Düzce Üniversitesi
Bilim ve Teknoloji Dergisi

Araştırma Makalesi

Pratik Açıdan SQLite ve Firebase Veritabanlarının Bir Performans Karşılaştırması

Abdullah Talha KABAKUŞ *

Bilgisayar Mühendisliği Bölümü, Mühendislik Fakültesi, Düzce Üniversitesi, Düzce, TÜRKİYE
** Sorumlu yazarın e-posta adresi: talhakabakus@duzce.edu.tr*

ÖZET

Android an itibariyle dünyanın en çok kullanılan mobil işletim sistemidir. Android'in resmi olarak desteklediği iki veritabanı yönetim sistemi SQLite ve Firebase'dir. Android Yazılım Geliştirme Kit'i, geliştiricilere bu veritabanınlarında verilerini depolayan uygulamalar geliştirebilmeleri için dahili paketler sunmaktadır. Bu aşamada, bu veritabanlarının performans karşılaştırmasının açığa çıkartılması gerekmektedir. Bu sebeple, bu çalışma kapsamında en çok kullanılan veri işlemlerini kapsayan çeşitli deneyleri bu veritabanları üzerinde yürüten bir Android uygulaması geliştirilmiştir. Deneysel sonuçlar, veri silme dışında SQLite'ın Firebase'e göre daha iyi performans sağladığını göstermektedir. SQLite ile Firebase arasındaki performans farklılıkları (1) veri işlem tipine ve (2) yönetilen veri boyutuna bağlı olarak değişkenlik göstermektedir.

Anahtar Kelimeler: *Android, SQLite, Firebase, mobil işletim sistemi, veritabanı*

A Performance Comparison of SQLite and Firebase Databases from A Practical Perspective

ABSTRACT

Android is currently the most used mobile operating system all over the world. The two database management systems that Android officially supports are SQLite and Firebase. Android SDK provides built-in packages to let developers implement applications which store its data on these databases. At this point, it is necessary to reveal the performance comparison of these databases. For this reason, an Android application that evaluates several experiments which cover the most used data operations on these databases is implemented within this study. The experimental result indicates that SQLite provides better performance compared to Firebase except deleting data. The performance differences between SQLite and Firebase vary through (1) the type of data operation, and (2) the size of data that is managed.

Keywords: *Android, SQLite, Firebase, mobile operating system, database*

Geliş: 08/07/2018, Düzeltme: 24/10/2018, Kabul: 05/11/2018

I. INTRODUCTION

Android is an open source mobile operating system which is developed by Google and powered by the contribution of eighty-four technology and mobile companies under a group named Open Handset Alliance (OHA)[1]. During Google I/O 2017, Google has announced that Android is being used by more than 2 billion monthly active devices [1], [2]. According to a recent report by Statista[2], Android has dominated the global smartphone market mobile in the second quarter of 2017 with being used by 87.7% of smartphones globally [3]. Android is built on native libraries based on C/C++ programming languages which are compiled and preinstalled, and SQLite is one of these native libraries which is a relational database management system and comes in with built-in SQLite database implementation [4]–[7]. As a result of that, the APIs (Application Programming Interfaces) that are needed while implementing applications that utilize SQLite are available in the *android.database.sqlite* package. Despite that these APIs provide sufficient classes and methods to develop applications which interact with SQLite, Android official documentation highly recommends using Room Persistence Library[3] since the APIs that provide interacting SQLite directly are fairly low-level and require a great deal of time and effort to use [8]. Room Persistence Library provides an abstraction layer over the connection to an SQLite database which aims to increase the security of the database alongside making the database access easier [9]. Another big advantage of using Room Persistence Library is that it provides compile-time verification of raw SQL (Structured Query Language) queries [8] which is a better way to interact with the database without worrying much about the SQL queries, and opening and closing database connections [10]. For all these great reasons, Room Persistence Library is utilized for all interactions with SQLite in this study.

