Department of Computer Science and Engineering
Indian Institute of Technology, Jodhpur MAY 2018

PROJECT REPORT

MOBILE APP BENCHMARKING

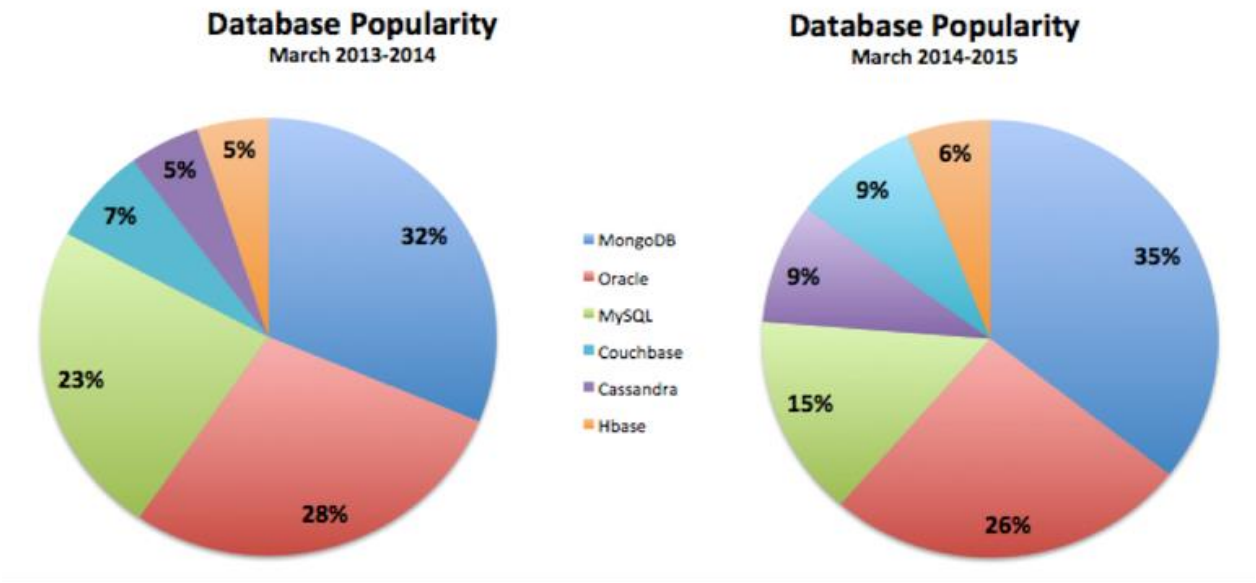
B-Tech Project Semester-4

Mentor:
Dr. Subhajit Sidhanta

Submitted By:
Manvendra singh(B16CS015)
Meet Mehta(B16CS016)

CONTENT

Problem Statement :



Find some standard benchmarking suites and report the results. Today most of the apps require database, and selecting the appropriate database is the very difficult task for developers as it determines the speed of the app to a great extent, slower apps has no place in the market today. Developers tend to escape from the SQL database, following this trend(which can be observed in Figure 1) blindly irrespective of the type of application they are building and requirement of their application. We need to find different parameters and circumstances under which one particular database is better than other.

Challenges

- First Challenge was to select proper benchmarking suite, that helps most developers.
- Working with android studio and monkey runner was also challenging in the beginning.
- Secondly integration was little bit difficult as this benchmarking suite was developed 2 years ago working with the latest compiler becomes difficult and some changes are to be made.
- Lack of documentation of involved tools.
- Working on windows environment was bit difficult as it involves many command line instructions.

- Benchmarking suites were created a long ago without mentioning their version requirements so it was difficult to run them with latest compiler without knowing their exact requirements.
- Today's all CPU's are having battery saving power so they don't give their best results in starting 2-3 cycles we need to remove them to get better results.
- Database can also affect the speed of the app because of their size too so we should also take size of database into consideration.

Approach:

Finding the Suite:

Mobile app performance depends upon various factors so deciding a particular benchmark was not a simple task.

We started with finding benchmarking suites for apps developed in different programming languages but very soon we realized that it is not a debatable topic because of higher speed of cpp and availability of resources in Java, We also find that a lot of researches have already taken place in this field, so we started thinking something else like algorithm or database but the problem with algorithm was that it is very difficult to categorise it because of it's possibly varying difficulty level, so finally we started with database selection as out criterion for selection of benchmarking suite.

After finding this that we need to select a benchmarking suite which can compare different databases and can be useful for finding comparison of their performance under various set of conditions.

We have selected a benchmarking suite that compares Realm to SQLite (and its ORMs greendao and ormlite) in a series of common operations:

- Batch Writes : write a bunch of objects in a single transaction.
- Simple Writes : write an object in a single transaction.
- Simple Queries : perform a simple query filter on:
  - integer equality
  - string equality
  - integer range

- Full Scan : perform a simple query that doesn't match any results (a full table scan).
- Sum : sum the values of an integer field
- Count : count the number of entries in a table.
- Delete: delete all entries in a table.

