stores designed to scale simple OLTP-style application loads over many servers and these systems are designed to scale to thousands or millions of users doing updates as well as reads, in contrast to traditional DBMSs and data warehouses. He contrasts the new systems on their data model, consistency mechanisms, storage mechanisms, durability guarantees, availability, query support, and other dimensions. These systems typically sacrifice some of these dimensions, e.g. database-wide transaction consistency, in order to achieve others, e.g. higher availability and scalability. However in his research he has individually compare RDBMS and NoSQL databases but not performance comparison both databases.

There is some recent work in proposing evaluate of SQL based relational embedded databases on mobile phones by McObject. [7].They have taken two quite different embedded database systems that have emerged for use on Google's Android are SQLite, a relational embedded database that supports the SQL application programming interface (API), and Perset, an object-oriented embedded database that works directly with Java objects. To compare Android databases in an "applets to applets" test, they developed the Test Index benchmark application, which measures performance along the insert, search, scan and delete parameter for 10,000 records. According to their experiment results SQLite performance was quite good at android environment. However in this research too they have not tested NoSQL database on Android environment.

Yahoo! Cloud serving benchmark (YCSB) is the one of the best known NoSQL benchmarking application [8]. YCSB framework, with the goal of facilitating performance comparisons of the new generation of cloud data serving systems. It define a core set of benchmarks and report results for four widely used systems YCSB uses data gathered from statistical distribution of data instead of real data. YCSB mainly focus on read, write, scan, update performance of the database. YCSB results are for elasticity is not conclusive. However YCSB only focus on NoSQL databases on cloud environment and it may not addressing the performance compression with RDBMS.

2.3 Summary

There are several database benchmarks available worldwide. Some of them are OLTP, TPC and YCSB. However above benchmarks are focusing individually on RDBMS or NoSQL databases.

Most of database performance evaluation conducted on locally installed operating system and there are only limited number of research conducted in mobile environment. Comparing with RDBMS there are very less research has been done testing how a NoSQL databases performs on the limited hardware like mobile devices.

In general, there are no standard benchmarks for embedded mobile databases. Hence conduct performance evaluation of mobile embedded database type RDBMS Vs NoSQL is worth to conduct.

Chapter 03

Technology Adapted

3.1Introduction

Mobile technology is one of the rapidly growing up technology. In 2003 with the introduction of Android operating system it has been vastly developed. Android powers hundreds of millions mobile devices in more than 190 countries around the world. It's the largest installed base of any platform and growing fast every day another million of users.

3.2 Android Environment.

Android is a mobile operating system based on the Linux kernel and currently developed by Google. With a interface based on direct manipulation, Android is designed primarily for touch screen mobile devices such as smartphones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear). The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touch screen input, it also has been used in game consoles, digital cameras, and other electronics.[9]

Android's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software. Currently available android versions are

- Alpha (1.0)
- Beta (1.1)
- Cupcake (1.5)
- Donut (1.6)
- Eclair (2.0–2.1)

- Froyo (2.2–2.2.3)
- Gingerbread (2.3–2.3.7)
- Honeycomb (3.0–3.2.6)
- Ice Cream Sandwich (4.0–4.0.4)
- Jelly Bean (4.1–4.3.1)
- KitKat (4.4–4.4.4)

Android's default user interface is based on direct manipulation using touch inputs, that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. The response to user input is designed to be immediate and provides a fluid touch interface, often using the vibration capabilities of the device to provide feedback to the users. Internal hardware such as accelerometers, gyroscopes and proximity sensors are used by some applications to respond to additional user actions, for example adjusting the screen from portrait to landscape depending on how the device is oriented, or allowing the user to steer a vehicle in a racing game by rotating the device, simulating control of a steering wheel.

Android has a growing selection of third party applications, which can be acquired by users either through an app store such as Google Play or the Amazon Appstore, or by downloading and installing the application's APK file from a third-party site.[61Google Play Store allows users to browse, download and update applications published by Google and third-party developers, and the Play Store client application is pre-installed on devices that comply with Google's compatibility requirements and license the Google Mobile Services software.

Applications (apps) that extend the functionality of devices, are developed primarily in the Java programming language using the Android software development kit (SDK). The SDK includes a comprehensive set of development tools, including a debugger, software libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) plugin. Other development tools are available, including a Native Development Kit for applications or extensions in C or C++, Google App Inventor, a visual environment for novice programmers, and various cross platform mobile web applications frameworks.

3.2 RDBMS vs NoSQL DB

A relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model .RDBMSs have become a predominant choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data, and much more since the 1980s. Relational databases have often replaced legacy hierarchical databases and network databases because they are easier to understand and use.

NoSQL or Not Only SQL database is next generation databases mostly addressing some of the points, being non-relational, distributed, open-source and horizontally scalable. It’s original intention has been modern web-scale databases**.** The movement began early 2009 and is growing rapidly. Currently there are around 150 NoSQL databases available.[10]

Below table 3.1 provide thecComparisson of RDBMS vs NoSQL DB

Table 3.1 Comparisson of RDBMS vs NoSQL DB

	RDBMS	**NoSQL**
Development History	Developed in 1970s to deal with first wave of data storage applications	Developed in 2000s to deal with limitations of SQL databases, particularly concerning scale, replication and unstructured data storage
Examples	MySQL, Oracle, db2,SQLite	MongoDB, CouchDB, Cassandra, HBase, Neo4j
Data Storage Model	Individual records are stored as rows in tables, with each column storing a specific piece of data about that, much like a spreadsheet. Separate data types are stored in separate	Varies based on database type. For example, key-value stores function similarly to SQL databases. Document databases do away with the table-and-row model altogether, storing all relevant data together in single

	tables, and then joined together when more complex queries are executed.	"document" in JSON, XML, or another format, which can nest values hierarchically.
Schemas	Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline.	Typically dynamic. Records can add new information on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically.
Scaling	Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required.	Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary
Development Model	Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database)	Open-source
Supports Transactions	Updates can be configured to complete entirely or not at all	In certain circumstances and at certain levels (e.g., document level vs. database level)
Data Manipulation	Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE…	Through object-oriented APIs

3.4 Embedded Android Databases

Android supportive RDBMS and NoSQL databases are given by table 3.2

RDBMS	**NoSQL**
SQLite	CouchDB
Interbase	Couchbase Lite

Table 3.2 Android supportive RDBMS and NoSQL databases

One of the popular choice embedded RDBMS in local and cloud is SQLite. SQLite is an Open Source database. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. The database requires limited memory at runtime hence it's really famous. It's defining the SQL statements for creating and updating the database. Afterwards the database is automatically managed in the operating system. Access to SQLite database involves accessing the file system

One of the famous NoSQL db is CouchDB It is an open Source database uses JSON (JavaScript Object Notation) to store data, JavaScript as its query language using Map Reduce, and HTTP for an API. Unlike in a relational database, CouchDB does not store data and relationships in tables. Instead, each database is a collection of independent documents. Each document maintains its own data and self-contained schema. An application may access multiple databases, such as one stored on a user's mobile phone and another on a server. Document metadata contains revision information, making it possible to merge any differences that may have occurred while the databases were disconnected.

3.5 Summary

With the introduction of Android operating system, mobile industry was widely increased. Android's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software.

Comparing RDBMS and NoSQL databases there are many advantages and disadvantages appear in both databases. Analyzing current available databases it can observe Relational databases management systems are ruling at the top. However recent huge popularity of NoSQL databases for cloud computing has inspired initiative non-relational data types the essence of big data flooding in organizations and large datacenters.

However there is very limited number of RDBMS and NoSQL databases support Android environment. They are SQLite,

Chapter 04

Approach

4.1 Introduction

Conduct the performance evaluation of mobile embedded database as initial step we may have to first derive the experiment plan. Since it is a comparison of two mobile embedded databases, mobile environment which is databases going to be install has a very big liability.

Experiment has to be design for keeping the mobile environment as constant parameter. It's really important to get the accurate database results.

4.2 Proposed Experiment

1. Install databases locally in chosen Android environment
   I. Open source db Apache CouchDB consider as NoSQL database
   II. Open source db SQLite consider as Relational database.

2. Create Android interface to each database which these two embedded databases that could be driven by benchmark test attach.

3. Benchmark Data set
   I. Create suitable schema for experiment.
   II. Using benchmarking tool YCSB create large datasets based on a created schema and fill them with random data.

4. Benchmark Queries
   I. Setup queries for both databases covering by all CRUD operation

5. Benchmark Metrics

I. Identify a set of metrics for the benchmarking system to record. It will represent the important factors with respect to an application performance on time base such as
   a) Queries per seconds
   b) Average Query execution time.
   c) Overall runtime.

6. Evaluate benchmark metrics for each test against in both databases.

   I. Evaluate the performance in a light load data set
   II. Evaluate the performance in a heavy load data set

4.3 Android testing environment

Table 4.1 provides the Android testing environment details which may databases install.

Table 4.1 Android test environment details

Equipment	Huawei MediaPad 7 Lite
OS	Android OS, v4.0.4 (Ice Cream Sandwich)
CPU	1.2 GHz Cortex-A8
Browser	HTML5, Adobe Flash
RAM	1 GB
Internal Memory	8 GB
Java	Java MIDP emulator
Data Speed	HSDPA, 3.6 Mbps; HSUPA
GPRS	Class 12 (4+1/3+2/2+3/1+4 slots), 32 - 48 kbps

4.4 Android Interface to interact with both databases.

Design android interface to which both SQLite and couchdb can be interact with. To design interface we may use following.

