Understanding the Energy, Performance, and Programming Effort Trade-offs of Android Persistence Frameworks

Jing Pu, Zheng "Jason" Song, Eli Tilevich
Software Innovations Lab
Virginia Tech, Blacksburg, VA, USA
Email: {pjing83, songz, tilevich}@cs.vt.edu

***Abstract*—One of the fundamental building blocks of a mobile application is the ability to persist program data between different invocations. Referred to as *persistence*, this functionality is commonly implemented by means of persistence frameworks. When choosing a particular framework, Android—the most popular mobile platform—offers a wide variety of options to developers. Unfortunately, the energy, performance, and programming effort trade-offs of these frameworks are poorly understood, leaving the Android developer in the dark trying to select the most appropriate option for their applications. To address this problem, this paper reports on the results of the first systematic study of six Android persistence frameworks (i.e., ActiveAndroid, greenDAO, OrmLite, Sugar ORM, Android SQLite, and Java Realm) in their application to and performance with popular benchmarks, such as DaCapo. Having measured and analyzed the energy, performance, and programming effort trade-offs for each framework, we present a set of practical guidelines for the developer to choose between Android persistence frameworks. Our findings can also help the framework developers to optimize their products to meet the desired design objectives.**

I. Introduction

Any non-trivial application provides the ability to preserve and retrieve user data, both during the application session and across sessions. This ability is called *persistence*, and in object-oriented applications is commonly implemented by means of object-relational mapping (ORM) or object-oriented (OO) frameworks. These frameworks relieve the developer from the necessity to write raw SQL to interact with the underlying database engines and thus streamline the development process.

As mobile devices continue to replace desktops as the primary computing platform, Android is poised to win the mobile platform contest, taking the 82.8% share of the mobile market in 2015 [1] with more than 1.6 million applications developed thus far [2]. Energy efficiency remains one of the key considerations when developing mobile applications, as the energy demands of applications continue to exceed the devices' battery capacity. Consequently, in recent years researchers have focused their efforts on providing Android developers with insights that can be used to improve the energy efficiency of mobile applications. The research literature on the subject includes approaches ranging from general program analysis and modeling [3], [4], [5] to application-level analysis [6], [7].

Despite all the progress made in understanding the energy impact of programming patterns and constructs, a notable omission in the research literature on the topic is energy behaviors of persistence frameworks. Without understanding the energy, performance, and programming effort trade-offs of persistence, one cannot gain a comprehensive insight on the overall energy efficiency of modern mobile applications. Although an indispensable building block of mobile applications, persistence has never been systematically studied in this context.

This paper reports on the results of a comprehensive study we have conducted to measure and analyze the energy, performance, and programming effort trade-offs of popular Android persistence frameworks. To that end, we consider the persistence libraries most widely used in Android applications [8]. In particular, we study five widely used ORM persistence frameworks (ActiveAndroid, greenDAO, OrmLite, Sugar ORM, Android SQLite), and one OO persistence framework (Java Realm) as our experimental targets [9], [10], [11], [12], [13], [14]. These frameworks operate on top of the popular SQLite or Realm database engines.

In an effort to understand the noticeable performance and programming effort disparities between different persistence frameworks, our experiments apply these six persistence frameworks to different benchmarks, and then compare the resulting energy consumption, runtime performance, and programming effort (i.e., the amount of programmer-written code). Our benchmarks include a set of micro-benchmarks designed to measure the performance of individual database operations as well as a well-known DaCapo H2 database benchmark [15]. We also conclude a set of guildlines to help developers select a suitable persistence framework for their applicat-ions.

Hence, the overriding goal of our study is to help Android mobile developers decide which persistence framework they should choose to achieve the desired energy/performance/-programming effort balance for the development scenario at hand. Depending on the amount of persistence functionality in a given application, the choice of a persistence framework may dramatically impact the levels of energy consumption and runtime performance. By precisely measuring and thoroughly analyzing the energy/performance/programming effort

characteristics of alternative Android persistence frameworks, this study aims at obtaining a deeper understanding of the persistence's impact on the mobile software development ecosystem.