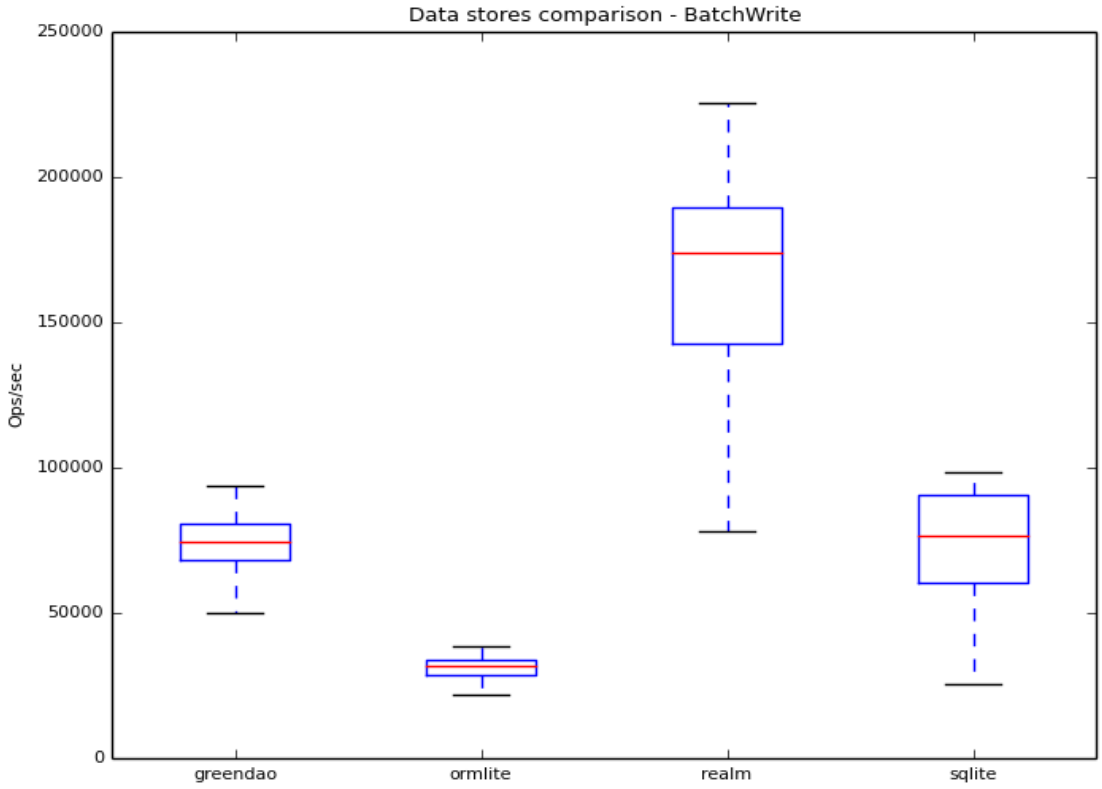
Integration and Running:

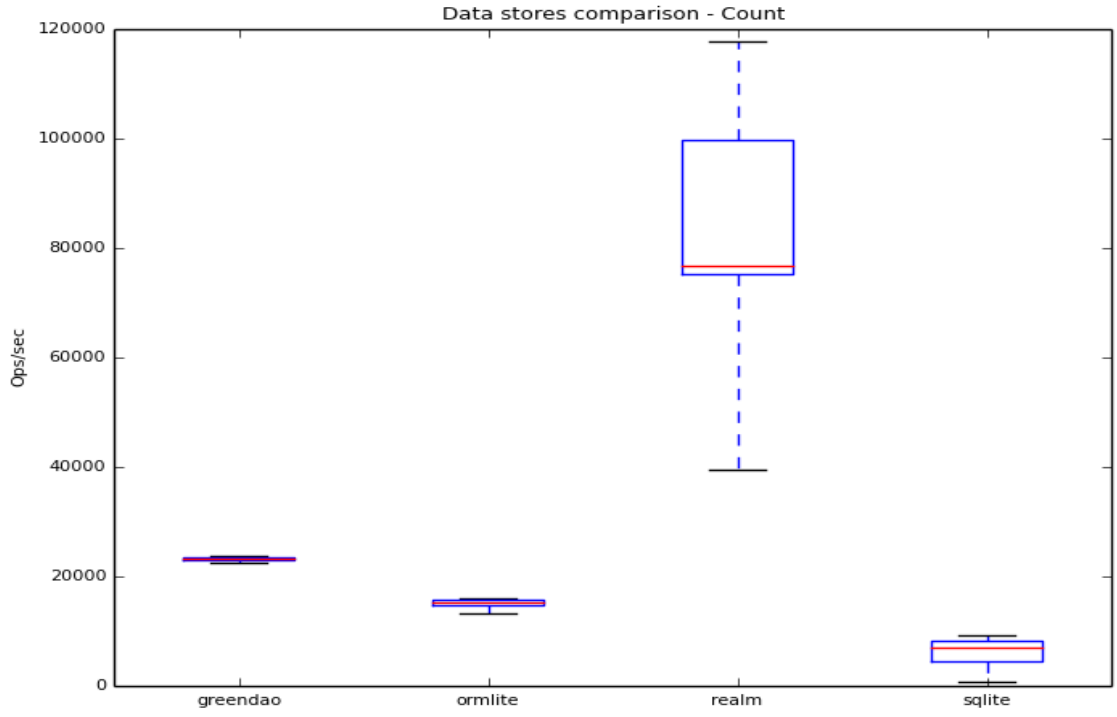
We have used android studio and monkey runner to run this benchmarking suite on emulator or device rather than using command line interface so that windows user can also use this benchmarking suite easilty. We updated the gradle version to 4.1 and java version to 9.0.1 so as to make compatible with benchmarking suite. This benchmarking suite collects data of throughput and stores in the external memory card of emulator or device. Finally, we copied this data from external memory card to hard drive of the pc using monkey runner. We have also used a python script to plot graphs using this data so that analysis becomes easier.