NoSQL (Not Only SQL) databases propose great advantages over SQL databases such as schema-free data structure and distributed architecture [11], [12]. While SQL databases rely on ACID (Atomicity, Consistency, Isolation, Durability) consistency model that ensures all the transactions are correctly committed and do not corrupt database, and the data are consistent, NoSQL databases are based on BASE (Basically Available, Soft-state, Eventually Consistent) consistency model in order to achieve scalability, high availability, and high performance [11], [13]–[17]. Another database that is officially supported by Android is Firebase which is a realtime cloud-hosted NoSQL database developed by Google. Firebase syncs data which is stored as JSON (Javascript Object Notation) across all clients in realtime and the data still remains available even the application goes offline due to network connectivity [18], [19]. Firebase clients are cross-platform which can be implemented using various programming languages such as Java, Javascript, and C++. Once the clients are connected, they all share one realtime database and automatically receive updates with the newest data [18]–[20]. Since both of these databases are officially supported by Android, the difference between them in terms of performance is needed to be revealed. Therefore, a performance comparison of SQLite and Firebase databases in terms of the elapsed times to complete various experiments is proposed in this study. This study covers the comparison of SQLite and Firebase from a practical perspective. The rest of the paper is structured as follows: Section 2 presents the related work. Section 3 describes the experimental setup where the experiments are evaluated on. Section 4 presents the experimental results and discussion. Finally, Section 5 concludes the paper with future directions.

[1] https://www.openhandsetalliance.com
[2] https://www.statista.com
[3] https://developer.android.com/training/data-storage/room/

II. RELATED WORK

Leavitt [17] compares SQL databases to NoSQL databases, and notes that even though NoSQL databases are fast for simple data operations, they are time-consuming for complex data operations. Leavitt lists the drawbacks of NoSQL databases over SQL databases. *Bartholomew* [11] proposes a brief history of SQL and NoSQL databases alongside the differences between them. He notes that SQL and NoSQL databases are complementary since they are designed to overcome different kinds of problems. *Sakr et al.* [21] propose a great survey about the data management approaches in cloud environments. They discuss the challenges of data management approaches face in the light of cloud. *Indrawan-Santiago* [22] proposes a comparison of ten NoSQL databases including MongoDB, HBase, CouchDB, and neo4j basis of data model, transaction model, license, ad-hoc query, indexes, and sharding. *Hecht and Jablonski* [23] compare fourteen NoSQL databases including MongoDB, CouchDB, Cassandra, and HBase in the light of their data models, support for queries, partitioning, replication, and concurrency controls.

Boicea et al. [24] compare MongoDB as a NoSQL database to Oracle as a relational database. They report that if the necessity is a fast, easily scalable database that is not in need of to store its data in relations, then MongoDB is the right choice. On the other hand, if the necessity is having a more complex database with relations between tables and a fixed structure, Oracle is reported as the right choice. They compare MongoDB and Oracle through the three experiments: (1) Elapsed time to insert data, (2) elapsed time to update data, and (3) elapsed time to delete data. The experiments are evaluated for various numbers of data, from 10 records to 1,000,000 records. For all experiments, they report that MongoDB provides better performance than Oracle.

Li and Manoharan [12] compare the performance of SQL and NoSQL databases according to the five experiments: (1) Elapsed time to instantiate the storage, (2) elapsed time to read the value corresponding to a given key, (3) elapsed time to write key-value pair into the storage, (4) elapsed time to delete the record corresponding to a given key, and (5) elapsed time to fetch all the keys from the storage. The experiments are evaluated for various numbers of data, from 10 records to 100,000 records. The databases they evaluate are MongoDB, RavenDB, CouchDB, Cassandra, Hypertable, Couchbase, and Microsoft SQL Server Express. According to the experimental result, it is reported that Couchbase and MongoDB provide the best overall performance for read, write, and delete operations. They also note that Couchbase does not provide any APIs to fetch all the keys available in the storage.

Lee and Shih [25] propose a framework that automatically migrate the data from a traditional SQL database to a NoSQL database. The proposed model preserves the relations defined on the traditional SQL database. In order to evaluate whether the migrated NoSQL database is well-functioned, the authors utilize two machine learning algorithms namely K-means and Random Forest. According to experimental results, the NoSQL database provides better performance compared to the SQL database in term of shorter execution time. Parker et al. [26] compares an SQL database namely Microsoft SQL Server to a NoSQL database namely MongoDB. According to their experimental results, the NoSQL database provides better performance for the experimented insert, update, and simple queries which are not based on relations. For the update query and queries based on non-key attributes as well as aggregation, the SQL database provides better performance. *Agarwal and Rajan* [27] compares an SQL database namely PostGIS to a NoSQL database namely MongoDB for spatial and aggregate queries. The experiments they evaluate indicate that the NoSQL database provides better performance

by an average factor of 10x-25x which increases exponentially when the size of data increases in both indexed and non-indexed operations.

