NoSQL in a Mobile World: Benchmarking Embedded Mobile Databases

Trevor Perrier Fahad Pervaiz

tperrier@cs fahadp@cs

Abstract

In this paper, we evaluate three types of embedded data storage models on mobile devices. SQLite is a relational database, CouchLite is a key-value NoSQL database and DbO4 is an object-oriented databases. Mobile phones are becoming powerful enough to do a large amount of processing without leveraging cloud services. Most mobile applications are data driven and their performance depends on data availability. We developed a comprehensive set of benchmarks to test each database in an Android application. The results from our tests clearly show that the relational model has superior performance on mobile phones; however, there may be situations where only key-based lookups are necessary that a NoSQL solution would suffice.

Background

The popularity of NoSQL databases for cloud computing has inspired initiatives to install NoSQL databases on mobile devices. There are several reasons why such a solution is attractive: including replication, cloud synchronization, and the document data model. However, most NoSQL solutions are designed for large data centers with access to substantial computational power and network infrastructure. Little work has been done testing how a NoSQL databases preforms on the limited hardware of a mobile devices.

The motivation for this project comes from the design of the next version Open Data Kit, an Android based tool set for mobile data collection and processing. We have been migrating the tools over from native Android code to HTML5 and Javascript. One discussion that arose during this switch was migrating from SQLite to CouchDB. We were unable to find performance metrics for any NoSQL database on mobile devices.

This work is of particular importance to the research in our lab because we deploy technology solutions in developing countries where infrastructure support is never a guarantee. This means that an always on network connection cannot be assumed and a local copy of the data should be present on the phone. At the same time it is also important to maximize battery life because the interval between phone charging could be large. Reduced battery life on modern smart phones is already a big issue in low resource environments and requiring the phone to do even more by running a NoSQL or SQL database could reduce it more. We want to make sure we pick the technology that the best performance.

Related Work

There are several standard benchmarks out there. OLTP and OLAP style benchmarks [2, 3, 4] are for large data-center level systems rather than mobile environment. TPC-C and TPC-E are benchmarks for SQL based enterprise applications [5]. YCSB is a benchmark, made by Yahoo, for evaluation of NoSQL based cloud serving systems [6].

In general, there are no standard benchmarks for mobile based datastores. There is some recent work in proposing evaluation of SQL based embedded databases on mobile phones [7, 8]. However, there is no work which compares locally installed NoSQL database like CouchDB with SQL databases like SQLite.

Our Approach

Benchmark Data Set

In order to standardized tests, benchmarking tools like YCSB create large datasets based on a predefined schema and fill them with random data. Generated data sets enable the benchmark to increase the size and complexity of tests while maintaining a consistent standard for comparison.

We developed the following simple schema where 'seller' and 'buyer' in transaction is a foreign key 'id' from the people relation. You can consider this as a subset of ebay's data model which is presenting data on who sold what to whom.

```
PEOPLE (id, name, age, gender, color)
```

```
TRANSACTION (id, seller, buyer, price, item, date)
```

This schema was populated by a FieldGenerator class that used an XorShift random number generator and arrays of predefined data to generate reasonable tables as efficiently as possible. Using the FieldGenerator we could easily scale the size of the database to compare how each database system performed on large datasets. When creating tables the number of transactions was always three times the number of people.

Benchmark Queries

The benchmark queries we created test all CRUD operations of the database and we have broken these operations down into the following benchmark test. Each benchmark is described using SQLite, however, the exact implementation varies from database to database; for example in a document based storage engine there is no schema so the structure is maintained while inserting documents.

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

```
CREATE TABLE people (id integer primary key, name text, age
integer,  gender text, color text)
```

```
CREATE TABLE transactions (id integer primary key, seller integer,
buyer integer, price real, item text, date text)
```

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 people (id,name,age,gender,color) VALUES (?,?,?,?,?)
```