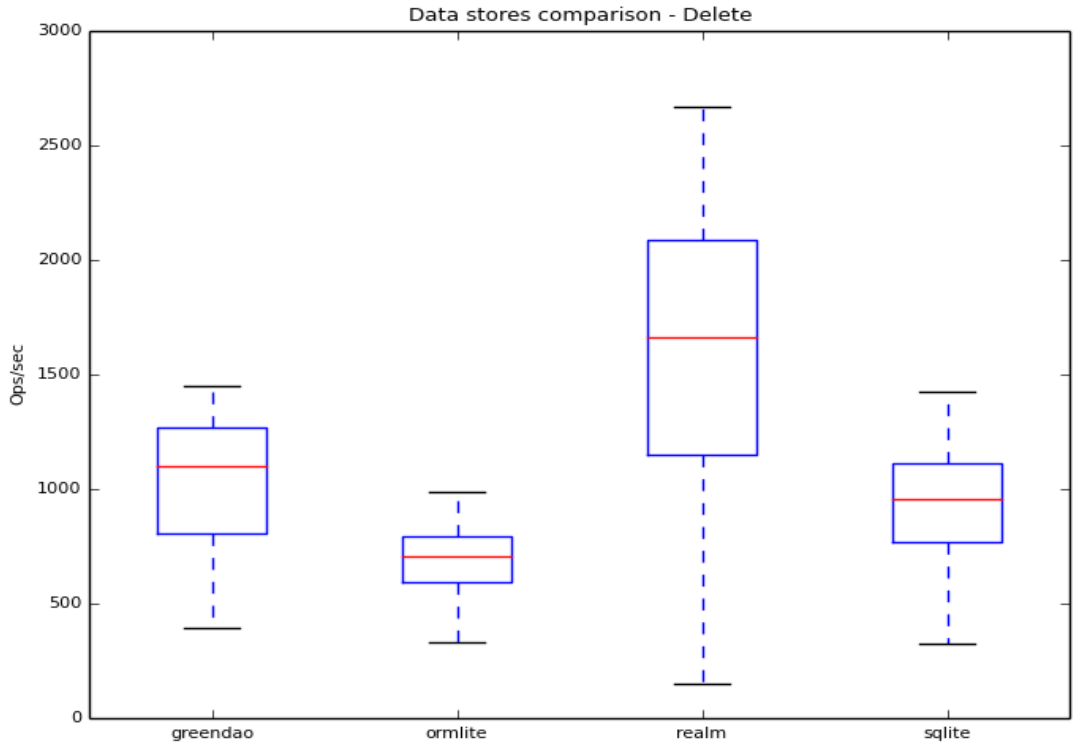
Different sized database were used to get more clear idea on the throughput. Also Many modern devices have CPUs that save power by reducing the speed of the CPU and only increasing the speed when there is a need. So we have used five iterations for CPU warmup and rest iterations were considered to obtain results so as to get better results.

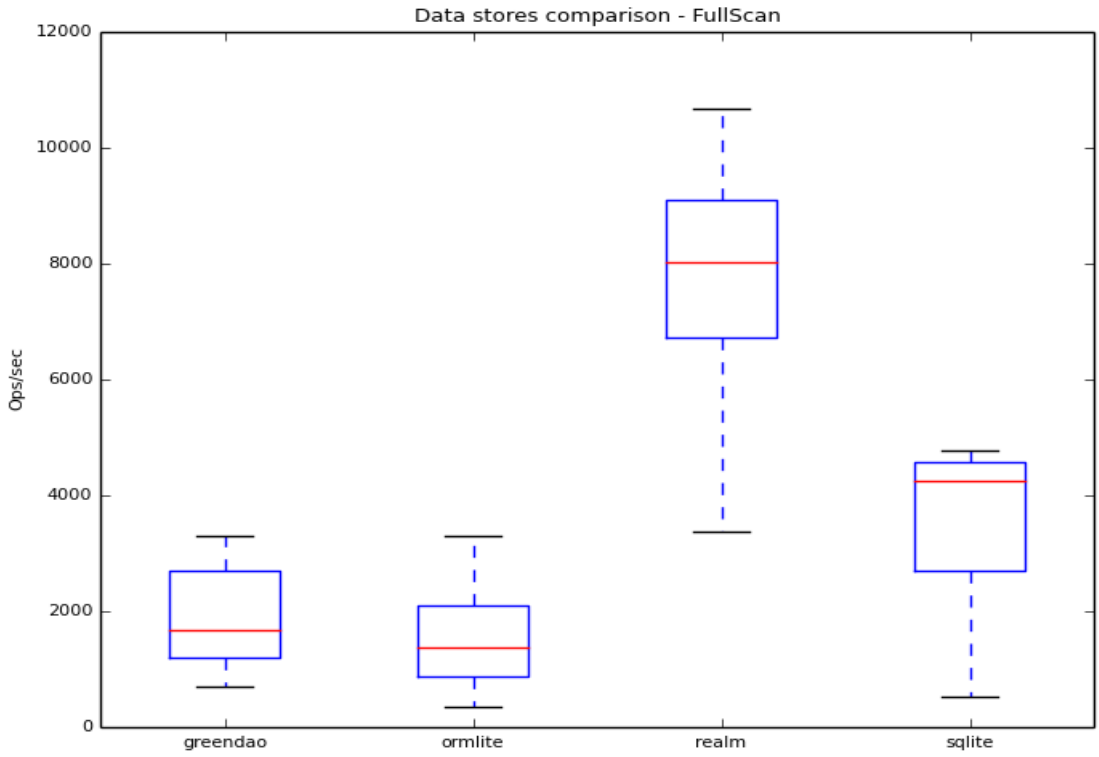
Results:

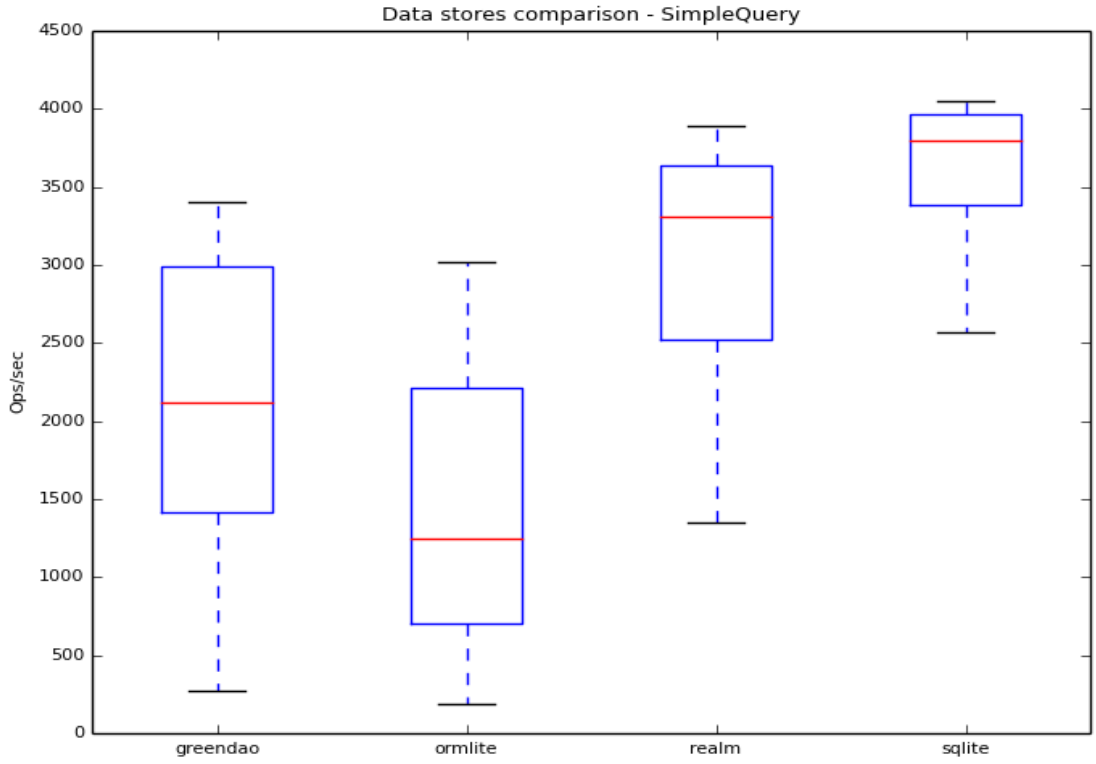
1. **Small Database:**

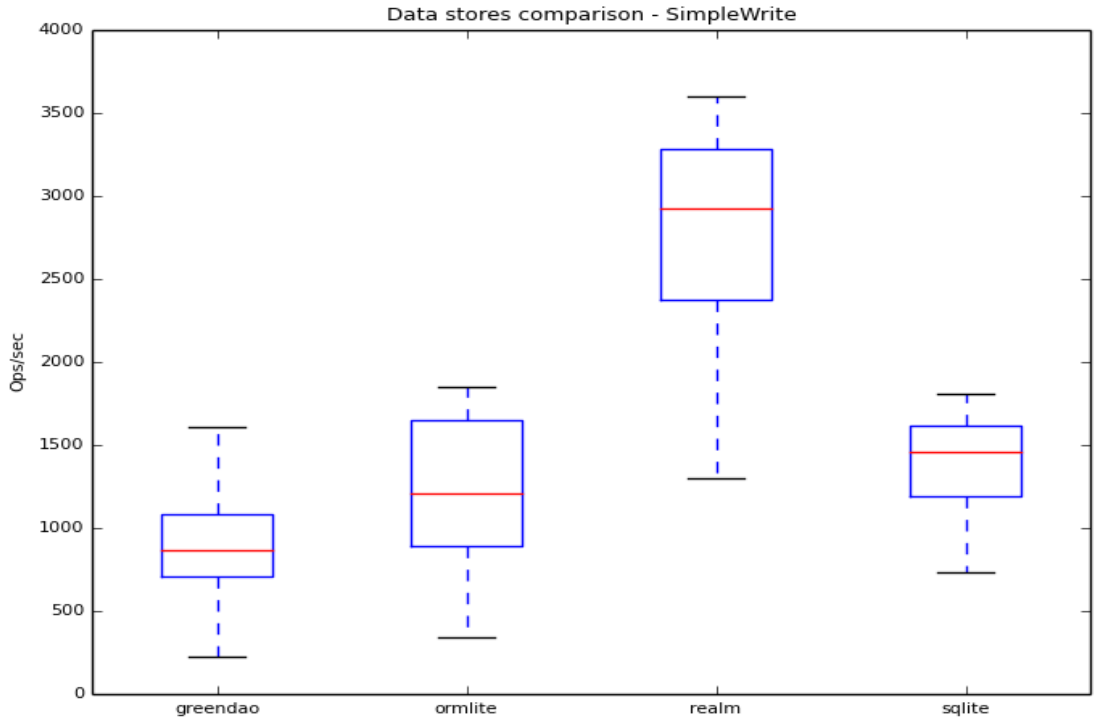

Data stores comparison - BatchWrite
Ops/sec
250000
200000
150000
100000
50000
0
greendao
ormlite
realm
sqlite

