Internship Report

David Matos and Tiago Brito

Instituto Superior Técnico, University of Lisbon, PT,
david.r.matos@tecnico.ulisboa.pt,
tiago.de.oliveira.brito@tecnico.ulisboa.pt

Abstract. File systems have been used in modern operating systems to control how data is stored and retrieved from physical storage devices. Since then, file systems have been the focus of several performance and structural enhancements by the research community as they have a high impact on system performance. Along side file systems, database management systems have also been an integral part of modern OS. During our internship we worked with SQLite, a database management system built on top of the file system abstraction, rather then beside it. The goal of our work was to create a workload for a real-world application which uses SQLite transactions. In this document we describe the development processes and results achieved during the intership.

1 Introduction

This report describes the work developed during our five week internship at the University of Texas at Austin, under the supervision of professor Emmett Witchel [1]. The goal of our work is to help current UTA researchers in their research projects. Upon our arrival, professor Emmett Witchel presented us several different projects on which our help would be useful. Of these different projects, two stood out as the most interesting projects for both of us.

The first project involved adapting a kernel level utility, such as the file system, and build it as a user library inside an application. Theoretically this has several advantages, such as portability and performance improvements. This is because the file system code becomes independent from the kernel code, and thus can be executed at the user level rather than kernel level. This first project was the one we wanted to pursue, but since this is a kernel development project, which is generally complex and time consuming, we chose the second project in order to complete the project in due time (five weeks).

The second project involved building a realistic workload for applications using the SQLite database management system. The goal of this project is to have a realistic scenario for testing an SQLite client. By building an easily reproducible workload for an application using SQLite we allow researchers to test different versions of SQLite in the same scenario and evaluate the performance differences. Since this project seemed more achievable during the timespan of this internship it was the project chosen by us.

The second project was then developed during the following weeks of the internship. During this period we presented intermediate versions of the prototypes to professor Emmett's students which were responsible for the project. In these meetings we presented research work done on related work, ideas and prototype implementations of the work done so far. The development process, including the specifics of our project choice and the steps taken during the development of this projects, will be described in the following section of this report (section 2). In section 3 we will present the prototypes developed during the internship and finally, in section 4 we give an general overview of our internship and experience gained during our stay at the University of Texas at Austin.

2 Development Process

Upon our arrival at the University of Texas at Austin, we met with professor Emmett Witchel. Professor Emmett is our program mentor and, as such, he was responsible for assigning us a project to be developed during our five week internship. During our first and only meeting with professor Emmett, he showed interest in knowing more about previous research work developed by us at Instituto Superior Técnico and about our research interests. Additionally, professor Emmett presented us some of his own research interests. It became clear that professor Emmett Witchel has a deep interest and knowledge of operating systems and specially in the field of file system and transaction research.

Similarly, our interests and research experience is on operating systems and distributed systems, although our focus is on the security aspect of these systems. This was good because it allowed professor Emmett to establish common ground between his students' research projects and our research experience. With the common ground established, professor Emmett described us the current research projects being developed by their students and which projects he believed would gain from our cooperation.

From the projects suggested by professor Emmett two stood out as being more interesting for the both of us. These were the projects which more closely resembled our research experience and were the projects that professor Emmett was convinced could enjoy our help the most. Professor Emmett than proceeded to introduce us to his research students responsible for the projects suggested. Since not all of professor Emmett's students were at the university during this meeting, we were only introduced to the students responsible for one of the projects that stood out for us.

The students responsible for this first project introduced and explained us the concepts and motivation behind their project as well as how our cooperation could help them achieve their project's goals. As explained in section 1 this first project involved adapting a kernel level utility, such as the file system, and build it as a user library inside an application. This first project was the one we wanted to pursue as it would allow us to improve our knowledge of kernel level development and specially of common file system utilities such as the EXT4 file system. But because kernel development projects are generally very complex, and because we have limited experience in such projects, we believed the project might not be viable for our limited timespan of five weeks. In addition, professor Emmett was not convinced with the student's approach to the project, thus lowering our confidence on this project.