- JDK 1.7
- Eclipse 4.2 Juno
- Android SKD 4.2

4.5 Summary

To approach the benchmarking mobile embedded databases, we may first have to choose mobile environment which is databases going to install. It may have to constant for both databases.

Then create a android interface where it can interact with both installed databases.

Conducting the proposed experiment by installing databases and benchmarking databases.

Chapter 05

Analysis and Design

5.1 Introduction

Design the process of benchmarking mobile embedded databases is very crucial task coz it may interact with many layers. Design should be capable to get the outcome of the main experiment.

As initial step need to be design how the android application will may interact with two databases. Then have to design how two databases will be created and then the datasets. It should not be bias to any database and equivalent to both. Then need to benchmark queries and the matrices.

5.2 Android application database design

Figure 5.1 will illustrate the android application and database installation design

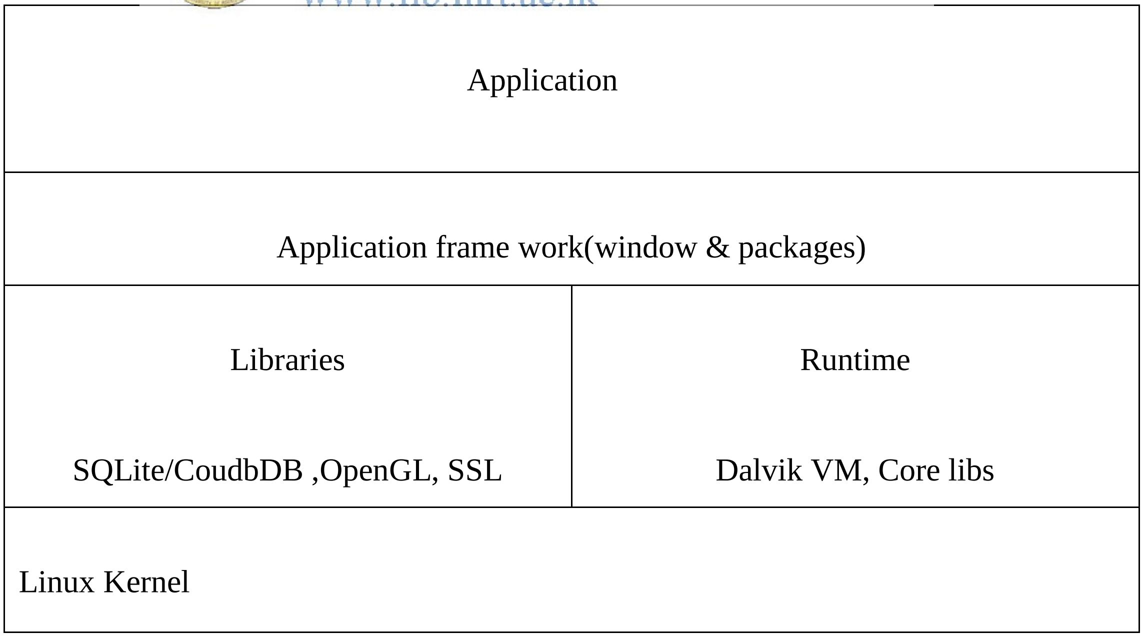
Application
Application frame work(window & packages)
Libraries
SQLite/CoudbDB ,OpenGL, SSL
Runtime
Dalvik VM, Core libs
Linux Kernel

Figure 5.1 Android application and database installation design

- Applications - The Android Open Source Project contains several default application, like the Browser, Camera, Gallery, Music, Phone and more.
- Application framework - An API which allows high-level interactions with the Android system from Android applications.
- Libraries and runtime - The libraries for many common functions (e.g.: graphic rendering, data storage, web browsing, etc.) of the Application Framework and the Dalvik runtime, as well as the core Java libraries for running Android applications.
- Linux kernel - Communication layer for the underlying hardware.

5.3 Benchmark Database

To standardized tests, benchmarking tools usually crated large database on predefined schema and fill them with random data. Generated data sets enable benchmark to increase the size and complexity of test while maintaining a consistent standard for comparison.

Designing ‘**University**’ database where two schema ‘**Student**’ and ‘**Course**’ .

Student schema has tables ‘StudentProfile’ and ‘StudentResults’ .

Course schema has ‘CourseProfile’ and ‘CourseTaken’ tables

Student ID will be foreign key from ‘StudentProfile’ table to ‘CourseTaken’ table.

Student. SutdentProfile (Student_ID, Name, Gender, Birthday, Age, Address, Major)

Course.CourseProfile (Course_ID, Course_Name ,Lecture_Name, prerequisite)

Course. CourseTaken (Student_ID, Course_ID, Year_taken, Results)

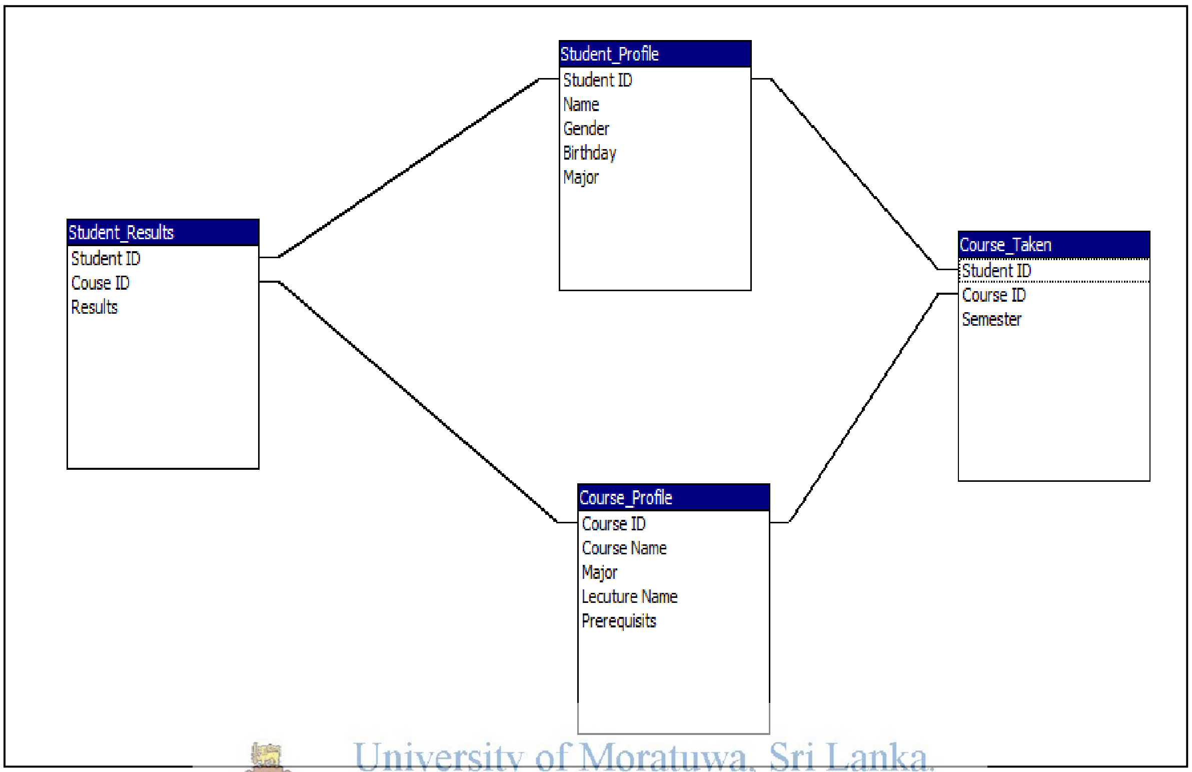

Figure 5.2 Relationship of University DB

5.4 Benchmark Data Set

When creating tables number of ‘CourseTaken’ will be always three times the ‘StudentProfile’

This schema will be populated by a FeildGenerator class that used an Xorshift random number generator and arrays of predefined data to generate reasonable tables as efficiency as possible. Using the FeildGenerator we could easily scale the size of the databases to compare how each database system performed on large datasets.

5.5 Bench mark Queries

The benchmark queries will be created test all CRUD operations of the databases and will be breakdown operations into the following benchmark sets.

Create Benchmark : This comprises of database initialization with defined tables/structures

Create Table SutdentProfile (Student_ID INTEGER primary key, Name TEXT, Gender TEXT , Birthday INTEGER, Age INTEGER, Address TEXT, Major TEXT)

Create Table CourseTaken (Student_ID INTEGER, Course_ID INTEGER , Year_taken INTEGER , Results INTEGER)

Note : SQLite supports the data types TEXT (similar to String in Java), INTEGER (similar to long in Java) and REAL (similar to double in Java). All other types must be converted into one of these fields before getting saved in the database. SQLite itself does not validate if the types written to the columns are actually of the defined type, e.g. can write an integer into a string column and vice versa.

Insert Benchmark: This comprises of all queries to load the data into the data management system and inserts an arbitrary number of tuples into the schema.

Insert into SutdentProfile (Student_ID, Name, Gender, Birthday, Age, Address, Major) Values (* ,* , * ,* ,* , *, *)

Insert into CourseTaken (Student_ID, Course_ID, Year_taken, Results)

Values (* ,* , * ,*)

Data Fetch: Queries which retrieve tuples based on specific value of indexed and non-indexed attributes.

Test 1: Select * from SutdentProfile where Student_ID=* (Indexed Attribute)

Test 2: Select * from SutdentProfile where Name=* (Non -Indexed Attribute)

Range Query: Queries which retrieves tuples within some range.

Test 3: Select * from SutdentProfile where Age >=* and Age <=*

Aggregation: Queries which return tuples with some computation like count average over a set of group based on some attributes.