Based on our experimental results, the main contributions of this paper are as follows:

1 To the best of our knowledge, this is the first study to empirically evaluate the energy, performance, and programming effort trade-offs of widely used Android persistence frameworks.
2 Our experiments consider multifaceted combinations of factors which may impact the energy consumption and performance of persistence functionality in real-world applications, which include persistence operations involved, the volume of persisted data, and the number of transactions.
3 Based on our experimental results, we offer a series of guidelines for Android mobile developers to select the most appropriate persistence framework for their mobile applications. For example, ActiveAndroid fit well for applications processing relatively large data volumes in a read-write fashion. These guidelines can also help the framework developers to optimize their products for the mobile market.

The rest of this paper is organized as follows. Section II provides the background and related work information for this research. Section III describes the design of our experimental study. Section IV presents the study results, listing our findings and offering guidelines for Android developers. Section V summarizes our conclusions.

II. Background and Related Work

To set the context for our work, this section describes the persistence functionality, as it is realized by means of database engines and persistence frameworks. Then it also discusses some prior approaches that have studied performance and energy efficiency.

The ORM (object-relational mapping) frameworks have been introduced and refined to facilitate the creation of database-oriented applications [16]. The prior studies of the ORM frameworks mainly have focused on their *performance efficiency*. However, as the energy demands of mobile applications continue to exceed the battery capacities of mobile devices, one cannot neglect *energy efficiency* as a key consideration when analyzing the performance of ORM frameworks on mobile platforms. Studies have shown that a lot of software development related factors can significantly influence the energy consumption of a software system[17] (e.g., design patterns involved, the Model-View-Controller architecture, information hiding, implementation of persistence layers, code obfuscation, refactoring, and data structure usage). In this paper, we compare the energy consumption of different ORM frameworks in a mobile execution environment with the goal of understanding the results from the software design perspective.

Multiple prior studies have focused on profiling the energy consumption of different applications/system calls [3], [4], [5], [6], [7]. The research literature includes approaches ranging from general program analysis and modeling, to application level analysis. However, to guide the mobile developers in selecting the most suitable ORM framework for their applications, we should further consider both the runtime performance and the programming effort incurred.

Android persistence frameworks A persistence framework serves as a middleware layer that bridges the application logic with the database engine's operations. The differences between object-oriented and relational models have been known as *the object-relational impedance mismatch*. Specifically, the object-relational mapping (ORM) and object-oriented frameworks that we study operate as follows. The database engine maintains a schema in memory or on disk, and the framework provides a programming interface for the application to interact with the database engine.

We evaluate six frameworks: Android SQLite, ActiveAndroid, greenDAO, OrmLite, Sugar ORM, and Java Realm, backed up by the SQLite and Realm database engine, which are customized for mobile devices, with limited resources, including battery power, memory, and processor.

III. Experiment Design

In this section, we explain the main design decisions we had to make in designing our experiments. In particular, we discuss the benchmarks, the measurement variables, and the experimental parameters.

A. Benchmark Selection

DaCapo H2, a well-known Java database benchmark that interacts with the H2 Database Engine via JDBC. To adapt this benchmark for Android, we replace H2 with SQLite or Realm. This benchmark manipulates a considerable volume of data to emulate bank transactions. The benchmark includes 1) a complex schema and non-trivial functionality, obtained from a real-world production environment. The database structure is complex (12 tables, with 120 table columns and 11 relationship between tables), while the database operations simulate the running of heavy-workload database-oriented applications; 2) complex database operations that require: batching, aggregations, and transactions.