The second project involved building a realistic workload for applications using the SQLite database management system. Although we were not introduced to the students responsible for this research project, the work suggested for us to do for this project during these five weeks seemed more achievable than the first project. But because we were interested in improving our kernel development skills, we had doubts about choosing this project in regards to the other one.

After being introduced to the suggested projects and the students responsible for these projects professor Emmett Witchel finished the meeting by giving us a tour of the laboratory and suggesting a place to work during the week days. After this, professor Emmett suggested us to think about the projects until the end of that week and answer with which project we wanted to cooperate in. In the mean time, professor Emmett suggested us to study the first class of his *Introduction to Operating Systems* course, which showed us how to prepare a kernel development environment, for the first project. The following subsections describe our approach to selecting one of the two projects mentioned before.

2.1 Project Selection

After the first and only meeting with professor Emmett Witchel we started researching projects similar to what was described during the meeting. In addition, we sent emails to the people responsible for both projects in order to gather more information and clarify some project specific questions. The students responsible for the second project seemed more interesting in our cooperation and sent us two articles [2, 4] of related work so we could familiarize ourselves with the project.

In addition to reading those papers, we also studied the first class of professor Emmett's *Introduction to Operating Systems* course, although we would eventually choose the second project, thus rendering this class unnecessary for the conclusion of the project. Given the high interest by the students responsible for the second project, together with complexity inherent to the first (kernel development) project, we decided to choose the second project to be developed during our internship.

Consequently, we contacted Yige Hu and Vijay Chidambaram (professor Emmett's students) to confirm our interest in cooperating in their research project during our stay at the University of Texas at Austin. As a response to our email, Yige and Vijay arranged a meeting via Skype, an application that provides video chat and voice call services. The meeting was arranged in this way because both Yige and Vijay were not at the university during the period of our internship. Because these students were not at the university during our internship, we never felt the need to work directly on professor Emmett's laboratory. Our project was done comfortably using our own personal computers and in San Jacinto Residence Hall.

The meeting had the goal to clarify project specific questions and to guide us towards the right path of development for their workloads. In addition, this meeting was used to schedule consequent meetings to update Yige and Vijay on the progress of our work. The following subsection describes the meetings and development process we followed during the remaining four weeks.

2.2 Development Steps

First Meeting: the goal of the first meeting was to describe the problem to be solved by us during our internship. The problem was described as the difficulty in testing realistic SQLite workloads consistently. So, in order to overcome this difficulty, we our proposed to build a reproducible workload based on realistic applications, such as email clients, web browsers and social networking application, which use SQLite. Since we were both new to SQLite and particularly in benchmarking SQLite performance we were proposed to research the literature and come up with an approach for the development of the workloads.

First week: as proposed in the first meeting, the following week was used to research related work regarding SQLite benchmarking and SQLite workloads. Our attention was focused on finding workloads which would represent realistic scenarios, and consequently realistic SQLite use. While researching for these realistic workloads and scenarios, we came across several articles regarding the trace and analysis of Input/Output characteristics in file systems where SQLite is used and analyzed [5–7].

This first article [5] studies the IO characteristics of SQLite transaction in Android platform. The authors collected block level IO traces for six months and developed a pattern matching algorithm which allows them to identify the individual SQLite transactions from the raw IO trace. This article highlights three key findings, which are as follows. First, SQLite transactions are extremely inefficiency since over 75% of the entire IO volume in a SQLite transaction come from SQLite journaling and EXT4 filesystem journaling. Second, the suspend and the wakeup feature of the smartphone can leave the SQLite transaction to an extreme delay, a few minutes. Third, SQLite transactions are not used efficiently by application programmers. Although this article describes preliminary solutions to these problems, we are not interested in these solutions but rather on the process of analysis and conclusions of the initial study.