Abramova et al. [28] compare the performance of five NoSQL databases namely Cassandra, HBase, MongoDB, OrientDB, and Redis using Yahoo! Cloud Serving Benchmark [29] which is an open source workload generator tool. They compare the execution times of these five NoSQL databases through the different workloads which are generated by changing the ratios of read, update, and delete operations. As an overall analysis, they report that NoSQL databases can be divided into two categories: (1) The databases which are optimized for reads, and (2) the databases which are optimized for updates. They note that while MongoDB, Redis, and OrientDB are optimized for read operations, Cassandra and HBase provide better performance for update operations.

III. Experimental Setup

All experiments are evaluated on the same machine whose hardware and software specifications are listed in Table 1. During the experiments, the unnecessary processes of the operating system are stopped in order to prevent any possible side effects. As the speed test result is listed in Table 1, the speed of internet connection is fast enough to upload or download the data without any latencies while evaluating experiments on Firebase.

Table 1. *The hardware and software specifications of the machine which is used to evaluate experiments*

Operating System	CPU	RAM	Network Download Speed	Network Upload Speed
Windows 10 (64-bit)	Intel Core i7-7700HQ 2.80 GHz, 4 cores	32 GB	48.8 Mbps	4.95 Mbps

An Android application is developed in order to evaluate several experiments on SQLite and Firebase databases which are discussed in Section 4. The developed application is deployed on a virtual device based on Google's smartphone Nexus 5 whose hardware and software specifications are listed in Table 2. In order to prevent any possible latencies which are critical for the experiments on the realtime cloud-hosted database, Firebase, the network speed, and network latency are set to "Full" and "None", respectively. The study targets the latest version of Android in order to provide an up-to-date comparison of both databases and the operating system. The device the experiments are evaluated on uses the latest version of Android, Android 8.0, which is also consistent with the Room Persistence Library.

Table 2. *The hardware and software specifications of the device which is used to run the developed application*

Android Version	API Level	RAM	ABI	Network Speed	Network Latency
Android 8.0 (Oreo)	26	1,536 MB	X86	Full	None

The elapsed time for each experiment is calculated using the *System.currentTimeMillis* method of Java programming language by calculating the time difference in milliseconds between just before and after each experiment is evaluated as it is illustrated in Fig. 1

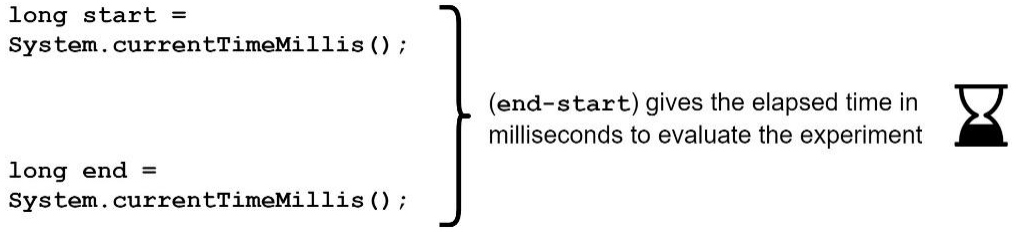

Figure 1. *Illustration of how the elapsed time is calculated for each experiment*

For the sake of representation of data, a sample model is designed as its properties with their data types are listed in Table 3. The "*id*" property represents the primary key of the model and it is incrementally assigned from starting 1 to the number of samples. The "*text*" property represents the sample *String* data which is generated by using the *RandomStringUtils* class of *Apache Commons Lang*[4] library. *RandomStringUtils* class proposes the method named "*random*" which generates random text which can be customized through the parameters. For the experiments in this study, a *String* which is 100 characters-length and contains letters but not numbers is generated for the "*text*" property.

Table 3. *The properties of the sample model and their data types*

Property	Data Type
id	Integer
text	String

[4] https://commons.apache.org/proper/commons-lang/

IV. EXPERIMENTAL RESULTS AND DISCUSSION

Five experiments to be evaluated on SQLite and Firebase are designed in order to compare the performances of databases in terms of most common data operations which are (1) inserting data into the database, (2) querying the database for a record, (3) fetching the whole data available in the database, (4) updating data, and (5) deleting data from the database. The templates of the queries used to evaluate the experiments are listed in Table 4. For the sake of evaluating the experiments, an Android application is implemented using the Java programming language that evaluates the designed experiments and reveals the experimental result of each experiment on both databases. Each designed experiment and its experimental result are discussed in the following subsections.

