PocketBench:
A Framework for Small Data Benchmark Design

Gokhan Kul and Gourab Mitra
University at Buffalo, SUNY
{gokhanku, gourabmi}@buffalo.edu

ABSTRACT

Mobile databases are the statutory backbones of many applications on smartphones. Their performance very much depends on the performance of the underlying databases. However, these databases and the querying engines in the applications are usually uncontrolled, not properly designed and not tuned for optimal performance. We take the initiative to analyze mobile database logs to investigate the interaction between the application and the database to model the application characteristics. Although various techniques have already been produced for database log exploration, they target enterprise environments where the data is accessed from many different machines and by many different users. On Android phones, on the other hand, the database is exclusive to one application, which means, understanding the log can lead to understanding the application, hence, allowing for opportunities to improve the performance considerably. In this paper, we introduce the first steps of a framework to create a benchmarking tool which aims to emulate the workloads of Android applications to compare different mobile database management system implementations. We first describe the PocketData dataset while pointing out the details that we exploit. We then propose a clustering scheme where we analyze the query logs to identify and group the SQL queries with similar interests together. We show experimentally that the clustering scheme is able to categorize queries with similar interests together. Finally, we elaborate on using these clusters to model common behaviors and unusual patterns. We believe these common patterns can be used to realistically emulate synthetic workloads created by Android applications, allowing to test the performance of different mobile database management systems. Another possible usage of this system is to identify unnecessarily repeating patterns; which can be an indicator of bugs in the source code, hence, allowing the application developers to solve problems that do not create the application to crash but reducing performance.

Keywords

Benchmark, Database, Workload, Mobile Systems

1. INTRODUCTION

Smartphones we so naturally carry and use today do not have a long history. The smartphones as we know them today started to get used worldwide by average people around 2007. Of course, there were earlier representatives of smartphones, but most of them did not reach to the mass crowds due to their price and lack of network coverage. This fundamental shift in technology pushed phone-makers into adopting their technology into the new trends as fast as possible, to be able to stay in the market. Once world leaders Nokia and Blackberry rapidly lost their market shares due to failing to adapt to the new trends [1, 2]. However, these developments led to a chaotic environment, where creating clear and consistent standards got disregarded.

One of the areas affected by this environment was how the data is stored on these devices. The data can be structured or unstructured, and the data storage methodologies got adopted from computers which had a lot more processing power and available memory. Although the processing power and memory barriers are fading away with the current technology in the smartphones, the applications still depend on either file-based storages like JSON and CSV or embedded SQL database systems like SQLite [3,4]. Although there is a limited number of choices for database management systems available for smartphones, we anticipate the release of alternative systems soon.

In this paper, we introduce PocketBench, a framework to create a benchmarking tool which aims to emulate the workloads of Android applications to compare different mobile database management system implementations. Utilizing Android query logs, we model common behaviors and unusual patterns which can be used to realistically emulate synthetic workloads created by Android applications, allowing to test the performance of different mobile database management systems.

Scenario 1. *Charles, a Mobile Systems Product Manager at Facebook wants to find a way to improve the performance of their mobile applications and decides to find the performance bottlenecks and performance improvement opportunities. Setting up PocketBench, his team can use the query logs of alpha test users and find out if the current DBMS system in use is better or worse than other alternatives.*

We also argue that many apps could benefit from understanding how the user uses that particular app. For example,

Scenario 2. *Bob, an Android photographer, may be using Instagram to post a lot of pictures for the brands, hotels and touristic places. Alice, on the other hand, may be using Instagram for browsing photos from the users she follows, and may not have the habit of posting too many photos.*

The workload these two people in the scenario create on the local mobile database is different and should be addressed accordingly. We can utilize this usage characteristics information to (1) increase performance for various workloads, (2) find out bugs and unnecessary database calls in the apps, (3) give more accurate recommendations to the user, and (4) explore the data flow improvement opportunities within the app.