```
INSERT INTO transactions (id,seller,buyer,price,item,date) VALUES
(?,?,?,?,?,?)
```

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

Test 1: `SELECT * FROM people WHERE id=? (Indexed Attribute)`

Test 2: `SELECT * FROM people WHERE name=? (Non-indexed Attribute)`

Range Query: Queries which retrieve tuples within some range. For example, return all health centers which are in a particular district.

Test 3: `SELECT * FROM people WHERE ?<age and age<?`

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

Test 4: `SELECT age, count(*) FROM people GROUP BY age`

Test 5: `SELECT buyer, sum(price) FROM transactions GROUP BY buyer`

Join: Join query which fetch data by joining two table on a specific attribute.

Test 6:
```
SELECT t.id,p.name,p.age FROM transactions t, people p WHERE
p.id=t.buyer and ?<=p.age
```

SQL, NoSQL and Object-Oriented databases have very different approaches to handling each type of operation than SQL. Therefore, we added a layer in the benchmarking system to enable the support of each test mentioned above. Db4oLite does not have a "Group By" operation and so this was implemented in high level Java code using a hash table and calculating the aggregation functions. In CouchDB any query not on the indexed key of the database must be processed through a map/reduce view. In CouchLite these views are created in Java and executed at runtime.

Benchmark Metrics

We identified a set of metrics for the benchmarking system to record. These metrics are listed below. They represent the important factors with respect to an application performance. These metrics were evaluated for each test against all selected databases.

Average Query Execution Time: Average time for executing an individual query of a particular query category, as defined above. Against each category, ten queries with different parameters will be executed.

Queries per Second: Average number of queries per second will be evaluated for each category. This will determine the throughput for each DBMS.

Overall Runtime: The total time for all benchmark to run for each DBMS.

Embedded Android Databases

The first step for getting these benchmarks working was setting up each database in Android. SQLite [9] is very easy to set up since the Android SDK comes preloaded with libraries to create and interface with SQLite files.