Table 4. *The templates of the SQL queries used to evaluate the experiments*

Experiment	SQL Query Template
Inserting Data into The Database	*insert into tweet values([id], [text])*
Querying for A Single Sample	*select * from tweet where id=[id]*
Fetching data	*select * from tweet*
Updating data	*update tweet set text=[text] where id=[id]*
Deleting data	*delete fom tweet*

A. EXPERIMENT #1 - INSERTING DATA INTO THE DATABASE

This experiment evaluates the performances of databases in terms of inserting data. For the sake of comparison, different numbers of samples (from 1 sample to 100,000 samples) are generated and inserted into each database as the evaluation result is presented in Fig. 2. As it can be conducted from the Fig. 1, when the size of data is increased, SQLite provides better performance compared to Firebase. The major reason behind this difference is that the data which is needed to be uploaded in order to insert it into Firebase since it is a realtime cloud-hosted database. Another conclusion which is obtained from the experimental result is that Firebase provides better performance when the size of data which is going to be inserted is less than 100.

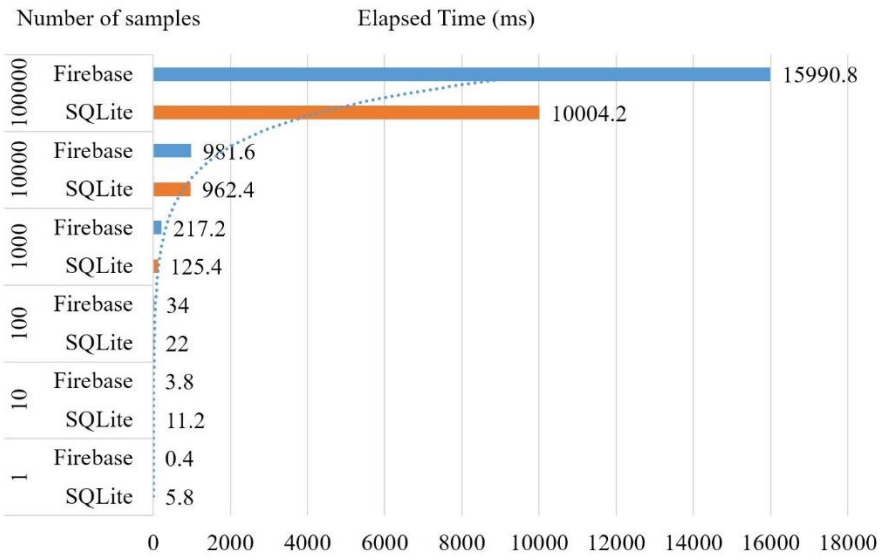

Figure 2. *The experimental result of inserting different numbers of data into SQLite and Firebase*

B. EXPERIMENT #2 - QUERYING FOR A RECORD

This experiment evaluates the performances of databases in terms of querying the whole data for a record. For the sake of identifying the performance of querying the whole data entirely, the queried record is specifically selected to be not included in the database. The experimental result which is presented in Fig. 3 clearly indicates that SQLite provides a lot better performance (about 100 times) compared to Firebase when it comes querying the whole data for a single record. Another important conclusion obtained from the experimental result is that the query performance of Firebase quiet similar despite the number of data dramatically changes.

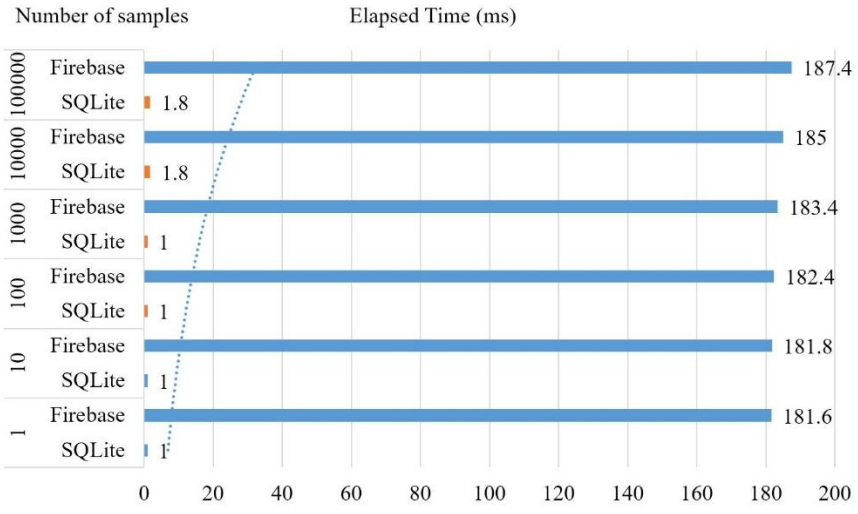

Figure 3. *The experimental result of querying different numbers of data from SQLite and Firebase*