However, using the DaCapo benchmark alone would leave unanswered the questions of the performance of persistence frameworks under the low data volumes with simple schema conditions. To establish a baseline for our evaluation, we thus designed a set of micro benchmarks, referred to as *the Android ORM Benchmark*, which features a simple database schema with few data records. Specifically, this benchmark's database structure includes 2 tables comprising 11 table columns, and a varying small number of data records. Besides, this micro-benchmark comprises the fundamental database operation invocations "create table", "insert", "delete", "select", "update". As the database operations in many mobile applications tend to be rather simple, the micro-benchmark's results present valuable insights for application developers.

B. Parameters and Variables

Our experimental setup comprises a mobile application that uses each of the benchmarked frameworks to execute both DaCapo and the Android ORM Benchmark[1]. Our experiments fix different settings while varying the persistence frameworks.

Next, we explain the variables used to evaluate the performance, energy consumption, and programming effort of the studied persistence frameworks. We also describe how these variables are obtained.

- **Overall Execution Time:** is the time elapsed from the point when a database transaction is triggered to the point when it stops.
- **Read/Write Database Operation Number:** is obtained by hooking into the SQLite operation interfaces provided by the Android System Library. We focus on comparing the Read/Write numbers only on SQLite-based frameworks (ActiveAndroid, greenDAO, OrmLite and Sugar ORM). When performing the same combination of transactions, the differences in Read/Write number is the output of how different persistence frameworks interpret database operation invocations. The read/write ratio can also impact the energy consumption.
- **Energy Consumption:** is obtained by monitoring the real-time current, voltage, and power of the the Android device's battery. We use the Monsoon Power Monitor [18] to monitor the energy of the device battery.
- **Uncommented Line of Codes (ULOC):** reflects the programming effort in terms of how much code the programmer has to write to use each persistence framework.

We further introduce the input parameters for different benchmarks. For the DaCapo benchmark, we want to explore the performance boundary of different persistence frameworks under a heavy workload. Therefore, we vary the amount of total transactions to a large scale, and record the overall time taken and energy consumed.

For the micro benchmark, we study the "initialize", "insert", "select", "update" and "delete" invocations in turn. We change the number of transactions for the last four invocations, so for the "select", "update" and "delete" invocations, the amount of data records also changes. Therefore, the input parameters for the micro benchmark is a set of two parameters, $\{\text{NUMBER OF TRANSACTIONS}, \text{AMOUNT OF DATA RECORDS}\}$.

C. Experimental Hardware

All the measurements are conducted on an LG LS740 smartphone, with 1GB of RAM, 8GB of ROM and 1.2GHz quad-core Qualcomm Snapdragon 400 processor, running Android 4.4.2 KitKat operating system. The device has a 3000mAh removable Lithium Ion battery. All experiments are executed as the only load on the device's OS. We run each benchmark 5 times within the same environment, with the first two runs to warm up the system, and the reported data as the average of the last 3 runs.

[1]All the code used in our experiments can be downloaded from https://github.com/AmberPoo1/PEPBench.

IV. Study Results

In this section, we report and analyze our experimental results.

A. Experiments with the Android ORM benchmark

In this group of experiments, we study how the types of operation (insert, update, select and delete) and the variations on the number of transactions impact energy consumption and performance with different frameworks using micro benchmark[2] The experimental results for each type of persistence operation are presented in Fig.1. The first row, Fig 1(a)-(c) shows the energy consumption, execution time, and read/write operations of the "insert" database invocation, and Fig 1(d)-(f), (g)-(i), (j)-(l) show that of the "select", "update" and "delete" database invocations respectively.

The results show that the persistence frameworks differ in terms of their respective energy consumption, performance, read, and write measurements. Next, we compare the results by operation:

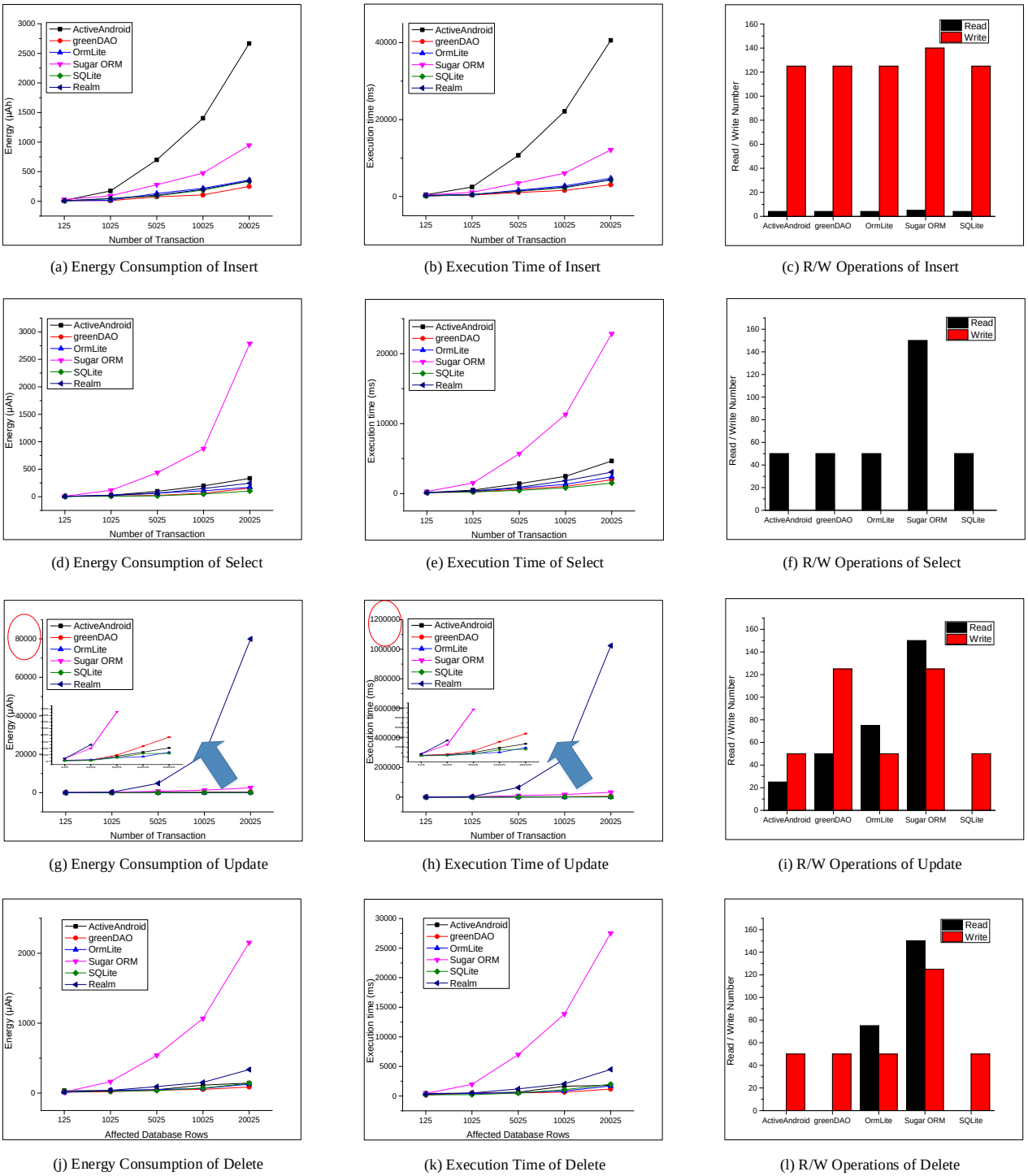
Insert We observe that ActiveAndroid has the longest insert operation runtime, while Sugar ORM is second longest, with the remaining frameworks showing similar performance levels. The runtime trace reveals that interactions with the cache triggered by insert in ActiveAndroid are expensive, costing 62% of the overall execution time. Sugar ORM performs the highest number of database operations, a measurement that explains its performance. By contrast, greenDAO's performance is the best, due to its simple but efficient batch insert encapsulation.