The second article [6] describes an IO trace and analysis framework, Androtrace, developed for the Android platform. Androtrace's novelty is how it efficiently traces IO operations and how it supports the analysis of multiple Android users by using a server and client model. The authors conclude, through the use of the framework, that write IOs dominate the workload of mobile applications. This is interesting because it gives us an insight on what realistic IO workloads might look like. On one hand, based on this analysis, one can say that realistic workloads are write IO intensive. But on the other hand, Androtrace does not specify the origin of these write IO operations, meaning that these writes may be from sources other than SQLite transactions. For this reason, the information from this article builds up on previously acquired information, but does not describe realistic SQLite workloads and as such is not sufficient for us.

Similarly, the third article [7] offers an in-depth IO characterization of an Android smartphone. The authors examined the correlations between seven IO attributes: originating application, file type, IO size, IO type (read/write), random/sequential, block semantics (Data/Metadata/Journal), and session type (buffered vs. synchronous IO). The authors also developed Mobile Storage Analyzer (MOST) [8], a framework for collecting IO attributes across software layers. In the underlying study done for this article, the authors noticed that SQLite puts too much burden on the storage. For example, a single SQLite operation results in at least 11 write operations being sent to the storage. As for storage, more than 50% of writes are for EXT4 Journal updating and 70% of all writes account for small data writes, about 4KB. Because in Android the SQLite and EXT4 filesystem require great effort to ensure reliability for transactions and journaling, respectively. The operations of SQLite and EXT4, when combined, generate unnecessarily excessive write operations to the storage device.

The authors conclude that this not only degrades IO performance but also significantly reduces the lifetime of the underlying NAND flash storage. This article is revealing for our work because the authors developed IO profiles for six different application categories, such as Web, Contact, Social Networking, Multimedia, System and Game. Namely this article profiled custom scenarios for specific well-known applications which fit the mentioned categories. These profiles allow us to understand how realistic workloads are partitioned regarding SQLite reads and writes. With this understanding we can implement a benchmarking tool which simulates these applications by leveraging SQLite reads and writes ratios.

Second Meeting: Before the second meeting we sent an email to Yige Hu and Vijay Chidambaram describing our research work during the week. During the meeting we talked about our findings and discussed the idea of developing a script leveraging Mobibench [9] to simulate the profiles described before. Mobibench is a tool which allows the user to specify several SQLite parameters to be tested. Mobibench then executes the benchmarking using the specified parameters. These parameters can be adjusted as to closely resemble the profiles discussed above. This would allow researchers to easily benchmark different realistic workloads. The idea was well received and as such we proposed to have a raw prototype of the mentioned script before the following meeting.

Second Week: During this second week we focused our efforts on developing a prototype of the benchmarking script proposed to Yige and Vijay. We created a *BASH* script which allows the user to easily specify which of the mentioned profiles he wants to benchmark. Additionally the script prepares the testing environment by installing Mobibench. By the end of this week we achieved in having a working version of this script which would benchmark the six different profiles using the Mobibench tool.

Third Meeting: This third meeting was used to show Yige and Vijay what was accomplished during the week. Unfortunately this meeting was not attended by both of professor Emmett's students since only Yige was available at the time of the meeting. We were still glad to be able to have this meeting in order to clarify questions regarding the next steps to take on this project. We explained to Yige what the script was capable of achieving and explained what was the goal of this script.

Yige seemed to understand and support the work done so far and suggested the addition of a feature to the script. The suggestion was to allow the user of the script to manually adjust the read/write ratio of SQLite transactions tested by Mobibench. We argued that adding this feature would defeat the purpose of having profiles of realistic workloads. But Yige felt that adding this feature would improve the usability of the script, so we added that to our to-do list.

We then asked Yige if they were interested in having our script to automatically average the results of running several tests of the same profile and also create graphs to compare the different results. She agreed to this feature and suggested us to test how many tests were needed to have a small standard deviation during the benchmarks. We scheduled the next meeting to the same day and time next week and proposed to have the new features implemented until then.