Test 4: select age ,count(*) from SutdentProfile group by age

Test 5: select course_ID, count(student_ID) from CourseTaken group by course_ID

Join : Join query which fetch data by joining two tables on specific attributes

Test 6: select S.Student_ID, S.Age , C.Course_ID from SutdentProfile S , CourseTaken Cwhere S.student_ID= C.student_ID and S.age <=*

5.6 Benchmark Metrics

Identify a set of metrics for the benchmarking system to record. It will represent the important factors with respect to an application performance on time base such as

a) Queries per seconds: Average number of Queries per second will be evaluate for each category. This will determine the throughput for each DBMS.

b) Average Query execution time: Average time for execution an individual query of particular query category as defined above. Against each category, ten queries with different parameters will be executed.

c) Overall runtime: The total time for all benchmark to run for each DBMS.

5.7 Summary

As an initial step designed the resistant of android application. Then designed database structure along with schema and tables which may suitable for both SQLite and CouchDB.

Then schema will be populated by a FeildGenerator class that used an Xorshift random number generator and arrays of predefined data to generate reasonable tables as efficiency as possible. Using the FeildGenerator we could easily scale the size of the databases to compare how each database system performed on large datasets.

After data set is designed the benchmarking queries which may run throughout the experiment. During the experiment benchmarking matrices will be evaluated and results will be obtained for comparison.

Chapter 06

Implementation

6.1 Introduction

In implementation first have to setup the android development tool s and make android environment suitable for build the application.

Once the Android developer tool sets up, can install application which may interact with two databases.

Then have to setup SQLite and CouchDB individually.

6.2 Setting up Android Developer Tools

First step is download packaged Android Developer Tools. Google provides a packaged and configured Android development environment based on the Eclipse IDE called ‘Android Developer Tool’s. Then install Stand-alone ADT. Finally update an existing Eclipse IDE.

The Android SDK contains an Android device emulator. This emulator can be used to run an Android Virtual Device (AVD), which emulates a real Android phone. AVDs allow to test Android applications on different Android versions and configurations without access to the real hardware.

6.3 Setting up SQLlite

SQLite database more convenient to setup on Android environment because SQLite is embedded into every Android device. Using SQLite database in Android does not require a setup procedure or administration of the database. Only have to define the SQL statements for creating and updating the database. Afterwards the database is automatically managed for you by the Android platform. [11]

Android SDK comes preloaded with libraries to crate and interface with SQLite files. Access to SQLite database involves accessing the file system. This can be slow. Therefore it is recommended to perform database operations asynchronously.

To create and upgrade a database in Android application that will create a subclass of the SQLiteOpenHelper class. In the constructor subclass call the super() method of SQLiteOpenHelper, specifying the database name and the current database version.

The android.database package contains all necessary classes for working with databases. The android.database.sqlite package contains the SQLite specific classes.

SQLite database is the base class for working with a SQLite database in Android and provides methods to open, query, update and close the database.

6.4 Setting up Couchbase Lite

CouchDB is contacted through a REST (Representational state transfer) API and the main tool for viewing the database is a web application call Futon. Futon is to CouchDB what phpMyAdmin is to MySQL. There is an Android application called Mobile Futon which brings CouchDB to an Android environment but the only method of accessing is it through the REST API.

Since requiring all data access for CouchDB to go through the network stack is not the proposed research method and SQLite not required the network, have to consider constant environment for both databases.

Couch has introduced, Couchbase Server to IOS and Android called as Couchbase Lite [12]. This turned out to be perfect implementation of couch to test since like SQLite Couchbase Lite act as an embedded DBMS. The documentation for Couchbase Lite say it is analogous to CouchDB in the same way SQLite is analogues to mySQL [12]. By using Couchbase Lite would be more similar to comparing SQLite.

For setting up Couchbase Lite have downloaded setup files from official Couchbase site where Couchbase Lite to android [13] .After that it was easy to embed Couchbase Lite source files to Eclipse IDE.

6.5 Interface

Before initiate database performance testing I have developed simple android application that will outline the structure of the testing.

This application has been help for visualize the scenarios that will be testing.

Figure 6.1 is providing sample screenshots of the application that I have created ‘University Database’

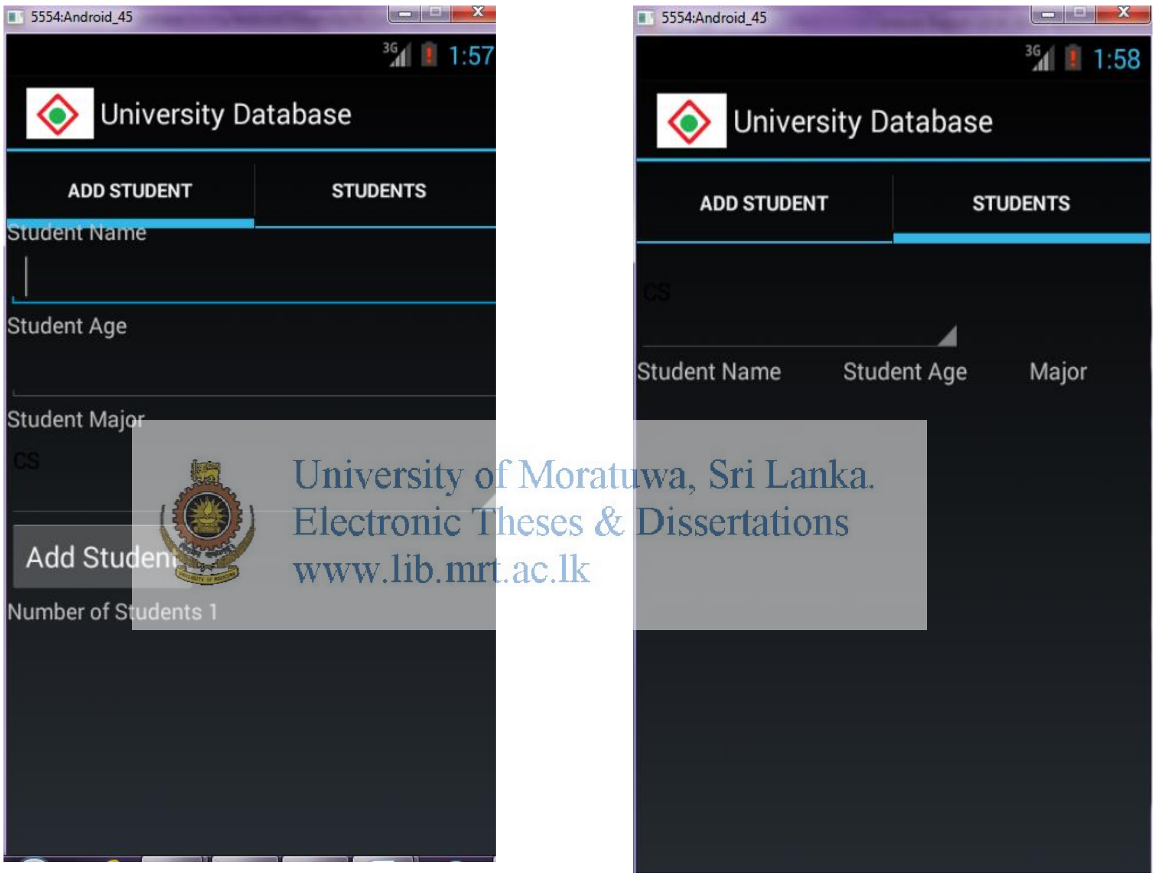

Figure 6:1 University Database App

While conducting database performance test, it’s not simply dealing with 10 or 20 tuples. It’s recommended to get a sample of more than 100 tuples for conducting database performance. Hence manually conducting CURD operation through application is not practical and recommended. Therefore all testing has been conducted by scripting on backend of the application.

6.6 Constructing SQLite Database on Android.

Constructing SQLite database was much easier than CouchdbLite because SQLite syntaxes were more similar to mySQL syntax and Android comes in with built in SQLite database implementation.

Figure 6.2 is providing sample codes for CREATE database and CREATE tables in SQLite .

```
public class DatabaseHelper extends SQLiteOpenHelper {

      static final String dbName="universityDB";
      static final String studentTable="Students";
      static final String colID="StudentID";
      static final String colName="StudentName";
      static final String colAge="Age";
      static final String colMajor="Major";

      static final String majorTable="Major";
      static final String colMajorID="MajorID";
      static final String colMajorName="MajorName";

      static final String viewStud="ViewStud";


      public DatabaseHelper(Context context) {
            super(context, dbName, null,33);

            // TODO Auto-generated constructor stub
      }

      @Override
      public void onCreate(SQLiteDatabase db) {
            // TODO Auto-generated method stub

            db.execSQL("CREATE TABLE "+majorTable+" ("+colMajorID+ "
INTEGER PRIMARY KEY , "+
                        colMajorName+ " TEXT)");

            db.execSQL("CREATE TABLE "+studentTable+" ("+colID+" INTEGER
PRIMARY KEY AUTOINCREMENT, "+
                        colName+" TEXT, "+colAge+" Integer, "+colMajor+"
INTEGER NOT NULL ,FOREIGN KEY ("+colMajor+") REFERENCES "+majorTable+"
("+colMajorID+"));");
```

Figure 6:2 SQLite DB creation and table creation codes

6.7 Constructing Couchbase Lite Codes on Android.

Comparing with SQLite it's difficult to constructing Couchbase Lite because it's using JSON structure to store data.Figure 6.2 is providing sample codes for CREATE database of Couchbase Lite

```
// create a name for the db and make sure the name is legal
       String dbname = "UniversityDB";
        if (!Manager.isValidDatabaseName(dbname)) {
            Log.e(TAG, "Bad database name");
            return;  }
        // create a new database
        Database database;
        try {
            database = manager.getDatabase(dbname);
            Log.d (TAG, "Database created");
        } catch (Couchbase LiteException e) {
            Log.e(TAG, "Cannot get database");
            return;  }
// add array of documents to Couchbase Lite
function couchbase_bulkAdd() {
  var max = 10000;
  var data = generate(max);
  console.time('Couchbase lite - bulk add');
  $.ajax({
    type: 'POST',
    url: globalUrl + 'performance/_bulk_docs',
    data: JSON.stringify({'docs': data}),
    contentType : 'application/json'})
  .done(function(res) {
    console.timeEnd('Couchbase lite - bulk add');
    dummyId = res[5000].id;
  });}
```

Figure 6:3 Creating CoudhbaseLite database

6.8 Bench Mark Data Set

Same unique data set that has been used for benchmark two databases performances. According to table structures, data has been populated.