Update From (g) and (h) we can observe that, the cost of Java Realm update is several orders of magnitude larger than other frameworks especially when the number of transactions grows. One reason is that Java Realm lacks support for batch update. Another cause is that its update procedure invokes the underlying Realm library method—`TableView.size()`—performed on a memory-hosted list of entities and costing 98.3% of the overall execution time. The cost of Sugar ORM is still high because it has the highest number of read and write operations. Sugar ORM needs to find the target object before updating it. This finding procedure involves an expensive recursive design for the `SugarRecord.find()` method, costing 96% of the overall execution time.

Select and Delete For select and delete operations, we observe from (d,e,f) and (j,k,l) that Sugar ORM 1) has the worst performance in terms of execution time and energy consumption; 2) has the highest number of database operations, as it executes an extra query for each atomic operation. Sugar ORM's select and delete inefficiency related to the extra underlying operations, but as mentioned above, most of the execution time is spent in the recursive $find$ method. OrmLite, greenDAO and Android SQLite show comparable performance levels with these two operations.

Table I sums up the ranking of each persistence framework w.r.t. different database operation invocations. We also

[2]We use the terms—micro benchmark and the Android ORM benchmark interchangeably in the rest of the presentation.



(a) Energy Consumption of Insert

(b) Execution Time of Insert

(c) R/W Operations of Insert

(d) Energy Consumption of Select

(e) Execution Time of Select

(f) R/W Operations of Select

(g) Energy Consumption of Update

(h) Execution Time of Update

(i) R/W Operations of Update

(j) Energy Consumption of Delete

(k) Execution Time of Delete

(l) R/W Operations of Delete

Fig. 1. Energy/Performance/Read and Write for Android ORM Benchmark Initialization with Alternative Persistence Frameworks

measure the ULOC for implementing all the basic database operation invocations for each persistence framework and include it in the table. From Table I and our above analysis, we can draw the following conclusions:

1 By adding up the ranking of different operations, we can rank these frameworks in terms of their overall performance: Android SQLite >greenDAO >OrmLite >Java Realm >Sugar ORM $\geq$ ActiveAndroid , where ">" means "faster than".

2 Considering the programming effort when using different frameworks, greenDAO, sugar ORM and ActiveAndroid require less programming effort.

Compared Item	ActiveAndroid	greenDAO	OrmLite	Sugar ORM	Android SQLite	Java Realm
ULOC	**253**	**241**	**326**	**226**	**306**	**313**
Initialization Ranking	**6**	**4**	**5**	**1**	**3**	**2**
Insert Ranking	**6**	**1**	**4**	**5**	**2**	**3**
Update Ranking	**3**	**4**	**2**	**5**	**1**	**6**
Select Ranking	**5**	**2**	**3**	**6**	**1**	**4**
Delete Ranking	**3**	**1**	**2**	**6**	**3**	**5**
Summed up Ranking	**23**	**12**	**16**	**23**	**10**	**20**

TABLE I
COMPARISON OF PERSISTENCE FRAMEWORKS IN THE ANDROID ORM EXPERIMENT

3 Considering the programming effort of implementing all database operations using different frameworks, greenDAO can be generally recommended for developing database-oriented mobile application with standard database operation/schema complexity.
4 Sugar ORM would not be an optimal choice when the dominating operations in a mobile application are select or delete, while Java Realm would not be optimal when the dominating operation is update.

B. Experiments with the DaCapo benchmark

In DaCapo experiments, we study how the energy consumption and performance of each framework changes in relation to the number of executed bank transactions. The benchmark comes with a total of 41,971 records, so in our experiments we differ the number of bank transactions.

In Fig.2, (a) shows the energy/performance of the DaCapo Initialization. The dominant database operation in this phase is insert, and (a) shows the performance levels consistent with those seen in the Android ORM benchmark for the same operation: Sugar ORM and ActiveAndroid have the longest runtime. greenDAO performs better than Android SQLite, possibly due to greenDAO supporting batch insert.