Third Week: During this third week we tried to implement the new features discussed during the previous meeting, with Yige. Originally, our *BASH* script allowed the user to run one benchmark for a specific profile. The result would be the specific measures of the benchmark, which are described in full detail in section 3. But, to support the new feature, which would allow the user to run a battery of benchmarking tests on a specific profile, we had to find a way to run our original script a predetermined amount of times, gather all the results, interpret these results by taking the average and standard deviation and finally produce the graphs which represent these results.

In order to avoid re-doing the original script we planned on creating a new script which would leverage the original script in order to produce the intended output. We thought about using *Python* as the scripting language for the new script because it is a flexible programming language and supports libraries for creating graphs. But because leveraging several scripts with different programming languages is unpractical we reconsidered writing our original script again, this time using *Python*. Since the only difference between the scripts is the programming language's syntax, re-writing the original script introduced little overhead to our project.

So, during this week we re-implemented our original script with *Python* and started to implement the support for the graph representation and to calculate the average of several benchmarks of one profile.

Forth Meeting: This meeting was again attended by both Yige and Vijay. We described the state of our work and discussed what was discussed during the previous meeting, at the time with Yige only. Unexpectedly, Vijay told us that what we were doing was not what they wanted. Because this was our forth meeting we were surprised by this remarked, but listened to what Vijay was suggesting to do. It became clear that there was a lack of understanding between both parties regarding what the goals of the project were. We understood they wanted to benchmark realistic SQLite workloads, but they failed to clarify how these workloads should have been built.

In previous meetings our suggestion of using Mobibench to simulate workloads was well received, but it seems the goal was not to use simulated workloads from a benchmarking tool such as Mobibench but instead use real-world SQLite queries. At this point we realized that most of our work done so far was rendered useless for the newly perceived goals of the project. Even so, Yige and Vijay suggested us to still deliver the script implemented until then, since it may be useful later on. In order to recover lost time we suggested analyzing a real-world SQLite application and use it as the desired workload. That work was done during the following week.

Forth Week: In order to build a realistic SQLite workload using real SQLite queries we had to study real applications. For time constraints we chose to analyze only one application corresponding to one of the profiles described above. We searched for open-source applications in order to have access to the SQLite queries executed by the workload. After extensive search of well-known open-source Android applications we chose K9mail [10], a popular Android email client application.

After inspecting the source code we noticed that all SQLite queries were triggered by one *Java* class. This allowed us to implement a wrapper class which replaces the original SQLite queries class and records all queries executed while the application is running. After implementing this wrapper class we proposed a scenario to record a realistic workload for our project. The scenario we proposed was opening the application, sending three emails and closing the application. This scenario is based on a specific scenario presented in the same article where the profiles are first described.

The idea is to record all the queries executed during this scenario so they can later be reproduced for benchmarking purposes. By reproducing the queries, instead of simulating the workload with Mobibench, researchers can test the same workload and interpret the performance differences between using different SQLite and file system implementations.

Fifth and Final Meeting: The final meeting was, once again, with Yige only since Vijay could not attend again. We described our progress to Yige and tried to clarify how our progresses can help in achieving the goal of the project. Yige understood our approach and how this could be used to build the realistic workload needed for benchmarking their project. On one hand, our approach works, on the other hand, Yige felt like we could have accomplished the same result with a more clever approuch. Considering that using a different, more generic approuch would be much better for adapting different applications to usable workloads. This is because writing wrapper classes to all applications we wish to use as a workload is unpratical, but if we took that approach on the last week, we would risk not having results.

The following section describes implementation details regarding the different solutions implemented during our internship.

3 Prototypes

During the internship we developed two prototypes to evaluate the throughput of SQLite [3] databases. At first it was asked to develop a workbench that would execute realistic SQLite workloads into a database. However, after we finished the first prototype we were informed that it wasn't useful so they asked us to develop a new prototype that would gather a real world workload of a database.