'Student_Profile' table considered as main parent table. For main benchmarking queries ,100 tuples used as minimum, 10,000 tuples used as maximum and as average used 1000 tuples.

Figure 6:4 shows the sample data set of 'Student_Profile' table.

ID	Name	Gender	Age	Bday	Major
1011247I	Justus Oconnor	Male	20	1993/6/11	IT
1070633C	Edwin Hagins	Male	21	1992/7/26	IS
1089701C	Cecil Tinajero	Male	21	1992/6/28	CE
1100406C	Rahel Schneck	Female	21	1992/4/7	IT
1282273I	Thera Sine	Female	22	1991/3/2	IT
1312154C	Nabendu Janis	Male	22	1991/2/12	CS
1346548C	Dollie Bayes	Female	22	1991/8/16	IS
1450667I	Etta Overbey	Female	21	1991/11/3	IS
1480473I	Yervant Deppe	Male	21	1992/4/16	IT
1490789I	Vitale Schafer	Male	20	1993/11/6	IT
1530924C	Sorano Mohn	Female	21	1991/6/5	IS
1533865C	Onawa McGee	Female	20	1993/9/28	IT
1542434I	Ekachakra Bourdeau	Male	20	1993/5/23	CE
1557059C	Volumnia Riddick	Female	22	1992/12/6	IS
1558675C	Nerhim Oberlin	Male	22	1991/11/23	CS

Figure 6:4 Sample data set of 'Student Profile' table

Figure 6:5 shows sample data set of 'Course_Profile' table

Major	Course ID	Cours Name	Lecture Name	Pre requisits
CE	CS3310	Artificial Intelligence	Curtis Fancher	
CS	CS2110	Calculus & Statistical Distributions	Rebekah Vance	CS1110
IT	IT1320	Computer Organization	Cadman Pasquale	
CE	CE2110	Data Structures and Algorithms	Rebekah Vance	IT1400
IT	IT1310	Digital Systems and Computer Hardware	Zoe Diego	IT1320
IT	IT1400	Fundamentals of Databases and Database Design	Suffield Paramore	
CS	CS1110	Fundamentals of Mathematics & Statistics	Suffield Paramore	
CS	CS3320	Logic Programming & Artificial Cognitive Systems	Hanford Templeton	CS3310
IT	IT1610	Multimedia Technologies	Francesca Siefert	
CE	CE2100	Object Oriented Programming	Curtis Fancher	IT1100
CE	CE2210	Operating Systems	Curtis Fancher	IT1320

Figure 6:5 Sample data set of ‘Course_Profile’ table

6.9 Testing Specification

A full trial for each benchmark consists of following

1. Creating the database files.
2. Setting up necessary schemas
3. Running each of the six queries three times and average time is displayed in results.
4. Recorded the time it took for each part of the benchmark complete.
5. Test the scalability of the two databases, ran all benchmark three times and increased the number of tuples on each run.

6.10 Summary

Initially set up Android Developer Tools (ADT) which includes eclipse IDE, Android SDK and Android virtual devised (AVD) which helps to develop android in virtual environment.

On top of Android SDK , Android SQLite was easily setup. Couchbase Lite was downloaded from official couchbase site and source files were embedded in Eclipse IDE. It was bit difficult rather than SQLite.

Simple android application created that will outline the structure of the database testing. According to application SQLite and Couchbase Lite databases set up on android environment. That was a time consuming task because for SQLite its Java and for Couchbase Lite it was JSON. After setting up databases generated benchmark data set that related for each tables. Finally created generic test specification steps that where two database fallow for conducted the performance test.

Chapter 07

Evaluation

7.1 Introduction

After implemented two databases on android, using predefined dataset I have conducted the experiments according to the test specification above given on section 6:9. Below presenting series of charts and tables showing experiment results.

7.2 Average DB Creation Time

Figure 7:1 showing the average time to create all files associated with the embedded databases.

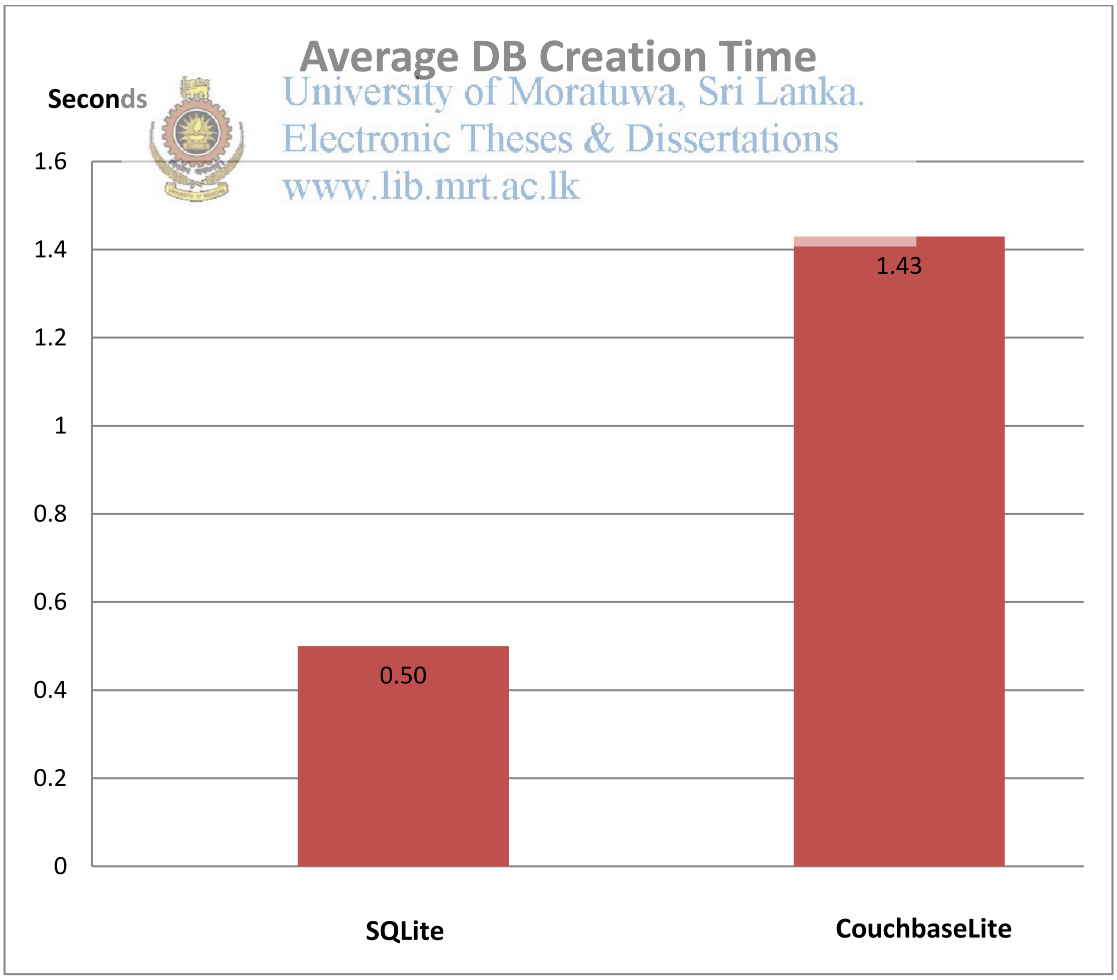

Figure 7:1 Average DB creation time in seconds

According to the above figures Couchbase Kite was nearly three times slower than SQLite. However at one second this is not a considerable overhead considering each database is only created once as initial creation.

7.3 Insertion to DB and average insertion per second.

In this test scenario I have segregated to three instant and insert into main table in db as 100, 1000 and 10,000 tuples. Figure 7:2 shows average insertion for table while increasing number of tuples.