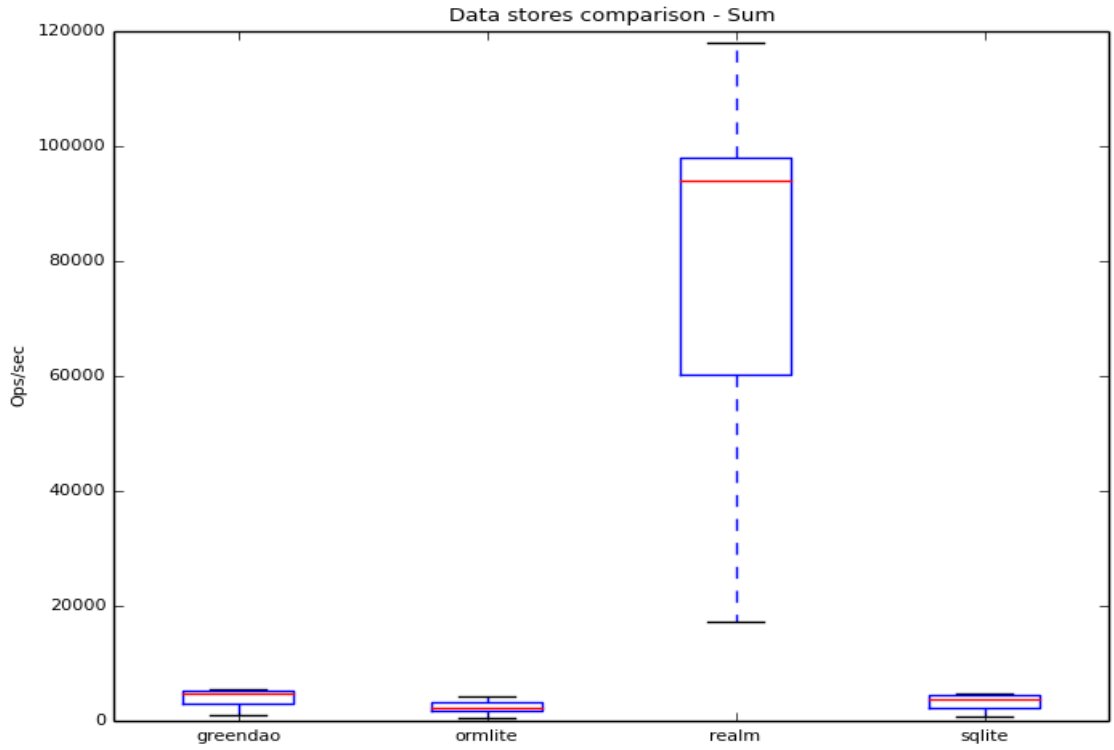

Data stores comparison - Count
Ops/sec
120000
100000
80000
60000
40000
20000
0
greendao
ormlite
realm
sqlite


Data stores comparison - Delete
Ops/sec
3000
2500
2000
1500
1000
500
0
greendao
ormlite
realm
sqlite

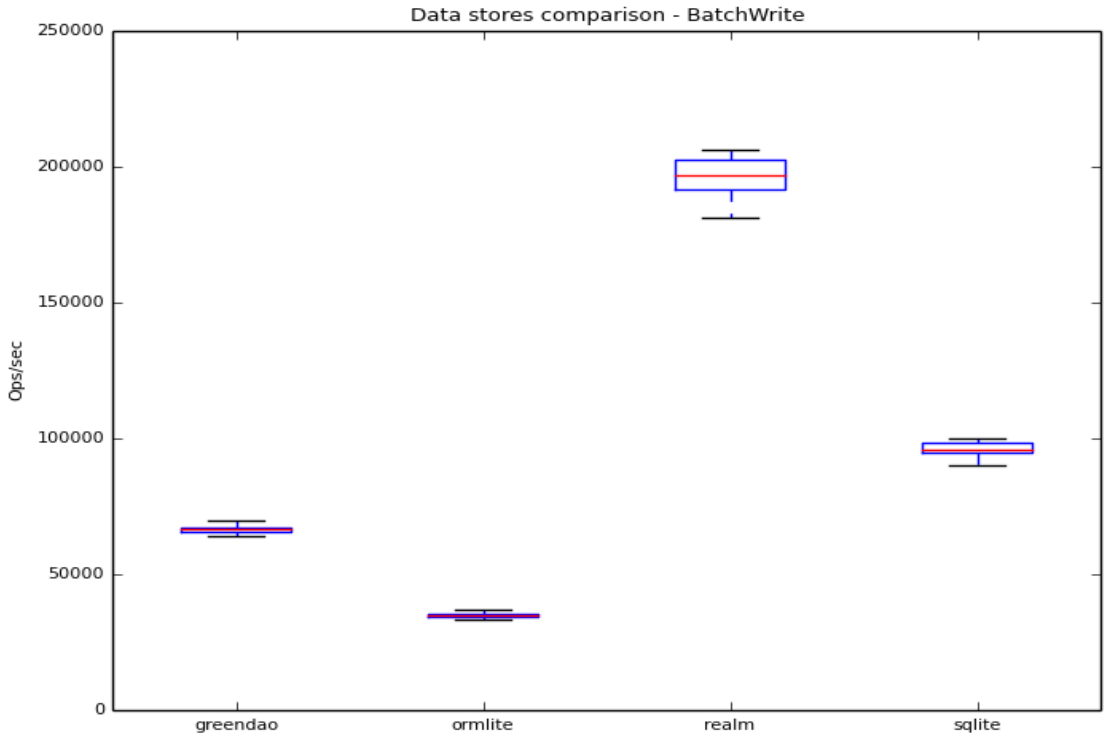

Data stores comparison - FullScan
Ops/sec
12000
10000
8000
6000
4000
2000
0
greendao
ormlite
realm
sqlite