Concretely, in this paper we: (1) motivate for a mobile database benchmark, (2) utilize query similarity metrics to find similar queries by structural similarity, (3) analyze the vectors and similarity matrix of the query load to create clusters of structurally similar queries, and (4) introduce techniques for exploring repeating usage patterns and unusual usage characteristics. We finally experimentally demonstrate that our methods are able to infer usage characteristics of users for specified Android applications.

Note that although we motivate for creating a benchmark in this paper, developing the data and query emulation step is not in the scope of this work.

This paper is organized as follows. We first describe the motivation and give background reasoning in Section 2. Then, we introduce a sample dataset for workload characterization and explain methods we utilize in Section 3. In Section 4, we evaluate the accuracy and performance of our proposed techniques. Finally, we conclude in Section 5 and identify the steps needed to deploy our methods into practice in Section 6.

2. BACKGROUND AND MOTIVATION

The lack of standards and the need for a better understanding of mobile storage systems can easily be seen by surveying through standardized and well-known mobile database system benchmarks, which in fact non-existent [5], while traditional database systems have a few dependable benchmarks [6, 7]. Also, traditional database systems are usually managed by professional database administrators who tune-up the databases according to changing workloads while smartphone databases work with predetermined indexes and are not subject to tuning up depending on the workload they are experiencing. Although there were some efforts to measure the performance of SQLite and Android devices under different workloads [8], these benchmarks do not specify the bottlenecks, how and where the tune ups should be performed or they do not provide any information specific to the app performance.

While the most visible parameter of data is its volume, it is not the only characteristic that matters. In fact, there are four key characteristics about data [9]:

- **Volume.** The most intuitive characteristic about data is the amount of data itself. In fact, it is the sheer amount of data that we generate and process these days that calls for a better approach to data management. It is one of the driving forces behind this work.
- **Velocity.** Velocity refers to the idea of the amount of data flowing through an interface in unit time.
- **Variety.** Traditional data formats were relatively well defined by a data schema and changed slowly. In contrast, modern data formats change at a dizzying rate. This is referred as variety in data.
- **Value.** The value of data varies significantly. The challenge in drawing insights from data is identifying what is valuable and then transforming and extracting the data for analysis.

Database servers and web applications experience a workload that is not typical in smartphones. The majority of these servers form the backbone of an application. In most cases, they store the business data to support OLTP and OLAP operations. The data volumes may range from medium to high amounts for systems with a large concurrent user base. The data velocity also grows in proportion to the number of concurrent users.

The usage pattern of databases in smartphones differs significantly from the above-mentioned ideas. Most modern day smartphones rely on some kind of a web service to help a mobile application deliver the desired functionality to the user. This introduces various new application design considerations. Mobile users must be able to work without a network connection due to poor or nonexistent connection. In that case, a mobile database serves as a cache to hold recently accessed data and transactions so that they are not lost due to connection failure. In many cases, users might not expect live data during connection failures; only recently modified data. Update of recent changes and downloading of live data can be deferred until connection is restored.

Mobile computing devices tend to have slower CPUs and limited battery life. The luxury of having a cluster of powerful computers to deploy a database is just not there. Also, the fact that battery power is scarce drives the case further to achieve high resource utilization.

It is a common practice among smartphone users to have multiple devices. Most smartphones have an authentication system which is powered by an email account which is accessed in other devices too. In most cases, the user has at least one more device or a web service that needs to sync with the smartphone. This leads to occasional synchronization activities that occur between different devices and a centralized data store. Often, this activity happens in the background so that the user is not blocked from using other functions on a smartphone.

The PocketData dataset [5] provides handset-based data directly collected from smartphones for multiple users. User sessions in context of smartphones might not be similar to a session on other more traditional computing devices like PCs. Typically, an end user would use their smartphones in multiple small intervals of time through a day. These 'bursts of activity' can be referred to as session. In context of a single mobile application, the user would access multiple logical transactions in these bursts of activity. Intuitively, a user session is quite straightforward to understand, but its technical aspects require defining. Some smartphone usage studies define a session as the time period where the smartphone's screen is active [10]. Smartphone usage is dominated by usage of the applications that the smartphone has to offer. Thus, the idea of a smartphone usage session can be reduced to an application usage session. This is relevant to us because we aim to study the interaction with the smartphone database through understanding a single application.