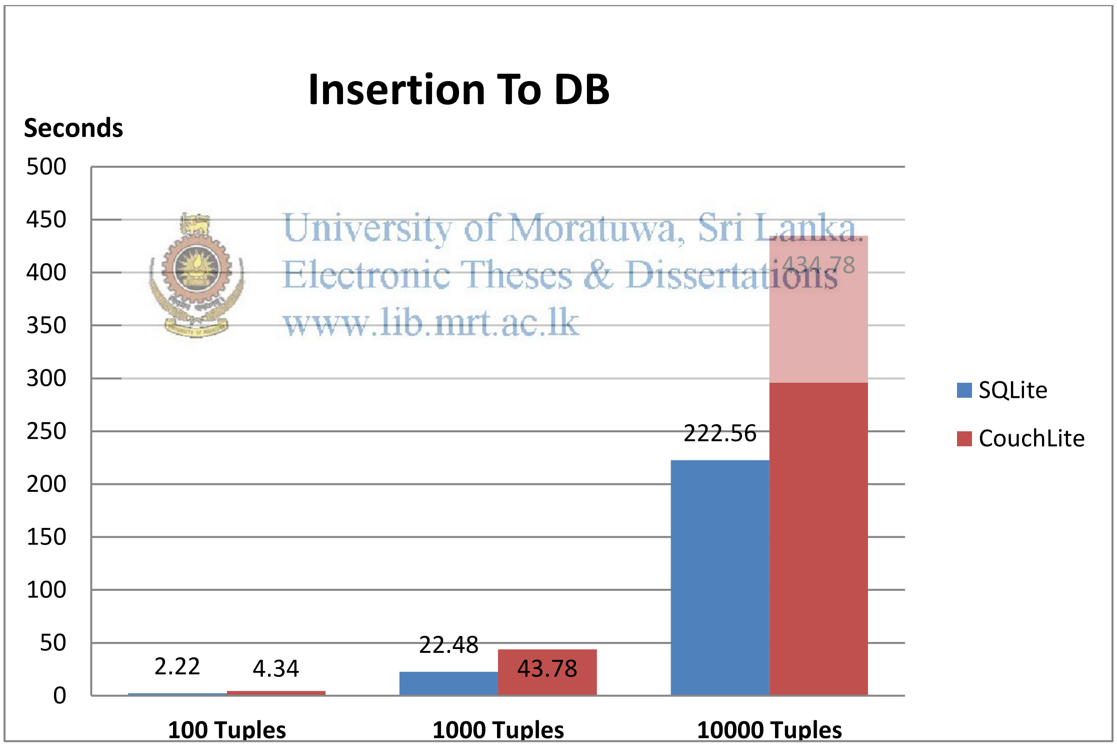

Figure 7:2 Data insertion to databases

According to the test results it clearly shows while increasing number of tuples the insertion time is increasing according to same. However SQLite is much faster than Couchbase Lite and Couchbase Lite is two times slower than SQLite while inserting data.

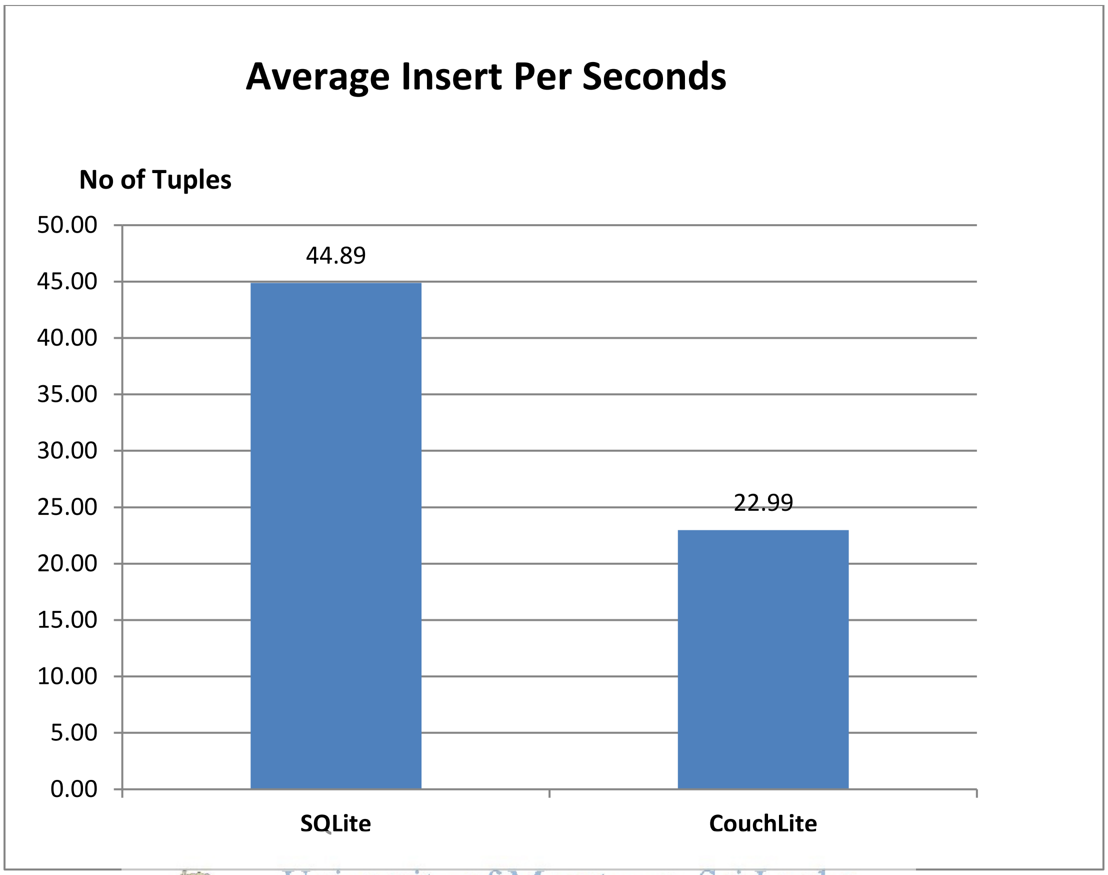

Figure 7:3 Average insertions per seconds for databases.

Figure 7:3 shows the average number of insert per second for each database. There is no significant difference in insert speed as the number of inserts increase from hundred to ten thousand. In a second SQLite inserts nearly 45 tuples and CouchLite inserts nearly 23tuples. On average SQLite can insert twice as may tuples per seconds as CouchLite. This is a significant difference and in ten thousand inserts SQLite took 3.7 minutes while CouchLite took 7.2 minutes.

7.4 Data fetch and average data fetch per second with an index

Figure 7: 4 shows the time spent executing Test 1 (Select * from Sutdent_Profile where Student_ID=* (Indexed Attribute)) which is a simple indexed table lookup.

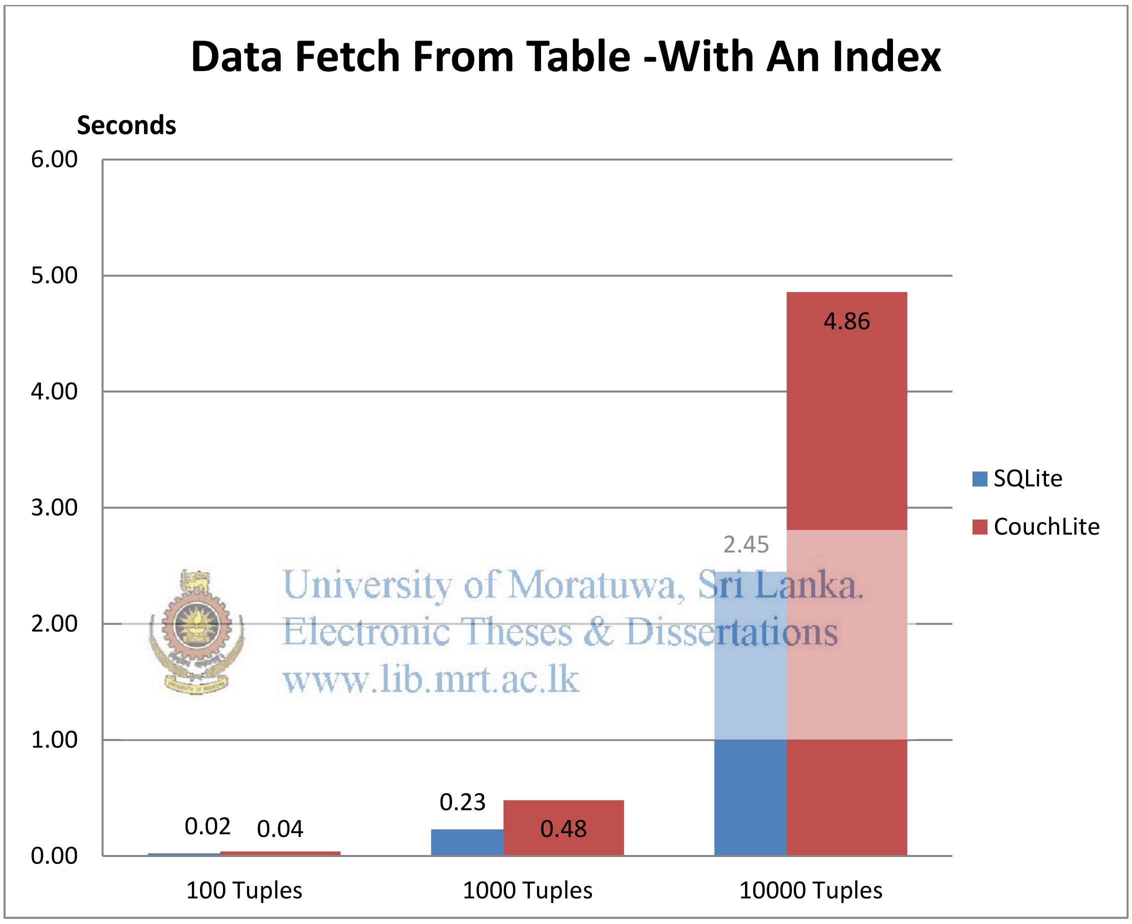

Figure 7:4 Data fetch from table with an index.

Not surprisingly there is little correlation between the number of tuples being searched and the time it takes for an indexed lookup since two databases are highly optimized for indexed searches. However SQLite consistently performed three to five times faster than CouchLite.

Figure 7:5 shows the average data fetch per second while index using.