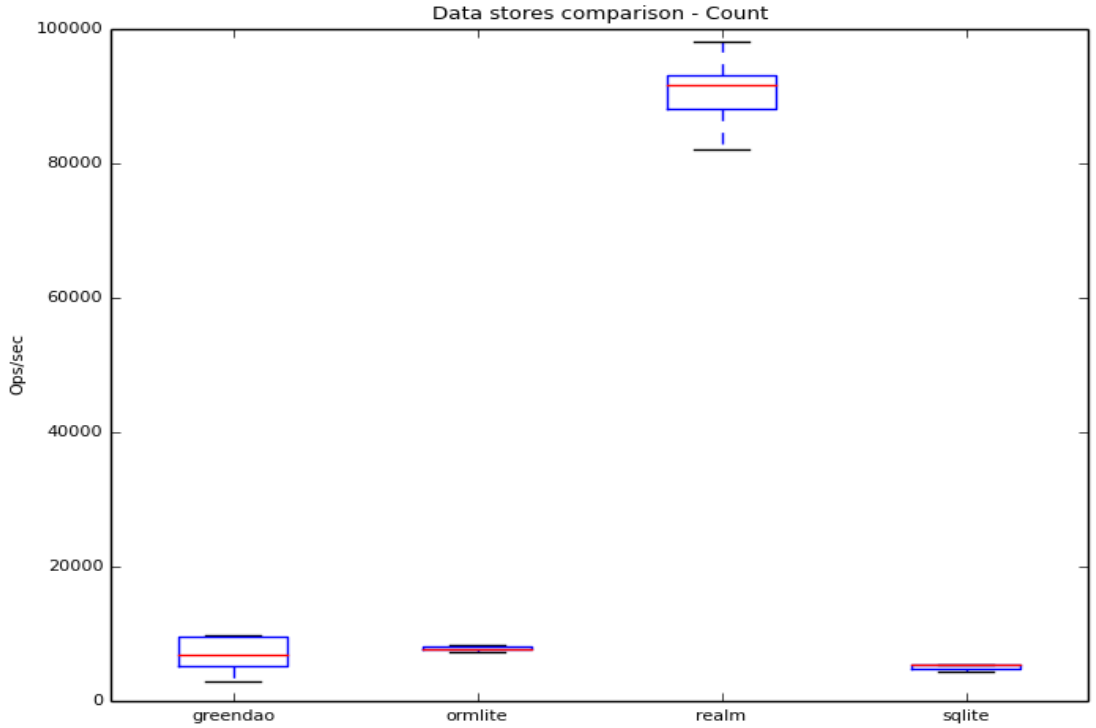

Data stores comparison - SimpleQuery
Ops/sec
4500
4000
3500
3000
2500
2000
1500
1000
500
0
greendao
ormlite
realm
sqlite

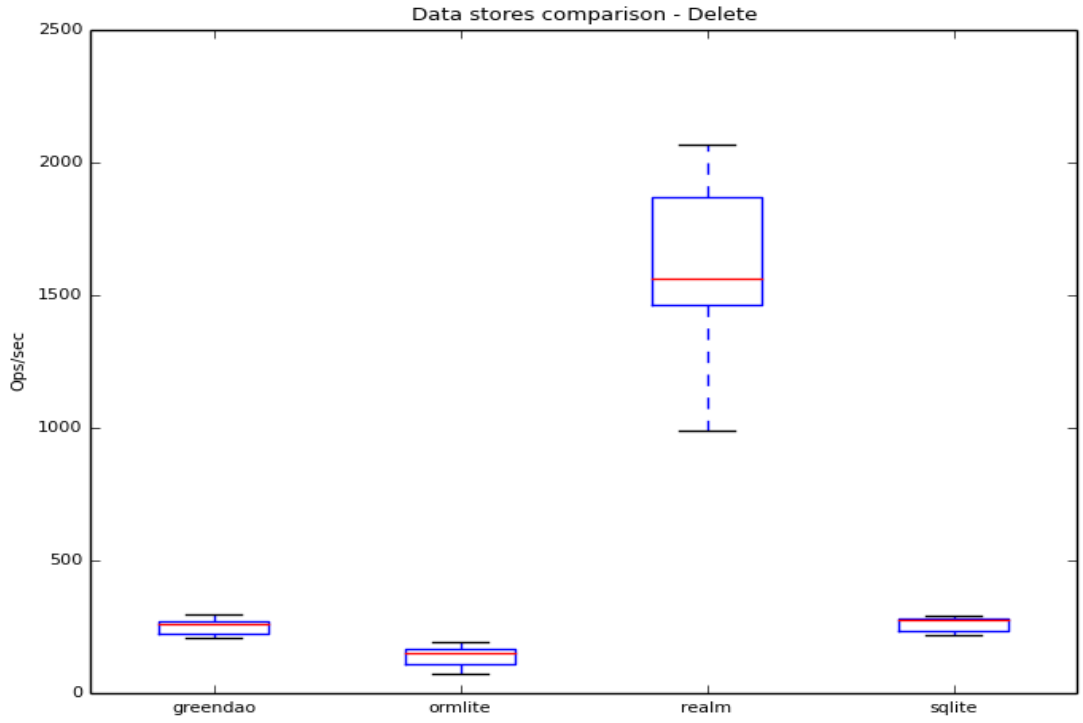

Data stores comparison - SimpleWrite
Ops/sec
4000
3500
3000
2500
2000
1500
1000
500
0
greendao
ormlite
realm
sqlite