3. METHODOLOGY

We propose a heuristic to analyze query logs and find out interesting patterns. This process has three steps:

1. Figuring out the activities of interest with respect to the application
2. Clustering the queries
    (a) Extracting features
    (b) Query comparison
    (c) Clustering with different strategies
3. Detecting patterns in user activity
    (a) Appoint an integer label to each cluster
    (b) Identify which cluster a new-coming query belongs to
    (c) Identify patterns with different strategies

The strategies for applying these steps are given in this section.

3.1 Dataset

As a sample dataset, we use PocketData [5] dataset. This dataset includes one month's trace of SQLite activity on 11 PhoneLab [11] smartphones running the Android smartphone platform.

This dataset consists of various information about the usage patterns across a wide variety of apps. Each line in the log has:

- Device ID: Unique identifier for each device
- UNIX timestamp: Milliseconds since 1970
- Ordering: Timestamp and request order
- Date and time: Human readable timestamp
- Process ID: Standard UNIX process ID
- Thread ID: Standard UNIX thread ID
- Log level: Verbose (V), Debug (D), Info (I), Warning (W), Error (E)
- Tag: Source of log information, "SQLite-Query-PhoneLab"
- JSON object that holds various information about the event that is logged

information.

Note that the app ID is not included in the log, because different apps can have the same process and thread IDs in different times. Our strategy to get the log lines for our app of interest is to search for the app name in JSON object parsing from the beginning of the file. When we first encounter the app of interest, we use process and thread IDs to identify the events related to that app until we encounter a different app name in the JSON object.

Privacy Concerns: PocketData dataset is a best-effort anonymized dataset. Most of the private information is irreversibly concealed but there are still some constants in the queries. Hence, we implement all the functions of the system with Java for it to be repackaged and be able to be imported into mobile phones as well as servers. This may solve the privacy concerns of the users: allowing the processing to be performed on the phone instead of a company server even if it is not the ideal case due to performance and energy consumption constraints.

3.2 Clustering

The main goal of this step is to group queries into classes that exhibit similar 'intent'. Clustering the query workload narrows down the space of possible patterns that could be detected. This facilitates easier and more accurate understanding of query workload [12]. In the clustering process, we first filter the activities belonging to the app of our interest without distinguishing which user the activity belongs to. Then, we create clusters using all the activities belonging to that specific app. The workflow is illustrated in Figure 1.

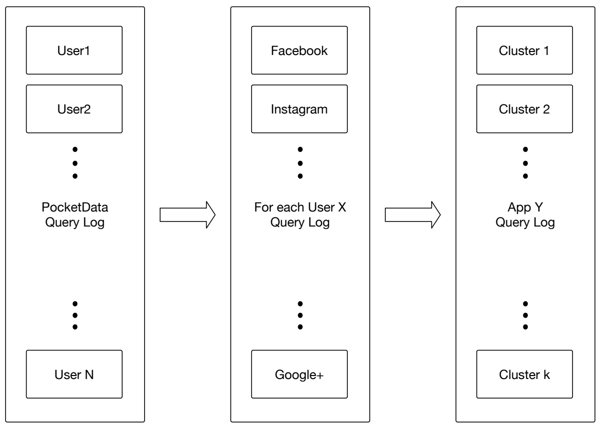

Figure 1: The workflow for clustering process

To achieve this we need to be able to extract features out of SQL queries, compare them and compute their similarity. Extracting features from a SQL query, can be done in many ways. Let's consider the following queries:

```
Q1: SELECT username FROM user WHERE rank = "admin"

Q2: SELECT rank, count(*) FROM user
    WHERE rank <> "admin" GROUP BY rank
```

These two queries share many attributes and seem to be working on similar concepts although not performing semantically very similar tasks. Usually, what we consider important in a query can roughly be listed as:

- Selection
- Joins
- Group-By
- Projection
- Order-By

Workload exploration:

One of the main topics in Ettu [13] project is to investigate how SQL queries can be compared effectively and accurately. In their research [14], the authors survey the literature for the methodologies used in this field and elaborate on three available methods [15–17].