CouchDB is more difficult to set up. Since both of us were unfamiliar with NoSQL databases we started by installing CouchDB on a Linux laptop. CouchDB is accessed through a REST API and the main tool for viewing the database is a web application called Futon. Futon is to CouchDB what phpMyAdmin is to MySQL. There is an Android application called MobileFuton which brings CouchDB to an Android phone but the only method of accessing it is through the REST API. We felt that requiring all data access for CouchDB to go through the network stack would bias our benchmarks towards SQLite and so we looked for a different solution.

We found a very recent port of Couchbase Server to iOS and Android called CouchbaseLite [10]. This turned out to be the perfect implementation of couch to test with since like SQLite CouchbaseLite acts as an embedded DBMS. The documentation for CouchbaseLite say it is analogous to CouchDB in the same way SQLite is analogous to mySQL. By using CouchbaseLite we would be comparing more closely to SQLite.

Setting up Db4o [11] was relatively easy comparing to CouchDB. Since Db4o is written in java and .NET, therefore it was easy to integrate it in android. Since we wanted it to be an embedded database, so we worked on ways make that possible. In that struggle, we found out Db4oLite that is java library. This library was then ported into android environment.

We implemented an interface in each of these three embedded databases that could be driven by a benchmark test harness.

Results

A full trial for each benchmark consists of creating the database files, setting up any necessary schema, and then running each of the six queries ten times. We recorded the time it took for each part of the benchmark to complete. To test the scalability of the three databases we ran all benchmarks seven times and increased the number of tuples inserted on each run. Below is a series of charts and tables showing our data.

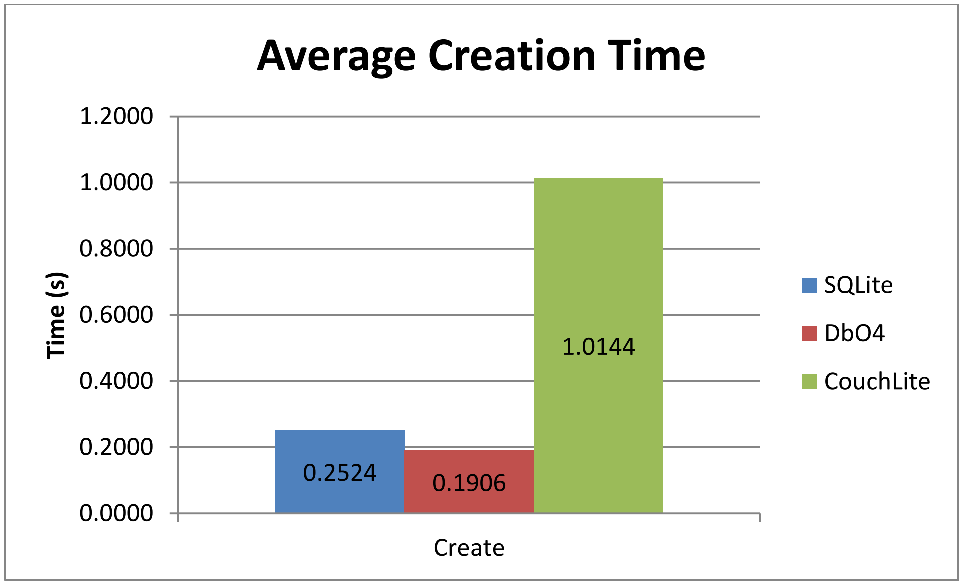

Figure 1: The average time to create all files associated with the embedded database. CouchLite was five times slower than SQLite and Db4O, however, at one second this is not a considerable overhead considering each database is only created once.

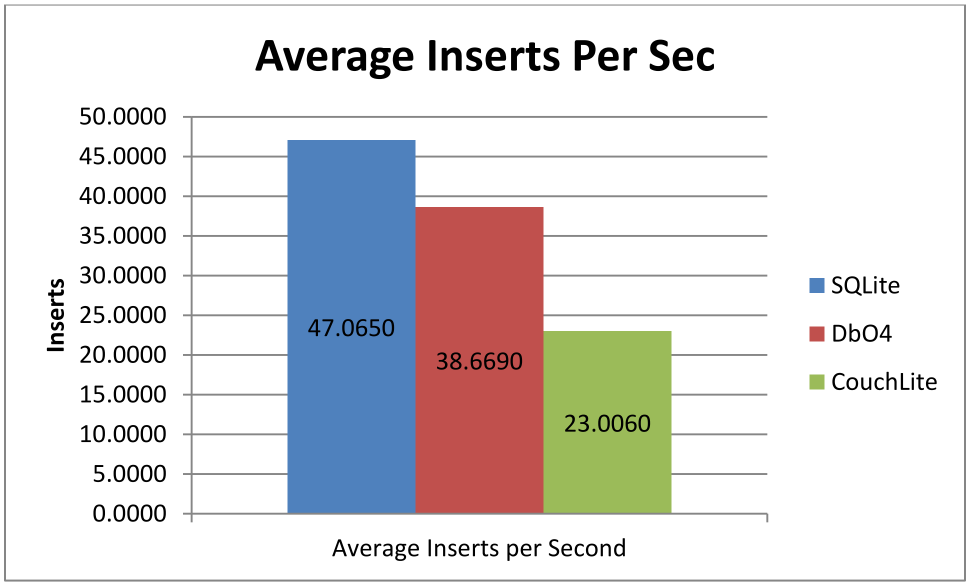