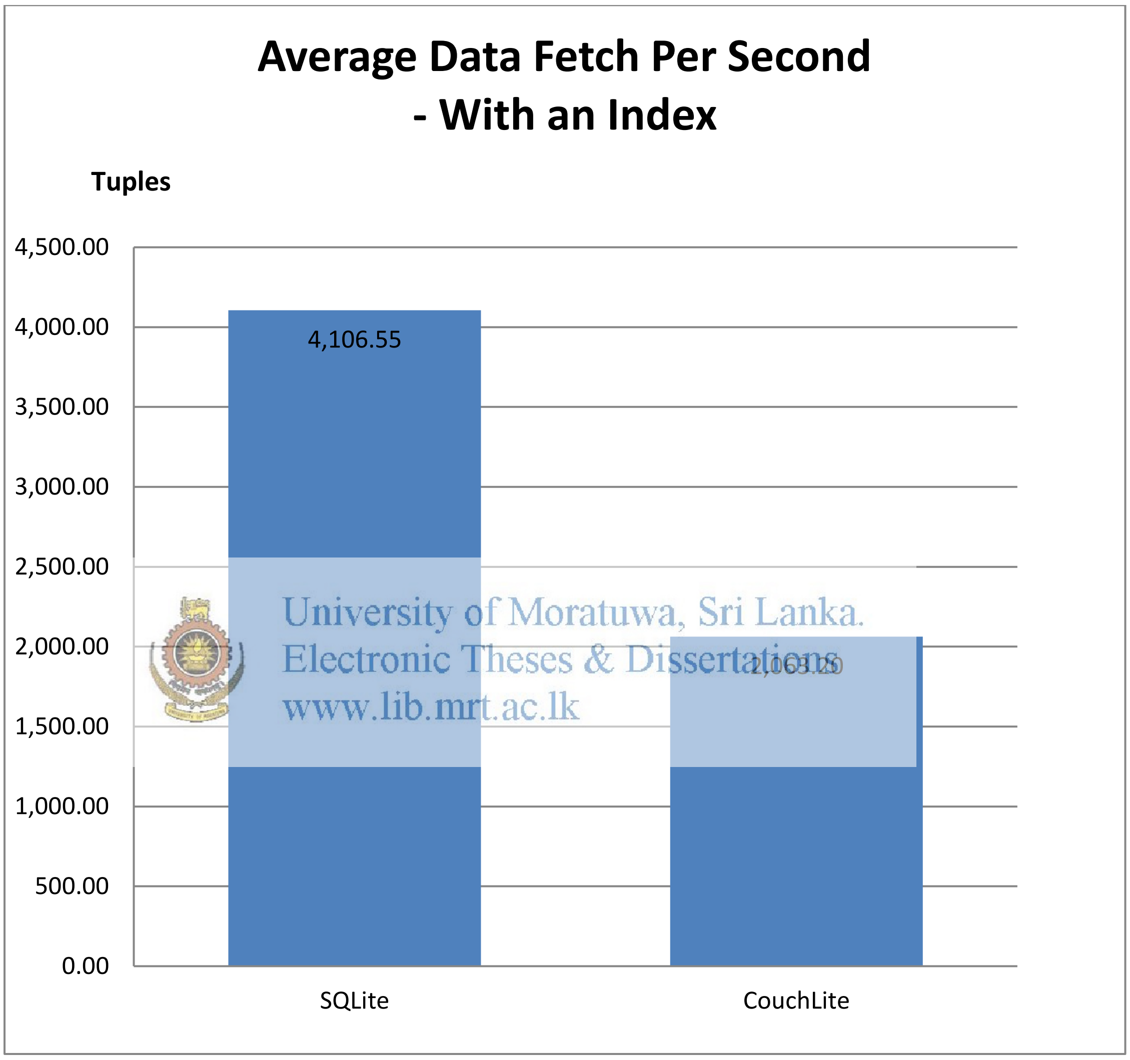

Figure 7:5 Average data fetch per second - with an index.

Looking at experiment results, it was so gleaming that SQLite fetching data super fast rather than Couchlite while index using. 4,106 tuples fetch per second is magnificent performance in SQLite.

7.5 Data Fetch and Average Data Fetch Per Second with Non Index Lookup

Figure 7.6 shows the time spent executing Test 02 (Select * from Sutdent_Profile where Name=* (Non -Indexed Attribute)) where data fetch from with non indexed lookup on a string field.

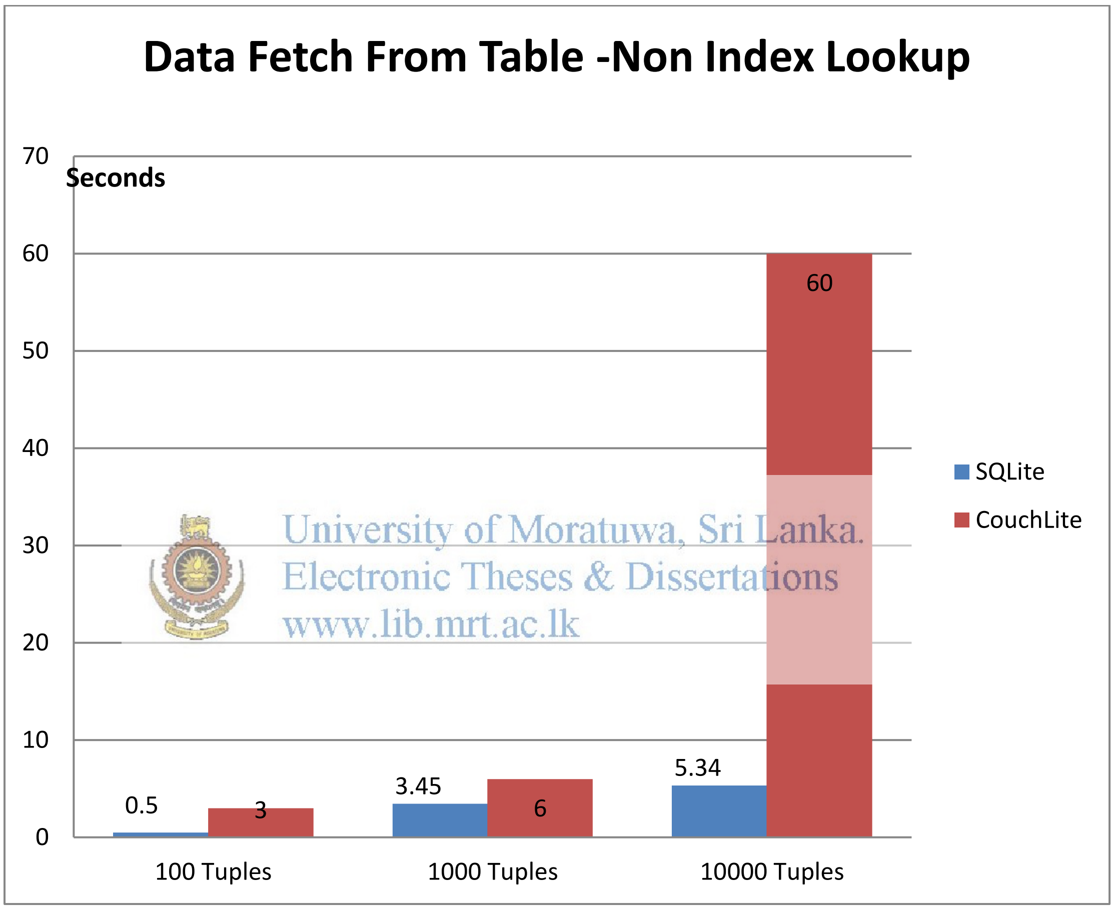

Figure 7:6 Data fetch from table –Non index lookup

Unlike data fetch from indexed lookup , this test scenario time an increase in execution time as the number of tuples increases is easily observed.

For CouchLite this increase is clearly exponential, while the increase for SQLite is not as dramatic.

Figure 7.7 shows the average data fetch per second when non index lookup.