Aouiche *et al.* [15] utilize a pairwise similarity metric between two SQL queries in order to optimize view selection in data warehouses by creating feature vectors out of the *selection*, *joins* and *group by* items in the query while not considering the appearance count. They use Hamming Distance to calculate a similarity value between these two vectors.

Aligon *et al.* [16] present a survey on a great range of approaches which seek to put forward a similarity function to compare the similarity of OLAP sessions. Inspired by their findings, they propose their own query similarity metric which considers *projection*, *group by*, *selection* and *join* items for queries issued on OLAP datacubes. Their method creates separate sets for each of these components and appoint an importance weighting to each set. When comparing two SQL queries, each of these three sets get compared to the other query's corresponding set, hence, by using Jaccard Coefficient, they get a similarity score for each set. Finally, they compute an overall similarity score by the average of these three scores.

Makiyama *et al.* [17] put forward the most similar work we are working on. They perform query log analysis with a motivation of analyzing the workload on the system, and they provide a set of experiments on Sloan Digital Sky Survey (SDSS) dataset. They extract the terms in *selection, joins, projection, from, group by and order by* items separately, and create the query vector out of their appearance frequency for each query in the dataset. They compute the pairwise similarity of queries with cosine similarity.

When we inspect closely, we can easily see that Aouiche method is a naive version of the other two methods. In our experiments, we apply both Aligon and Makiyama methods, since they have competitive performance as shown in Ettu's log summarization study [14]. Makiyama method can be used to perform k-means and hierarchical clustering since they provide the feature vectors, whereas Aligon method can only be used to apply hierarchical clustering since the method is used to create a distance matrix, not feature vectors.

To clarify the ambiguity between distance and similarity terms, we define distance as follows:

$$distance = 1 - similarity$$

where the similarity is the score we get from the methods explained above.

Structural complexity:

Another approach to calculate a pairwise similarity score for SQL queries is to treat SQL as a programming language, and utilize an approach that code plagiarism detectors take [18]: first, we generalize all grammar items into their SQL types, and extract n-grams of each query. For example,

```
Q1: SELECT rank, count(*) FROM user
    WHERE rank <> "admin" GROUP BY rank
```

gives us the following terms:

```
Terms(Q1) = {SQL_SELECT,
             COLUMN,
             COUNT,
             OPEN_PARANTHESIS,
             ALL_COLUMNS,
             CLOSE_PARANTHESIS,
             SQL_FROM,
             TABLE,
             SQL_WHERE,
             COLUMN,
             NOT_EQUALS,
             CONSTANT,
             SQL_GROUPBY
             COLUMN}
```

Pairwise comparison of the n-gram vectors created from these items for each query with cosine similarity is the pairwise similarity score.

Although this approach clearly cannot be used to understand the workload characteristics of a query log due to the information loss while converting the query into term type n-grams, it is expected to successfully cluster queries engineered with the same approach.

We consider two possible clustering approaches for use in our system: k-means and hierarchical clustering [19]. K-means outputs a set of queries for k clusters. On the other hand, hierarchical clustering outputs a dendrogram – a tree structure which shows how each query can be grouped together. In addition, a dendrogram is a convenient way to visualize the relationship between queries and how each query is grouped in the clustering process. We will explore the possibility of using both of these methods.

3.3 Pattern Matching

Soikkeli *et al.* [10] say that even considering the time between launch and close of an application is not a reliable notion of an application usage session. Applications running in the foreground are visible to the user. Applications which are running in background are not visible to user even though he might have launched them before. An individual session now consists of two parameters: a start time and an end time. Two user sessions can be back to back or might have an idle time in between them. User sessions for a single application can now be modeled as a time-wise closely-spaced series of queries issued to the smartphone database. A threshold value T is defined for the idle time between two transactions. If the idle time is less than or equal compared to T, the transaction belongs belong to the same user session. This approach enables us to identify a time window which can be applied to the transactions in the PocketData dataset. This time window would contain a chronologically ordered subset of queries issued to the smartphone database.