Figure 2: The average number of inserts per second for each system. There is no significant difference in insert speed as the number of inserts increase from ten to one thousand. On average SQLite can insert twice as many tuples per second as CouchLite. This is a significant difference and in our largest insert test SQLite took 5.8 minutes while CouchLite took 16.9 min.

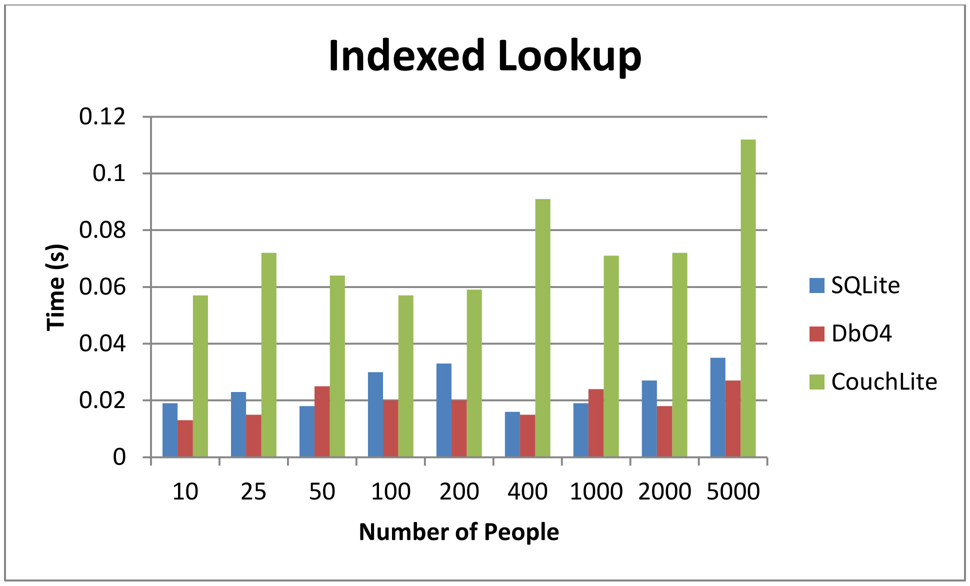

Figure 3: The time spent executing Test 1, which is a simple indexed table lookup. Not surprisingly there is little correlation between the number of tuples being searched and the time it takes for an indexed lookup since all three databases are highly optimized for indexed searches. However, SQLite consistently performed three to five times faster than CouchLite and DbO4 was often slightly faster than SQLite and at 5000 tuples DbO4 takes 65.27 times as long as SQLite while CouchLite takes 257.47 times as long. A similar pattern is observed for Test 3 and Test 6.

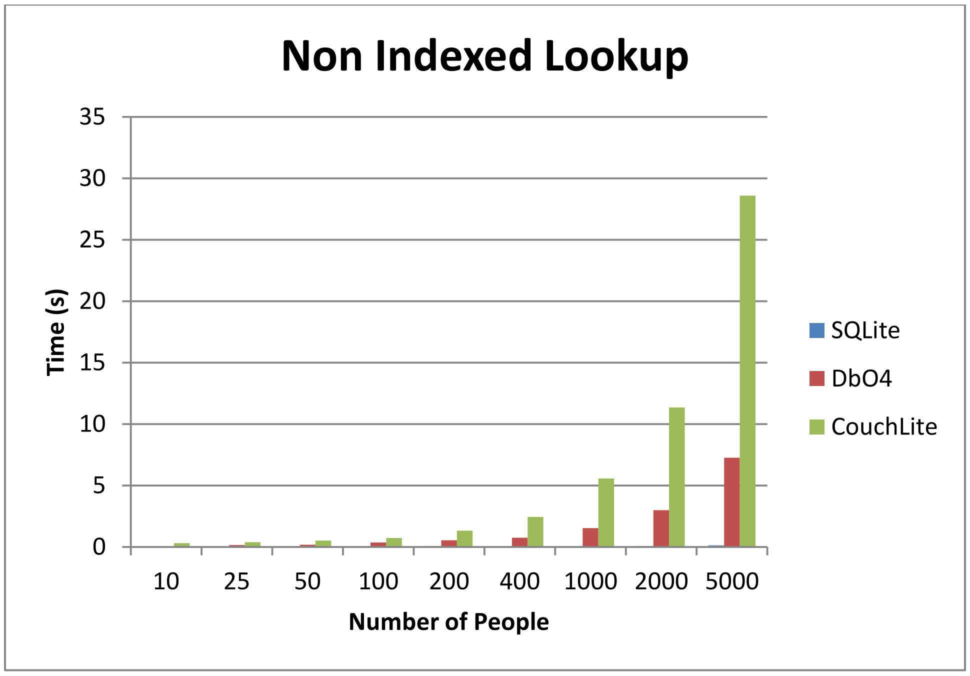

Figure 4: The time spent executing Test 2, which is a non-indexed lookup on a string field. Unlike in Test 1, where the look up field was an index, this time an increase in execution time as the number of tuples increases is easily observed. For DbO4 and CouchLite this increase is clearly exponential, while the increase for SQLite is not as dramatic. A similar pattern in execution time vs number of tuples was observed for Test 3, Test 4, and Test 6.

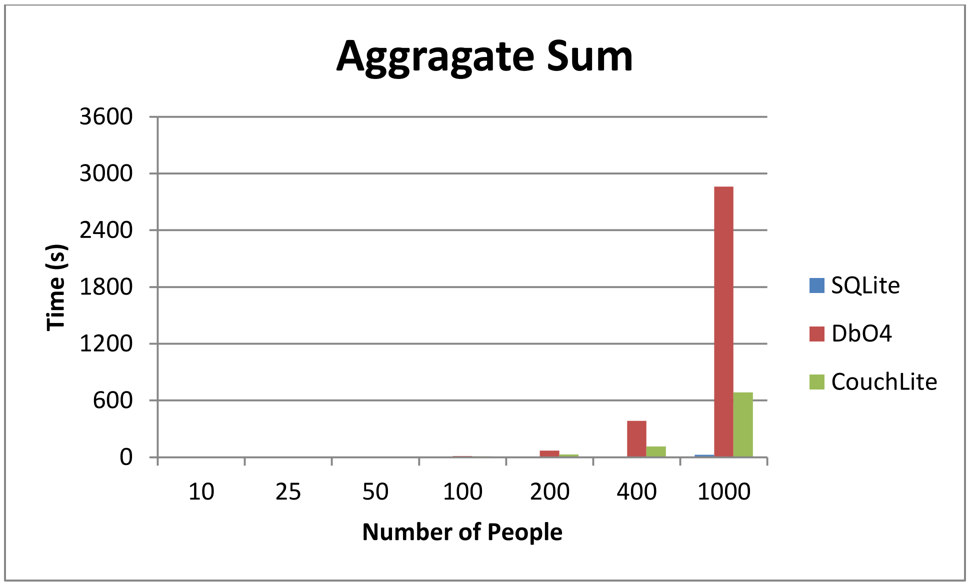

Figure 5: The time spent executing Test 5 which is an aggregate sum. At one thousand tuples this query took forty eight minutes for DbO4 and ten minutes for CouchLite, while only 25 seconds for SQLite. This query was taking so long that we excluded it from our benchmarks for 2000 and 5000 tuples. This query groups by a non-indexed field and sums over a floating point value. Neither DbO4 or CouchLite are optimized for this type of query.

Tuples	SQLite	DbO4	CouchLite
10	0.0199	0.0274	0.0793
25	0.031	0.0648	0.14046
50	0.11	0.1587	0.246
100	0.2856	0.528	0.54823
200	0.3295	1.5984	1.290916
400	0.498	7.3842	3.537616
1000	1.6969	50.341	15.252016

Table 1: The total time, in minutes, each database takes to run all benchmarks with a given number of input rows. There is a very clear ordering and SQLite consistently outperforms CouchLite which is slightly faster than DbO4.

Conclusion