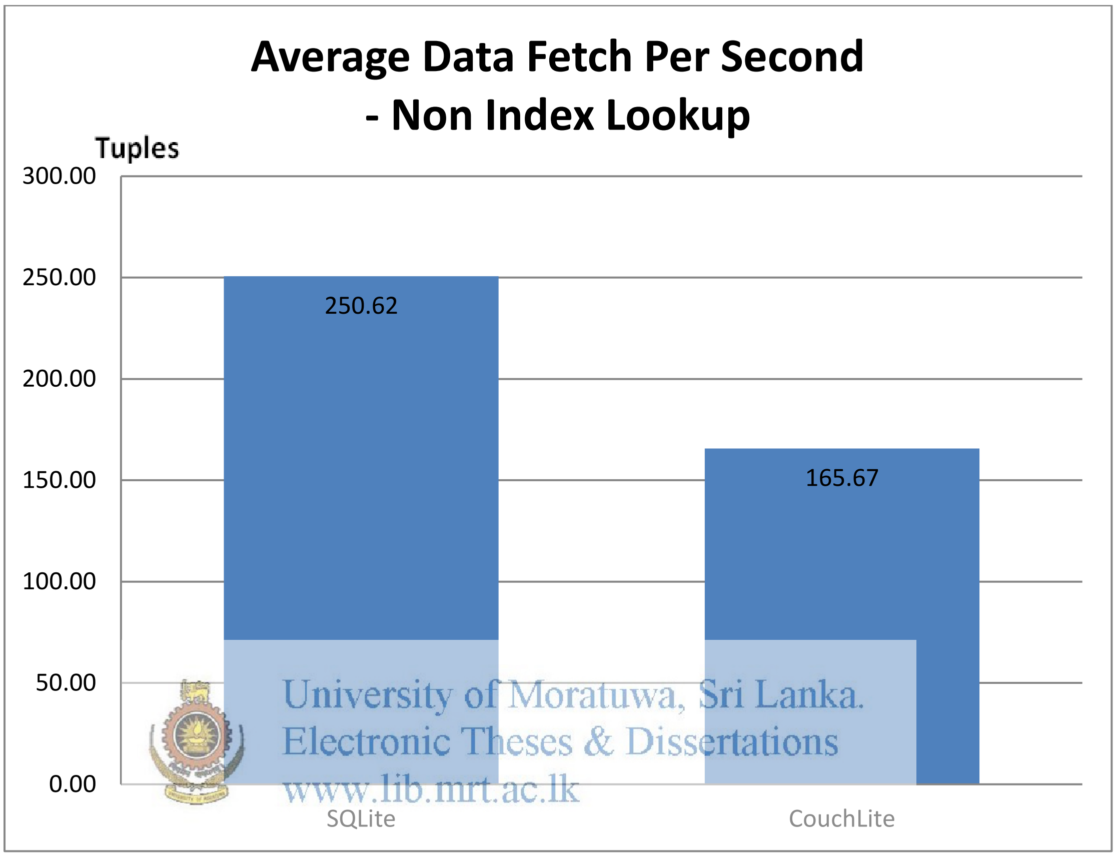

Figure 7:7 Average data fetch per second –Non index lookup

According to test results average data fetch per second while non-index lookup in SQLite was nearly 251 tuples and Couchlite was 166. When comparing with data fetching with indexed lookup, both databases are absolutely very slow.

7.6 Data Fetch and Average Data Fetch Per Second When Range Selected.

Figure 7:8 shows the test 03 (Select * from SutdentProfile where Age >=* and Age <=*) where data fetching speed while range defining .

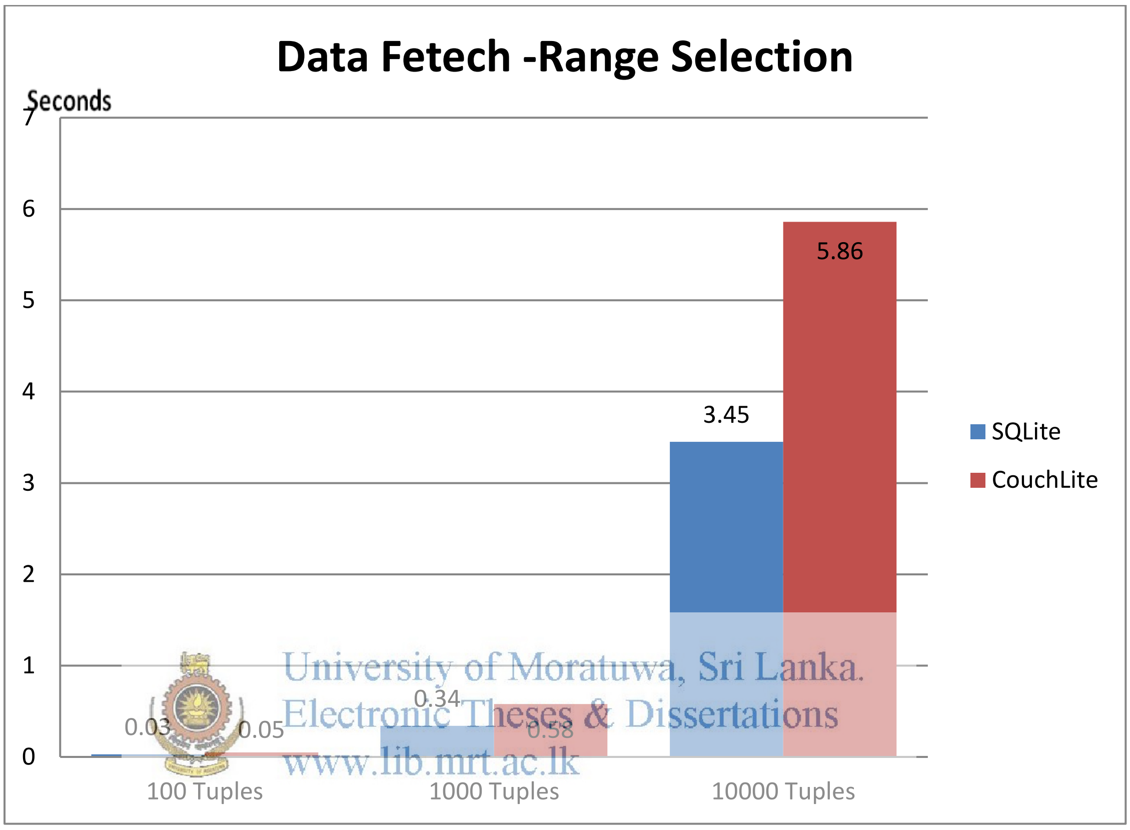

Figure 7:8 Data fetch –Range Selection

There is little correlation between the number of tuples being searched and the time it takes for data fetch while selecting range. Both databases are highly optimized for data fetch in selected range while indexed using and other hand when non –indexed look up both databases not vastly optimized. However comparing other test results in range selection both databases spend substantial amount of time.

Figure 7:9 illustrate the average data fetch per seconds when range selection provided.

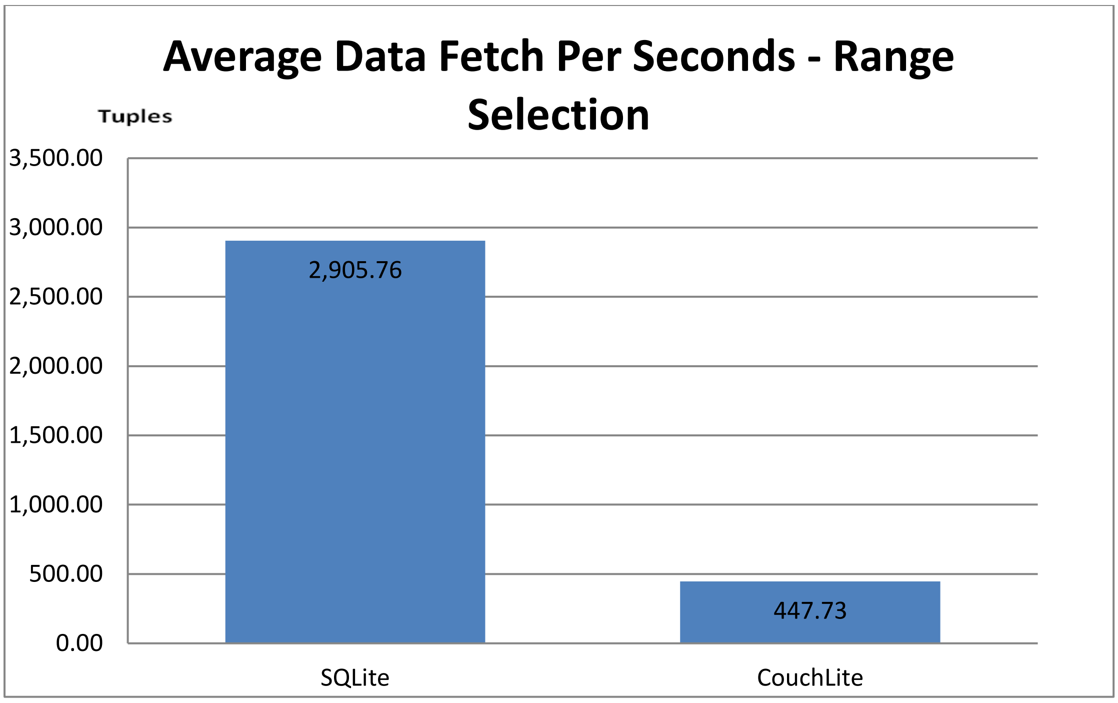

Figure 7:9 Average Data Fetch Per Seconds – Range Selection

According to the test results average data fetch per second is higher in SQLite rather comparing with CouchLite.

7.7 Inner join with index.

Figure 7:10 illustrate the consumed time when fetch data by joining two tables on specific attributes

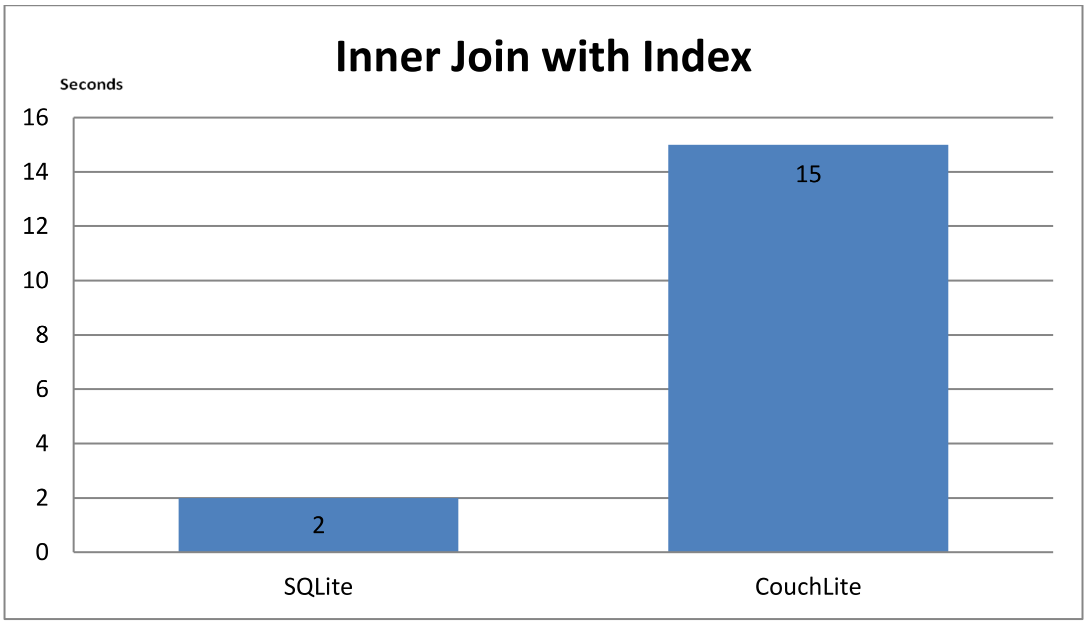

Figure 7:10 Inner Join with Index

According to test results average amount for fetch data by joining two tables on specific attributes in SQLite was two seconds and for same in Couchlite it was 15 seconds. When comparing two databases Couchbase Lite was 7 times slower than SQLite when fetching data by joining two tables.