In our measurements, we vary the number of bank transactions over the following numbers: 40, 120, 200, 280, 360, 440, 520, 600, 800, 1000, 1500. The total number of transactions is the sum of basic bank transactions, as listed in Table II. Each transaction comprises a complex set of database operations. The key transactions in each run are "New Order", "Payment by Name", and "Payment by ID", which mainly execute the "query" and "update" operations. In our experiments, "New Order" itself takes 42.5% of the entire number of transactions. (b),(c) show the execution time and energy consumption for each transaction number, and Table II shows the average time consumption for each transaction.

From Table II, we observe that Java Realm and Sugar ORM have the longest execution time when executing the transactions whose major database operation is update (e.g., "New order", "New order rollback", "Payment by name", and "Payment by ID"). This conclusion is consistent with that derived from the Android ORM update experiments above. Android SQLite takes rather long to execute, as it involves database aggregation (e.g., sum, and the table queried had 30,060 records) and arithmetic operations (e.g. $field - 1$) in the select clause. Meanwhile, as ActiveAndroid only uses raw SQL manipulation interface for complex update operations, its performance is the fastest, albeit at the cost of additional programming effort.

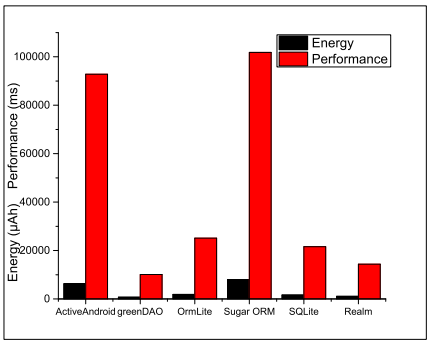
Table II also shows that greenDAO, ActiveAndroid, and Android SQLite incur higher execution costs for the "Stock level" transaction. One possible explanation is that this transaction contains a multiple entities conjunctive query action, and only these three frameworks provide the SQL "JOIN" interface. Supporting this interface is known to be computationally expensive. However, the SQL "JOIN" interface can help save the programming effort.

From Fig.2 (a-c) and Table III we can conclude that:

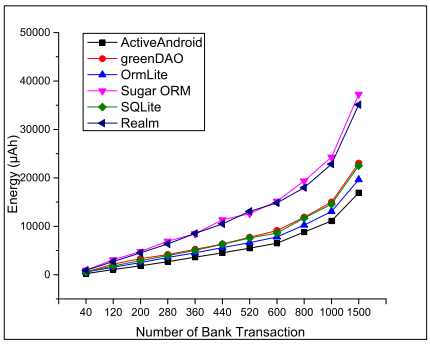
1 ActiveAndroid offers the overall best performance for all DaCapo transactions. It shows the best performance for the most common transactions, at the cost of additional programming effort. Besides, its execution invokes the smallest number of database operations, due to its caching mechanism.
2 Sugar ORM and Java Realm have the longest execution time, in line with the Android ORM benchmark's results above.
3 greenDAO's performance is in the middle, while requiring the lowest programming effort, taking 24.5% fewer uncommented lines of code to implement than the other frameworks.

V. CONCLUSION

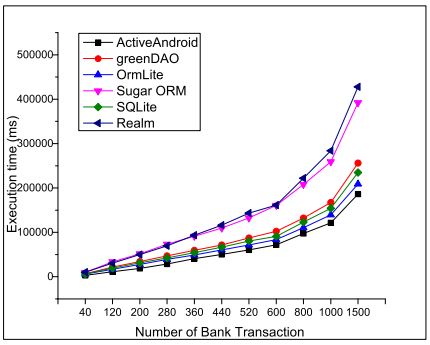
In this paper, we present a systematic study of popular Android ORM/OO persistence frameworks. We first compare and contrast the frameworks to present an overview of their features and capabilities. Then we present our experimental design of two sets of benchmarks, variables, and input parameters, used to explore the performance, energy consumption and programming effort of these frameworks in different application scenarios. We describe the benchmark results, and also use our analysis of the framework features and capabilities to explain the results. Finally, we summarize a set of guildlines from our experiments to help mobile developers in their decision making process when choosing a persistence framework for a given application. To the best of our knowledge, this research is the first step to better understand the trade-offs between the performance, energy efficiency, and programming effort of Android persistence frameworks.