3.1 Prototype 1

The first prototype was a workbench to evaluate the throughout of several workloads in a real SQLite database. To do so we based our work in an existing SQLite workbench called MobiBench [9]. Mobibench is a CLI tool that allows to measure IO performance and SQLite of Android filesystem. Mobibench is highly configurable, allowing to adjust several parameters of a workload. The configurable parameters are:

- **p** set path name
- **f** set file size in KBytes
- **r** set record size in KBytes
- **a** set access mode
- **y** set sync mode
- **t** set number of thread for test
- **d** enable DB test mode
- **n** set number of DB transaction
- **j** set SQLite journal mode
- **s** set SQLite synchronous mode
- **g** set replay script
- **q** do not display progress

The problem with MobiBench is that it is necessary to give as input the exact parameters of a workload. In order to simulate the behavior of real world applications we need to adjust these parameters to match the workloads generated by real applications. The workload of each application varies depending on the application features. For instance, an email client app will generate several read operations (gathering the inbox messages every time a user opens the app) and less write operations (every time an email is received it must be stored in the inbox). A social network application (for example: Facebook or Twitter) will generate more write operations, since every post or tweet must be stored in local storage, and there are more posts and tweets being generated than emails.

So, in order to generate workloads in MobiBench that represent real world applications we need to know the number of read and write operations generate by each applications, as well as the size of these operations and frequency. After researching several articles we discovery two [2, 4] that describe in detail the workloads of popular applications. Using the numbers in these articles we developed a script that given a profile (a category of an application, for instance, email) executes the respective workloads in MobiBench. With our script it is possible to simulate a real application in MobiBench giving only a category of an application instead of setting manually all the parameters of MobiBench. Besides running realistic workloads, our script also install the latest version of MobiBench by using the -i option.

The script was developed in bash and takes as input the following parameters:

- **h** Prints to stdout the usage of this script;
- **i** Git clones and compiles the latest version of MobiBench;
- **b** Starts a benchmarking evaluation given a profile. The available profiles are:
  - **web:** web browser opening a couple of pages and performing a single operation in each page;
  - **sns:** social network application. Simulates 10 scroll down operations in the wall;
  - **multimedia:** a video application. Simulated the user opening a video playing it for a couple of seconds and closing the video;
  - **system:** simulates a system application that runs in background constantly;
  - **game:** simulates a game.

After we finished the first version of this script, in the third week, we had a meeting with Yige Hu and Vijay Chidambaram. In this meeting we presented our work and asked for improvements. However, they told us that although the tool is useful it does not give a reliable evaluation of real world applications, since the numbers used in the script were based in an old article and not in real operations. It was important to have the exact executed queries by each application, instead of the total number of read and write operations.

3.2 Prototype 2

With only a week and a half left for our internship we propose a second prototype. Our proposition consisted in a database wrapper, which is a layer between the application and the database that is responsible for intercepting every query and register a log. With this wrapper it is possible to collect the exact queries executed by each application. This way a user may simulate a behavior in an application and then consult the log with executed queries to later reproduce the interactions. Since we only had a week and half until our return we wrote the description of this prototype and waited for confirmation before developing. A day later Yige Hu confirmed our project and we started developing it.

We based this prototype on the application K9 Mail [10]. K9 Mail is a well known open source e-mail client for Android that uses SQLite to store information about the user account and the messages. Figure 3.2 represents how K9 Mail interacts with SQLite. On the left side of the figure K9 Mail executes queries (1) in the SQLite database and gets the results (2). On the right side of the figure we included our Database Wrapper and Log components. In practice the database wrapper consists of a Java class that substitutes the standard SQLiteDatabase class of SQLite. So, instead of interacting directly with the database, K9 Mail interacts with the Database Wrapper (1) which registers every query to a Log file (2) and then redirects the query to the database (3), finally the result of the query is returned to the Database wrapper (4) which is forwarded to the K9 Mail application (5). There is no difference, from the user perspective, in the application. Everything works as it did before introducing the wrapper. After interacting with the application for a short period it is possible to consult all the executed operations in the database.