We expected that each database would excel at some parts of each benchmark and do poorly on others and that based on the results we could recommend a different embedded database for different tasks. This turned out to only be true for DbO4 which was very fast at creating, inserting, and simple field based lookups but horrible at any query implementing a group by or aggregate function. SQLite performed very well under a high load and the most complicated queries we included in our benchmark.

CouchLite did preform adequately on indexed lookups and queries requiring only a simple map function, however, it did not scale very well. This makes sense since one of the primary use cases for NoSQL databases like CouchDB 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 our tests SQLite is the best embedded database for mobile phones.

References

1. M. Satyanarayanan, Fundamental challenges in mobile computing, Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, p.1-7, May 23-26, 1996, Philadelphia, Pennsylvania, United States [doi>10.1145/248052.248053]

2. Rick Cattell, Scalable SQL and NoSQL data stores, ACM SIGMOD Record, v.39 n.4, December 2010 [doi>10.1145/1978915.1978919]

3. Yingjie Shi , Xiaofeng Meng , Jing Zhao , Xiangmei Hu , Bingbing Liu , Haiping Wang, Benchmarking cloud-based data management systems, Proceedings of the second international workshop on Cloud data management, October 30-30, 2010, Toronto, ON, Canada

4. C. Turbyfill, C.U. Orji, D. Bitton, ASAP--an ANSI SQL standard scaleable and portable benchmark for relational database sytems, in: J. Gray (Ed.), The Benchmark Handbook, second ed., Morgan Kaufmann, Los Altos, CA, 1993

5. TPC - Transaction Processing Performance Council. http://www.tpc.org/.

6. Brian F. Cooper , Adam Silberstein , Erwin Tam , Raghu Ramakrishnan , Russell Sears, Benchmarking cloud serving systems with YCSB, Proceedings of the 1st ACM symposium on Cloud computing, June 10-11, 2010, Indianapolis, Indiana, USA [doi>10.1145/1807128.1807152]

7. Fröhlich, Nadine and Möller, Thorsten and Rose, Steven and Schuldt, Heiko. (2010) A benchmark for context data management in mobile context-aware applications. In: Proceedings of the 4th International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB 2010), 6 S.. Singapore.

8. McObject Benchmarks Embedded Databases on Android Smartphone. http://www.mcobject.com/march9/2009

9. http://www.sqlite.org/

10. https://github.com/couchbase/couchbase-lite-android

11. http://www.db4o.com/