C. EXPERIMENT #3 - FETCHING DATA

This experiment evaluates the performances of databases in terms of fetching the whole available data. For the sake of comparison, different numbers of data (from 1 sample to 100,000 samples) are generated and inserted into each database before the experiment is evaluated. As it can be conducted from the Fig. 4 which presents the experimental result, it is quite obvious that SQLite provides a lot better performance when it comes to fetching the whole data. The result is reasonable since the data is needed to be downloaded for Firebase due to being a cloud-hosted database.

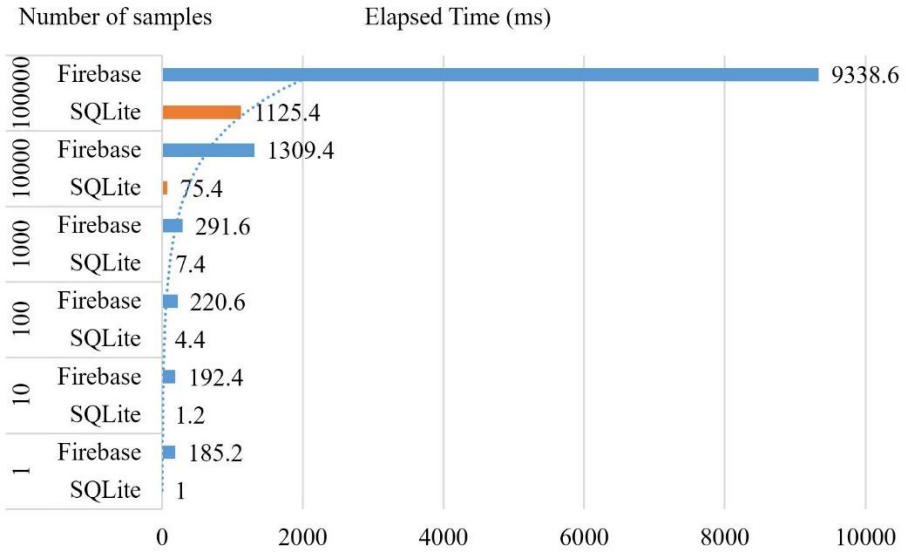

Figure 4. *The experimental result of fetching different numbers of data from SQLite and Firebase*

D. EXPERIMENT #4 - UPDATING DATA

This experiment evaluates the performances of databases in terms of updating data. For the sake of comparison, different numbers of data (from 1 sample to 100,000 samples) are generated and inserted into each database before the experiment is evaluated. It can be conducted from the experimental result which is presented in Fig. 5 that SQLite provides slightly better performance than Firebase in terms of updating data. Another conclusion which is obtained from the experimental result is that Firebase provides better performance for updating a single sample despite being a cloud-hosted database.

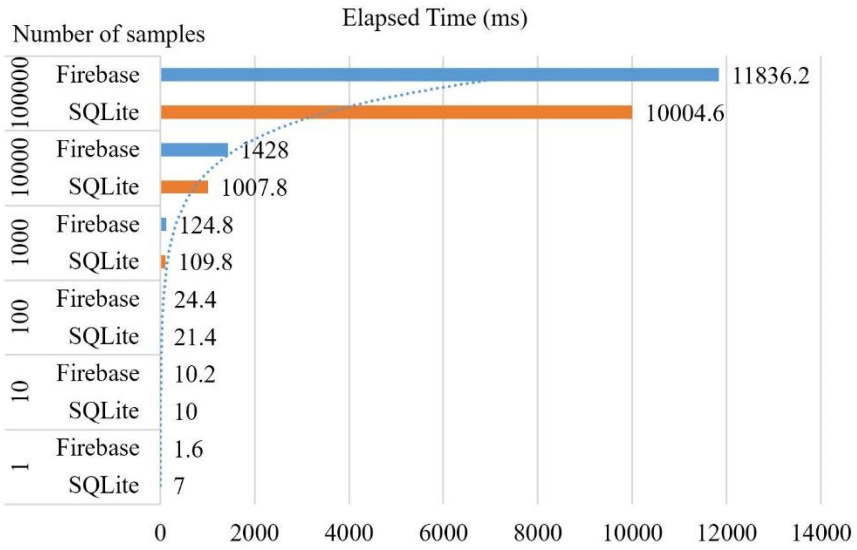

Figure 5. *The experimental result of updating different numbers of data in SQLite and Firebase*