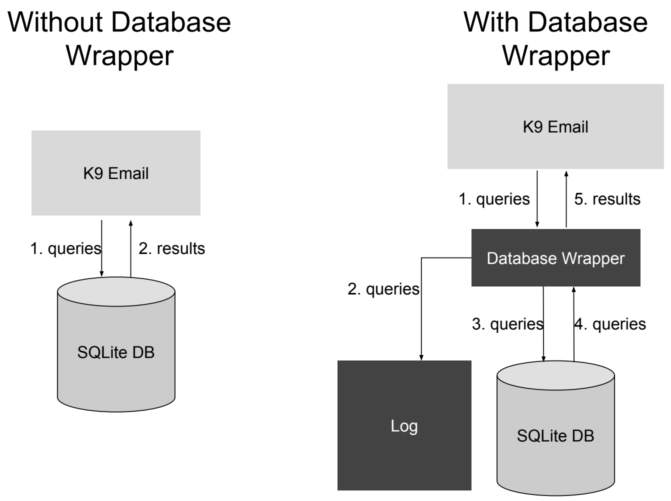
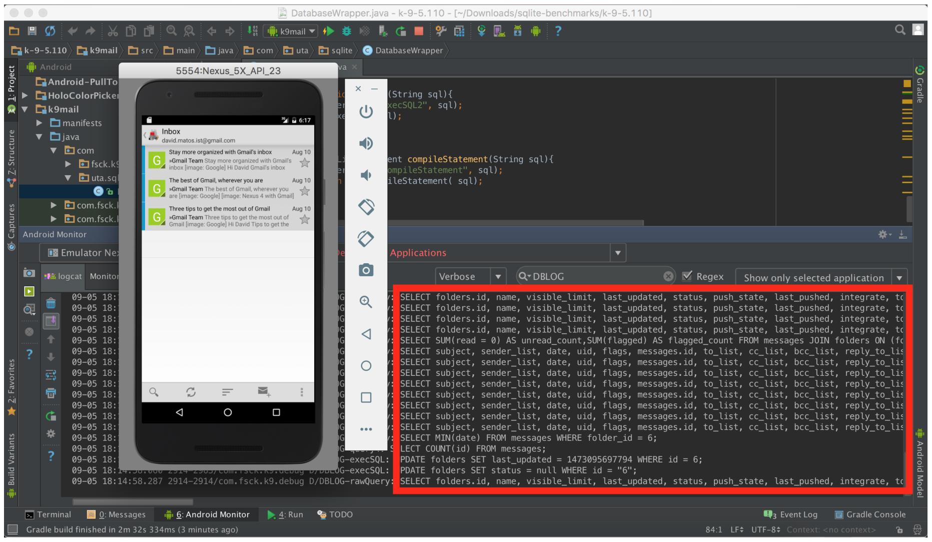


Fig. 1. Flow diagram of K9 Email without Database Wrapper and with Database Wrapper