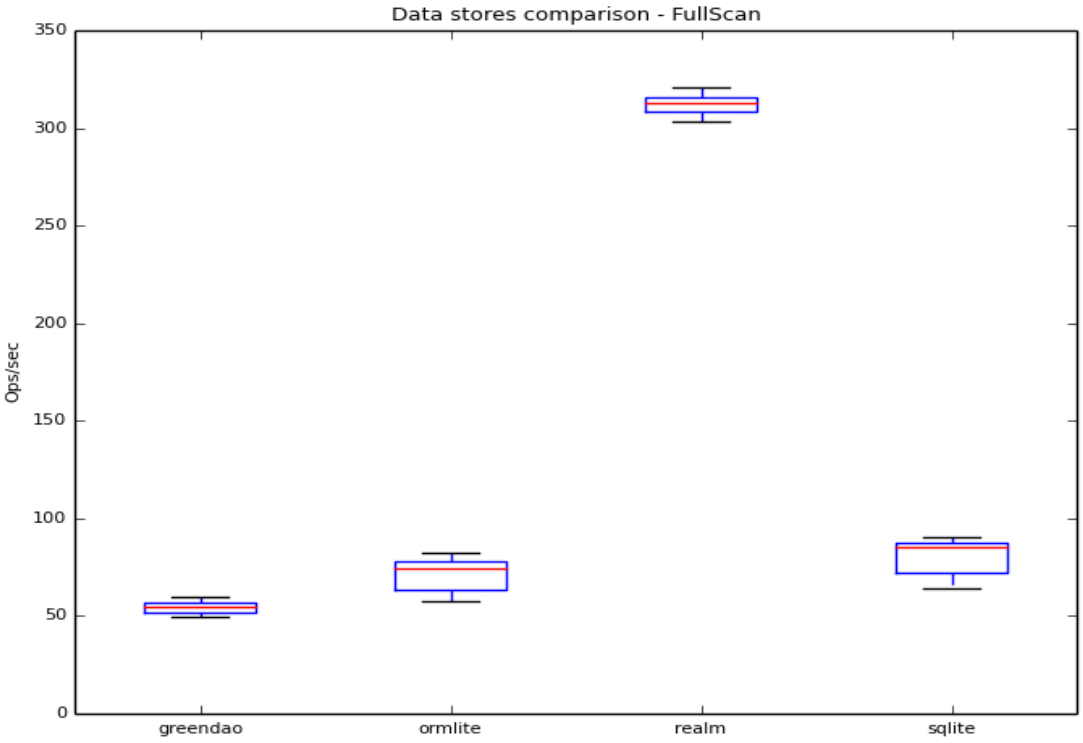


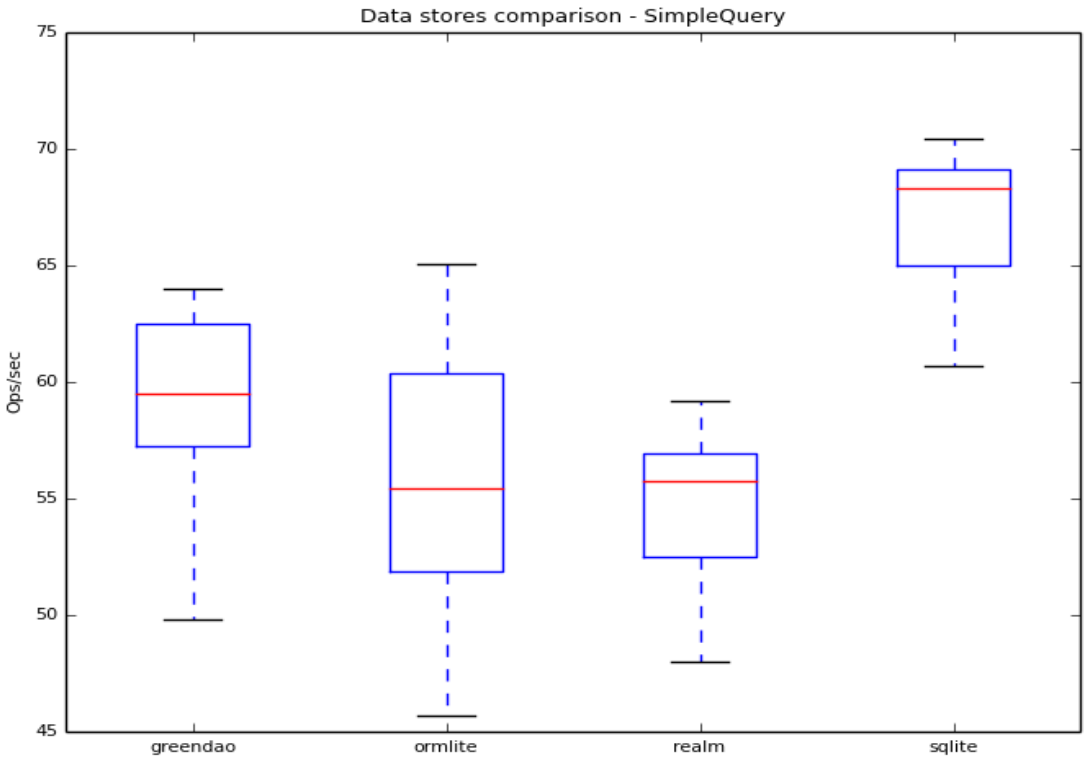
2. Large Database:

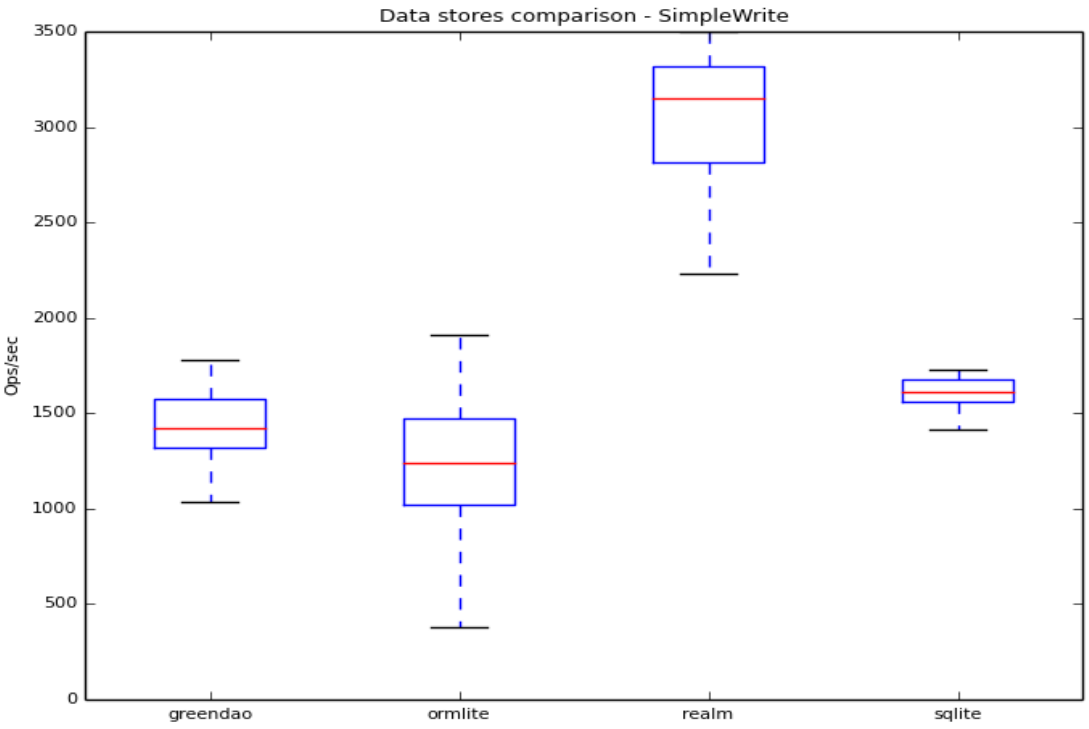


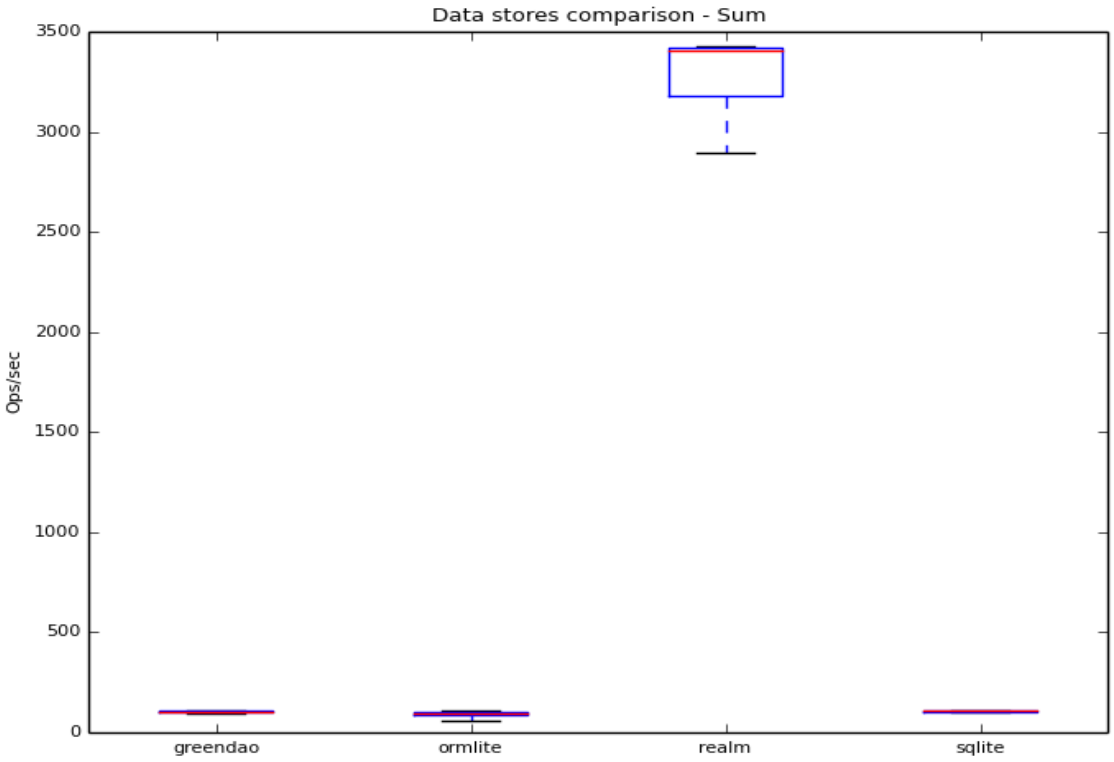

Data stores comparison - Count
Ops/sec
100000
80000
60000
40000
20000
0
greendao
ormlite
realm
sqlite


Data stores comparison - Delete
Ops/sec
2500
2000
1500
1000
500
0
greendao
ormlite
realm
sqlite


Data stores comparison - FullScan
Ops/sec
350
300
250
200
150
100
50
0
greendao
ormlite
realm
sqlite


Data stores comparison - SimpleQuery
Ops/sec
75
70
65
60
55
50
45
greendao
ormlite
realm
sqlite


Data stores comparison - SimpleWrite
Ops/sec
3500
3000
2500
2000
1500
1000
500
0
greendao
ormlite
realm
sqlite


Data stores comparison - Sum
Ops/sec
3500
3000
2500
2000
1500
1000
500
0
greendao
ormlite
realm
sqlite

Conclusion

It is clearly seen from the data that for most of the queries realm database is faster than SQLite and its ORMs.

How to Run:

For beginners :

1. Open Android Studio
2. Go to Open
3. Select folder realm-java-benchmarks-master
4. Generate an apk
5. Run it on some device (it will display message - "Running Benchmarks" while it is running and after it is done, message "done" will be displayed)
6. Its results will be stored in external memory card with the name of the folder "datastorebenchmark".
7. Copy all the files in that folder into a folder ./database-comparison-benchmarks/tools/1000
8. Run the python script graph.py to generate graphs in the same folder.

For Advanced Users :

1. Run the benchmark the following way:
2. Set NUMBER_OF_OBJECTS in app/src/main/java/io.realm.datastorebenchmark.Constants.java. The rest of this guide assumes 1000, which is also the default value.
3. Deploy the app to the phone or emulator. It will auto-start and report Done in the UI when complete. Don't touch the phone while it is running.
3. > ./gradlew installDebug> adb shell am start -a android.intent.action.MAIN -n

io.realm.datastorebenchmark/io.realm.datastorebenchmark.MainActivity

How to analyse - TLDR

4. The benchmark results from the supported datastores are analyzed the following way.
1. Goto the tools folder.
2. > cd tools
3. Copy all results from the phone/emulator using ADB to folder named after NUMBER_OF_OBJECTS:
5. > adb pull /sdcard/datastorebenchmark/ ./1000
6. Run python script:
7. > python dsb.py -b

Note – gradle must be upgraded according to the java version that is being used. Here gradle version 4.1 is used with java version – 9.0.1 .

Glossary:

Realm: **Realm** is an open-source object DBMS, it comes under the NoSQL category.
SQLite: It is a relational database management system.
ORM: It is a programming technique for converting data between incompatible type systems.

GreenDAO: It is a framework which works as ORM for SQLite.

References:

- https://developer.android.com/studio/test/monkey.html
- https://developer.android.com/studio/test/monkeyrunner/index.html
- https://github.com/realm/realm-java-benchmarks
- https://en.wikipedia.org/wiki/Box_plot