E. EXPERIMENT #5 - DELETING DATA

This experiment evaluates the performances of databases in terms of deleting data. For the sake of comparison, different numbers of data (from 1 sample to 100,000 samples) are generated and inserted into each database before the experiment is evaluated. As it can be conducted from the Fig. 6 which presents the experimental result, Firebase provides significantly better performance compared to SQLite for any sizes of data. The performance difference becomes even more evident when the size of data increases.

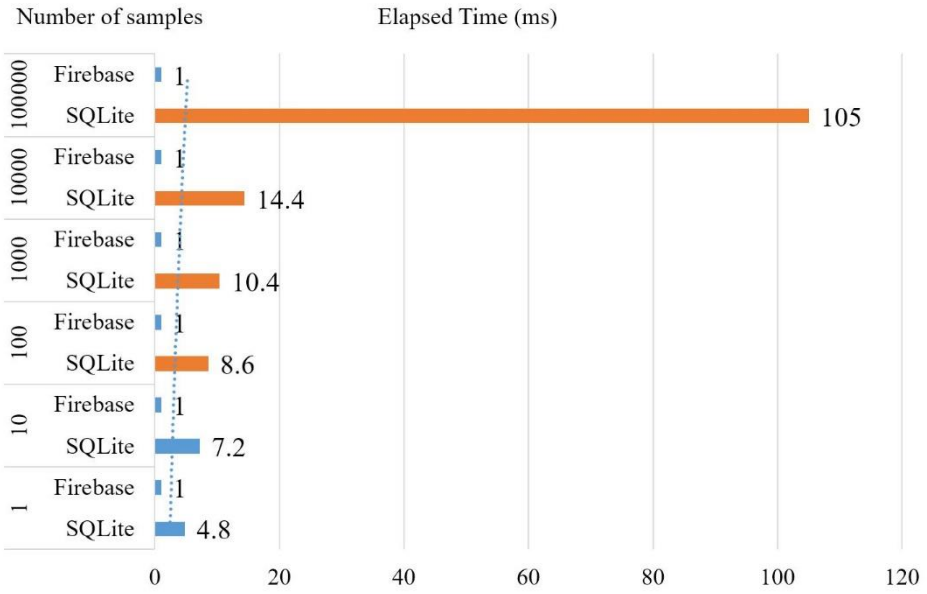

Figure 6. *The experimental result of deleting different numbers of data from SQLite and Firebase*

V. CONCLUSION

Android officially supports both SQLite and Firebases databases by providing built-in packages in order to let developers implement their applications which store its data on one of these databases. SQLite as a relational database and Firebase as a NoSQL database are based on different consistency models and provide different ways of handling data operations. As a natural consequence of that, the performances of these databases are expected to be different. Alongside that, another difference which affects the performances of databases is that while SQLite serves from local, Firebase is a realtime cloud-hosted database. By considering all these differences, the performance differences of SQLite and Firebase databases in terms of elapsed time to complete several experiments which cover the most common data operations are revealed in this study. The experimental result clearly indicates that SQLite provides better performance than Firebase for the most types of data operations except deleting data. Alongside the performance criteria, SQLite can be preferred instead of Firebase when (1) the data is not shared between different types of clients (i.e. desktop application, web application), and (2) the local storage is not limited in terms of available space. On the other hand, Firebase can be preferred instead of SQLite when (1) the data of the application is shared amongst different types of clients, and (2) the local storage is limited in terms of available space. The technical reasons behind these performance differences are needed to be explained in detail which are not in the scope of this study and planned to be a part of a future work.

VI. REFERENCES

[1] B. Popper, "Google announces over 2 billion monthly active devices on Android," *The Verge*, 2017. [Online]. Available: https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users. [Accessed: 07-Jul-2018].