7.8 Aggregate Sum

Figure 7:11 illustrate the computation average time over a set of group based on some attributes.

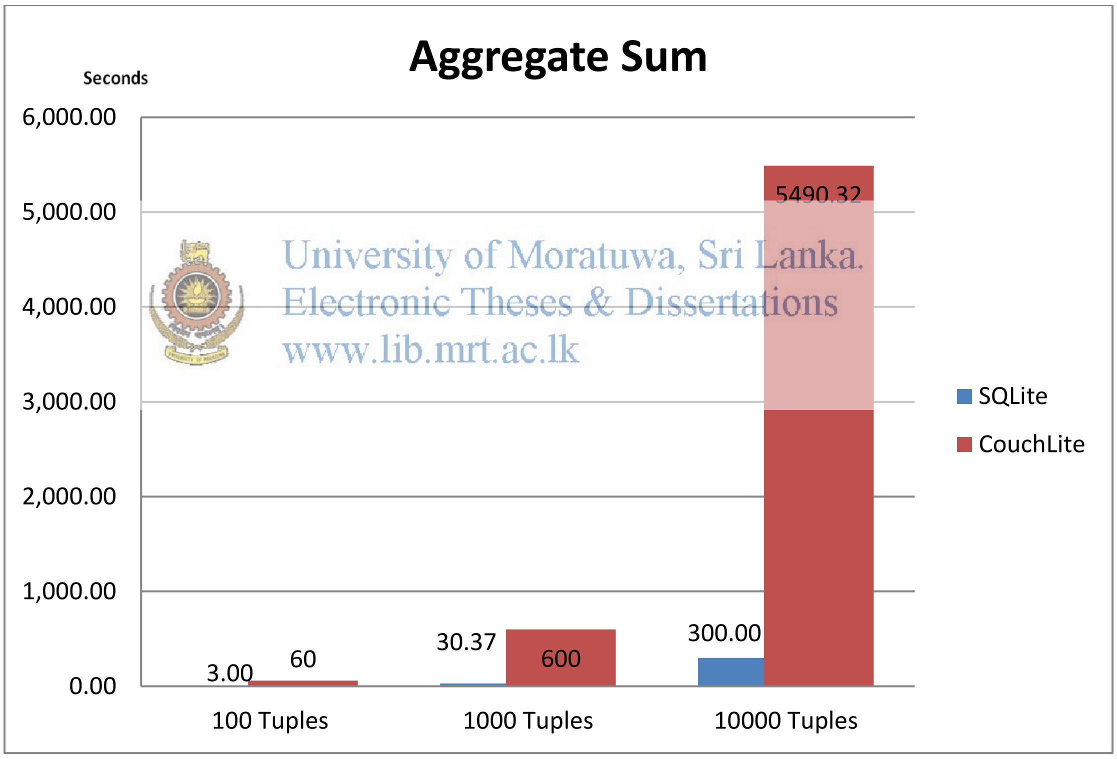

Figure 7:11 Aggregate Sum

Figure 7:11 shows the time spent executing test 5 which is aggregate sum. At ten thousand touples it took three hundred seconds which mean s five minutes and CouchLite took 5490 seconds which means almost 91 minutes .This was the highest amount of consuming time spent in entire test. This query groups by a non-indexed field and sum over a floating point values. CouchLite is not optimized for this type of query.

Calculate overall time consumed for each database for stranded test cases and 100 tuples shows in Table 8:1. All time calculation are in unit seconds

Category	SQLite	Couchbase Lite
Average DB creation time	0.05	1.43
1000 tuples insert to DB	22.48	43.78
1000 tuples fetch from db-with index lookup	0.23	0.48
1000 tuples fetch from db-with non index lookup	3.45	6
1000 tuples fetch – range selection	0.34	0.58
1000 tuples aggregate sum	3.37	600
Inner join with index	2	15
TOTAL	**31.92**	**667.27**

Table 8:1 Overall time consuming in both DBs in performance test

Chapter 08

Conclusion and Future Work

8.1 Introduction

Previous chapter (Chapter seven) demonstrates on the evaluation of experiment results on SQLite and CouchLite performance in same test scenarios. Main objective of this research was out current available embedded databases identifying most suitable embedded database for mobile enviornment. This chapter concludes what embedded database is more suitable for mobile environment.

8:2 Conclusion

By conducting performance evaluation of SQLite and CouchLite databases would excel at some parts of each benchmark and do poorly on others and that based on the results that can recommended a different embedded databases for different tasks.

According to the test results it clearly shows while increasing number of tuples the insertion time is increasing according to same. However SQLite is much faster than Couchbase Lite and Couchbase Lite is two times slower than SQLite while inserting data.

There is little correlation between the number of tuples being searched and the time it takes for an indexed lookup since two databases are highly optimized for indexed searches. However SQLite consistently performed three to five times faster than CouchLite.

Comparing other test cases both databases took substantial amount of time to fetch data from database when non-indexed lookup.

Correlation noted between the number of tuples being searched and the time it takes for data fetch while selecting range. Both databases are highly optimized for data fetch in selected range while indexed using and other hand when non –indexed look up both databases not vastly optimized.

According to test results average amount for fetch data by joining two tables on specific attributes in SQLite was faster than Couchbase Lite. When comparing two databases Couchbase Lite was 7 times slower than SQLite when fetching data by joining two tables.

In entire test cases the highest amount of consuming time spent test case is aggregate sum. This query groups by a non-indexed field and sum over a floating point values. CouchLite is not optimized for this type of query.

SQLite performed very well under high load and the most complicated queries that included in benchmarks.

CouchLite did perform adequately on indexed lookup and queries requiring only simple map function, however, it did not scale very well. This makes sense since one of the primary use cases for NoSQL database like CouchLite is in embarrassingly parallel environments where map/reduce views can be executed and cashed for quick key based lookup. On a mobile device the views have to be calculated at runtime.

Based on the results of my research SQLite is the best embedded database for mobile.

Therefore it can illustrate RDBMS is better than NoSQL databases on mobile

Environment.

Due to popularity of NoSQL databases for cloud computing has inspired initiatives to conduct this research to verify is NoSQL is suitable for mobile devices. However final conclusion is NoSQL databases are not suitable for mobile environment.

8:3 Future Works.

In this research I have obtained only one database from RDBMS and one database from NoSQL. In future it’s better to compare performance elevation for RDBMS and NoSQL by obtaining more databases from each category.

Also this research can be expand by inserting more tuples in to databases and see the performance in very high loaded tuples.

8:4 Summary

By evaluating each test results presented in chapter 07, I have concluded which type of embedded database is more suitable for mobile environment. According to my research experiment out of SQLite and Couchbase Lite , SQLite performed well in mobile environment. Therefore I concluded beast database type for mobile embedded database is RDBMS.

References

[1] Noris, A .K (2007) ,*Mobile and embedded databases,* In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp 1175-1177, New York, USA .

[2] DATASTAX Cooperation (2013), *Benchmarking Top NoSQL Databases - A Performance Comparison for Architects and IT Managers,* White Paper,California,USA.

[3]TPC –Transaction Processing Performance Council

http://db-engines.com/en/ranking

[4] Cattell, R. (2011) *Scalable SQL and NoSQL Data Stores,* ACM SIGMOD Record, v 39.4, pp 12-27 ,Chicago, USA.

[5] Difallah D. E. , Pavlo A. ,Curino C. , Mauroux P. C.(2013) ,*An Extensible Testbed for Benchmarking Relational Databases,* At 40th International Conference on Very Large Data Bases 2014, In Proceeding of the VLDB Endowment, v 7.4 , pp 277 -288, Hangzhou, China.

[6] http://www.tpc.org/information/benchmarks.asp

[7] McObject Benchmarks Embedded Databases on Android Smartphone

http://www.mcobject.com/march9/2009

[8]Cooper B. F., Silberstein A. , Tam E., Ramakrishnan R., Sears R.(2010), *Benchmarking Cloud Serving Systems with YCSB,* Proceeding of 1st ACM Symposium on cloud computing , pp. 143-154,Indiana ,USA.

[9]Satyanarayanan M. (1996),*Fundemental challenges in mobile computing,* Proceeding of the 15th annual ACM symposium on Principles of distributed computing, pp1-7,Pennsylvania,United States.

[10] Turbyfill, C. , Orji C.U. ,Bitton D. (1993), *ASAP-an ANSI SQL standard saleable and portable benchmark for relational database systems*,in J.Gray(Ed.) , The Benchmark Handbook, second ed. , California,USA.

[11] http://www.android.com/

[12]http://nosql-database.org/

[13] http://www.sqlite.org/

[14] http://www.couchbase.com/nosql-databases/couchbase-mobile

[15] http://www.couchbase.com/nosql-databases/downloads#cb-mobile