(a) Energy Consumption/Execution Time of DaCapo Initialization



(b) Energy Consumption of DaCapo Transactions



(c) Execution Time of DaCapo Transactions

Fig. 2. Energy/Performance/Read and Write for DaCapo Benchmark with Alternative Persistence Frameworks

Transaction Type	ActiveAndroid	greenDAO	OrmLite	Sugar ORM	Android SQLite	Java Realm
Stock level	136	189	86	91	98	41
Order status by name	72	101	95	100	112	36
Order status by ID	106	91	94	113	108	33
Payment by name	50	55	50	124	59	86
Payment by ID	25	32	40	119	44	60
Delivery schedule	1	1	1	1	1	1
New order	177	209	189	402	272	496
New order rollback	186	271	176	299	248	427

TABLE II
PERFORMANCE ANALYSIS FOR INDIVIDUAL DACAPO BANK TRANSACTIONS, THIS TABLE SHOWS THE EXECUTION TIME (MS) FOR EACH PERSISTENCE FRAMEWORK IN EACH TRANSACTION TYPE

DaCapo	LOC
ActiveAndroid	2923
greenDAO	2200
OrmLite	3310
Sugar ORM	2911
Android SQLite	3068
Java Realm	3071

TABLE III
LOC FOR DACAPO BENCHMARK

ACKNOWLEDGMENT

This research is supported by the National Science Foundation through the Grant CCF-1116565.

REFERENCES

[1] "Smartphone os market share, 2015 q2." [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[2] "Statista: Mobile apps available in leading stores (july 2015)." [Online]. Available: http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/
[3] M. Dong and L. Zhong, "Sesame: Self-constructive system energy modeling for battery-powered mobile systems," *arXiv preprint arXiv:1012.2831*, 2010.
[4] M. Cohen, H. S. Zhu, E. E. Senem, and Y. D. Liu, "Energy types," in *ACM SIGPLAN Notices*, vol. 47, no. 10. ACM, 2012, pp. 831–850.
[5] S. Hao, D. Li, W. G. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 92–101.
[6] Y.-W. Kwon and E. Tilevich, "Reducing the energy consumption of mobile applications behind the scenes," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 170–179.
[7] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 22–31.
[8] S. Mukherjee and I. Mondal, "Future practicability of Android application development with new Android libraries and frameworks," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 4, pp. 5575–5579, 2014.
[9] "Sqlite database engine." [Online]. Available: https://www.sqlite.org/
[10] "Realm database engine." [Online]. Available: https://realm.io/
[11] "Activeandroid by pardom." [Online]. Available: http://www.activeandroid.com/
[12] "greenDAO: the superfast Android ORM for SQLite." [Online]. Available: http://greenrobot.org/greendao/
[13] "Ormlite - lightweight object relational mapping (ORM) Java package." [Online]. Available: http://greenrobot.org/greendao/
[14] "Sugar ORM - insanely easy way to work with Android databases." [Online]. Available: http://satyan.github.io/sugar/
[15] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer *et al.*, "The DaCapo benchmarks: Java benchmarking development and analysis," in *ACM Sigplan Notices*, vol. 41, no. 10. ACM, 2006, pp. 169–190.
[16] C. Xia, G. Yu, and M. Tang, "Efficient implement of ORM (object/relational mapping) use in J2EE framework: Hibernate," in *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*. IEEE, 2009, pp. 1–3.
[17] A. Vetro, L. Ardito, G. Procaccianti, and M. Morisio, "Definition, implementation and validation of energy code smells: an exploratory study on an embedded system," 2013.
[18] "Monsoon power monitor." [Online]. Available: https://www.msoon.com/LabEquipment/PowerMonitor/