Figure 3.2 shows the K9 Mail application running in the Android emulator. The red square marks the log with every SQLite query executed so far. With this prototype it is possible to gather real workloads of a real application (K9 Mail). The prototype allows any execution scenario to be tested. For instance, it is possible to collect the SQLite operations of a simple user interaction (open account, read an email and send an email) as well as testing a complex scenario of execution (reading several emails, responding to old emails, deleting emails and performing searches). By using this prototype it is possible to simulate any user interaction and it is simple collect SQLite operations of different applications. All that needs to be done is import our Database Wrapper in any Android application.



```
09-05 18:                                    SELECT folders.id, name, visible_limit, last_updated, status, push_state, last_pushed, integrate, to
09-05 18:                                    SELECT folders.id, name, visible_limit, last_updated, status, push_state, last_pushed, integrate, to
09-05 18:                                    SELECT folders.id, name, visible_limit, last_updated, status, push_state, last_pushed, integrate, to
09-05 18:                                    SELECT folders.id, name, visible_limit, last_updated, status, push_state, last_pushed, integrate, to
09-05 18:                                    SELECT SUM(read = 0) AS unread_count,SUM(flagged) AS flagged_count FROM messages JOIN folders ON (fo
09-05 18:                                    SELECT subject, sender_list, date, uid, flags, messages.id, to_list, cc_list, bcc_list, reply_to_lis
09-05 18:                                    SELECT subject, sender_list, date, uid, flags, messages.id, to_list, cc_list, bcc_list, reply_to_lis
09-05 18:                                    SELECT subject, sender_list, date, uid, flags, messages.id, to_list, cc_list, bcc_list, reply_to_lis
09-05 18:                                    SELECT subject, sender_list, date, uid, flags, messages.id, to_list, cc_list, bcc_list, reply_to_lis
09-05 18:                                    SELECT subject, sender_list, date, uid, flags, messages.id, to_list, cc_list, bcc_list, reply_to_lis
09-05 18:                                    SELECT subject, sender_list, date, uid, flags, messages.id, to_list, cc_list, bcc_list, reply_to_lis
09-05 18:                                    SELECT subject, sender_list, date, uid, flags, messages.id, to_list, cc_list, bcc_list, reply_to_lis
09-05 18:                                    SELECT subject, sender_list, date, uid, flags, messages.id, to_list, cc_list, bcc_list, reply_to_lis
09-05 18:                                    SELECT MIN(date) FROM messages WHERE folder_id = 6;
09-05 18:                                    LECT COUNT(id) FROM messages;
09-05 18:                          G-execSQL: PDATE folders SET last_updated = 1473095697794 WHERE id = 6;
09-05 18:                          G-execSQL: PDATE folders SET status = null WHERE id = "6";
09-05 18:14:58.287 2914-2914/com.fsck.k9.debug D/DBLOG-rawQuery: SELECT folders.id, name, visible_limit, last_updated, status, push_state, last_pushed, integrate, to
```

Fig. 2. K9 Mail application running in the emulator and log with database operations.

4 Conclusion

During 5 weeks we study and develop prototypes to simulate real world applications in SQLite databases. We manage to complete two functional prototypes and present a state of the art of SQLite in Android. Both prototypes are capable of producing a realistic experimental evaluation of Android applications using SQLite. The first prototype calculates the throughput of a database with different applications. With the second prototype it is possible to collect the exact queries that were executed by an Android application. We also provided an example of an application (K9 Mail) using the second prototype.

We are satisfied with our work and believe that we achieve our goals for this internship. However, the fact that 5 weeks if a short period of time to develop a significant research project and present useful results, it would be more useful if we had precise directions, in terms of goals of our project and a better support from our mentors. In the end, we felt that we ended up wasting time researching and developing tools that were not exactly what was needed. It would be beneficial if we had the topic and goals of our project and before going to the internship. This way we could get ahead and start working in sooner and perhaps present a better work.

In conclusion, the internship was a very fulfilling experience. We got the chance to know and work with different people and learn interesting topics that are not related with our research fields. Five weeks is a short period to develop a relevant project nevertheless we were able to produce useful tools and contribute to their work.

References

1. Emmett Witchel, PhD. Web site: https://www.cs.utexas.edu/users/witchel/
2. Ji, Cheng, et al. "An Empirical Study of File-System Fragmentation in Mobile Storage Systems." 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16). 2016.
3. Owens, Mike, and Grant Allen. "SQLite". Apress LP, 2010.
4. Luo, Hao, Hong Jiang, and Myra B. Cohen. "Why Do We Always Blame The Storage Stack?." 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16). 2016.
5. Tuan, Dam Quang, Seungyong Cheon, and Youjip Won. "On the IO characteristics of the SQLite Transactions."
6. Lim, Eunryoung, Seongjin Lee, and Youjip Won. "Androtrace: framework for tracing and analyzing IOs on Android." Proceedings of the 3rd Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads. ACM, 2015.
7. Lee, Kisung, and Youjip Won. "Smart layers and dumb result: IO characterization of an android-based smartphone." Proceedings of the tenth ACM international conference on Embedded software. ACM, 2012.
8. Lee, Kisung, and Youjip Won. Mobile Storage Analyzer: https://github.com/ESOS-Lab/MOST
9. Mobibench Web site: https://github.com/ESOS-Lab/Mobibench
10. Jesse Vincent. Advanced Email for Android: http://k9mail.github.io/