[2] D. Burke, "Android: celebrating a big milestone together with you," *Google*, 2017. [Online]. Available: https://www.blog.google/products/android/2bn-milestone/. [Accessed: 07-Jul-2018].

[3] "Smartphone OS global market share 2009-2017," *Statista*, 2018. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. [Accessed: 07-Jul-2018].

[4] Padmini, *Android App Development: A Complete Tutorial For Beginners*. Educreation Publishing, 2016.

[5] P. K. Dixit, *Android*. Vikas, 2016.

[6] S. Hashimi, S. Komatineni, and D. MacLean, *Pro Android 3*. Apress, 2011.

[7] I. Krajci and D. Cummings, *Android on x86 : an introduction to optimizing for Intel architecture*, 1st ed. Apress, 2013.

[8] "Save data using SQLite | Android Developers." [Online]. Available: https://developer.android.com/training/data-storage/sqlite. [Accessed: 05-Jul-2018].

[9] P. Mainkar, *Expert Android Programming: Master skills to build enterprise grade Android applications*, 1st ed. Birmingham, UK: Packt Publishing, 2017.

[10] E. Obugyei and N. Raman, *Learning Kotlin by building Android Applications: Explore the fundamentals of Kotlin while building real-world Android applications*. Packt Publishing, 2018.

[11] D. Bartholomew, "SQL vs. NoSQL," *Linux J.*, vol. 2010, no. 195, pp. 54–59, 2010.

[12] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings*, 2013, pp. 15–19.

[13] M. Carro, "NoSQL Databases," *CoRR*, vol. abs/1401.2, 2014.

[14] J. D. Cook, "ACID versus BASE for database transactions," 2009. [Online]. Available: https://www.johndcook.com/blog/2009/07/06/brewer-cap-theorem-base/. [Accessed: 07-Jul-2018].

[15] S. K. Gajendran, "A Survey on NoSQL Databases," 2012.

[16] D. Pritchett, "Base: an Acid Alternative," *Queue*, vol. 6, no. 3, pp. 48–55, 2008.

[17] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," *Computer (Long. Beach. Calif).*, vol. 43, no. 2, pp. 12–14, 2010.

[18] L. Moroney, *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*. Seattle, WA, USA: Apress, 2017.

[19] N. Chatterjee, S. Chakraborty, A. Decosta, and A. Nath, "Real-time Communication Application Based on Android Using Google Firebase," *Int. J. Adv. Res. Comput. Sci. Manag. Stud.*, 2018.

[20] R. Das, S. Mondal, and N. Mukherjee, "MoRe-care: Mobile-assisted remote healthcare service delivery," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, 2018, pp. 677–681.

[21] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A Survey of Large Scale Data Management Approaches in Cloud Environments," *Commun. Surv. Tutorials, IEEE*, 2011.

[22] M. Indrawan-Santiago, "Database research: Are we at a crossroad? Reflection on NoSQL," in *Proceedings of the 2012 15th International Conference on Network-Based Information Systems, NBIS 2012*, 2012, pp. 45–51.

[23] R. Hecht and S. Jablonski, "NoSQL evaluation: A use case oriented survey," in *2011 International Conference on Cloud and Service Computing (CSC 2011)*, 2011, pp. 336–341.

[24] A. Boicea, F. Radulescu, and L. I. Agapin, "MongoDB vs Oracle - Database comparison," in *Proceedings of 3rd International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2012*, 2012, pp. 330–335.

[25] C. Lee and Z. Shih, "A Comparison of NoSQL and SQL Databases over the Hadoop and Spark Cloud Platforms using Machine Learning Algorithms," in *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, 2018, pp. 1–2.

[26] Z. Parker, S. Poe, and S. V. Vrbsky, "Comparing NoSQL MongoDB to an SQL DB," in *Proceedings of the 51st ACM Southeast Conference on - ACMSE '13*, 2013.

[27] S. Agarwal and K. Rajan, "Analyzing the performance of NoSQL vs. SQL databases for

Spatial and Aggregate queries Analyzing the performance of NoSQL vs. SQL databases for Spatial and Aggregate queries," in *Free and Open Source Software for Geospatial (FOSS4G)*, 2017.

[28] V. Abramova, J. Bernardino, and P. Furtado, "Which NoSQL Database? A Performance Overview," *Open J. Databases*, vol. 1, no. 2, pp. 17–24, 2014.

[29] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, pp. 143–154.