Suppose, if a query for a user consists of a series of chronically ordered queries $Q = [q_1, q_2, ...q_n]$ and f be the function which converts this series into windows using the above mentioned logic.

$$f(q_1, q_2, ...q_n) \rightarrow [w_1, w_2, ..., w_m]$$

where $[w_1, w_2, ..., w_m]$ is the series of user sessions the are obtained. Each w_i consists of chronologically ordered bag queries Q_{w_i}. Also, $\bigcup_1^m Q_{w_i} = Q$ and $Q_{w_i} \cap Q_{w_j} = \varnothing$ $\forall\, i, j \in [1, m]$ and $i \neq j$

There are some peculiarities of the query logs that must be considered in order to design the methodology of working with them. Any user activity on a smartphone app consists of a sequence of multiple asynchronous operations. For example, a user might want to refresh the Facebook feed updates from time to time. The user might perceive this as a single repeating activity performed multiples times in a day. But on app performs multiple transactions during each "burst" of the same activity. Given the asynchronous name of most smartphone applications, the relative order in which these queries are issued is not fixed. This is also reflected in the query logs. User sessions might be similar to each other in terms of the intent. The intent is reflected by the query cluster that a particular query belongs to. But we can not test for similarity among these sessions by searching for common subsequences. It is highly probable that a group of queries might be be issued as part of the same logical task but they might appear to be interleaved in the query log. Each user session can be treated as a bag of queries. Hence, we need to use a similarity measure which works on the basis of membership for a particular bag. Jaccard Similarity is a simple measure to meet the above mentioned requirements.

The Jaccard similarity coefficient is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. For two user sessions w_i and w_j, we compare the membership of the clusters that the constituent queries belong to.

$$J(w_i, w_j) = \frac{[w_i \cap w_j]}{[w_i \cup w_j]}$$

If w_i and w_j are both empty, we define $J(w_i, w_j) = 1$. Also, $0 \leq J(w_i, w_j) \leq 1$.

We calculate $J(w_i, w_j)$ for all pairs of user sessions. Now, we start to look for "interesting" user sessions. One notion of of user sessions being interesting can be that their contents occur in the query log more frequently. A high Jaccard similarity score for a pair of user sessions can be interpreted as them being similar to each other, thereby leading the contents to occur more frequently. For a particular user session w_i, we would be now looking out for the top K user sessions which are most similar with w_i. Calculating the average similarity of w_i with the most similar K sessions $[w_1, w_2..., w_k]$ yields a notion of the importance of w_i in representing the characteristics of the workload represented in the query log. We denote this average similarity for w_i with top K windows as $J_{w_i avg}$.

$$J_{wi_{avg}} = \frac{\sum_{j=1}^{k} J(w_i, w_j)}{k}$$

When we calculate $J_{w_i avg}$ for all $w_1, w_2..., w_m$, we obtain a vector of average similarity scores for the entire query log for a user.

$$[J_{w_{1_{avg}}},\ J_{w_{2_{avg}}},\ J_{w_{3_{avg}}},\ ...,\ J_{w_{m_{avg}}}]$$

This vector denotes the relative importance of the user sessions to characterize the workload of the application.

4. EXPERIMENTS

4.1 Environment

In our experiments regarding clustering, we used an Apple Macbook Pro with macOS Sierra operating system, 2.7 GHz Intel Core i5 processor, 8GB RAM, Java 1.8 SE Runtime Environment and R v3.3.2.

In our experiments regarding pattern matching, we used a Lenovo Thinkpad with Windows 10, 2.3 GHz Intel Core i5 processor, 8GB RAM, Java 1.8 SE Runtime Environment and R v3.3.1.

4.2 Clustering

For our experiments, we selected Facebook to be our example app. For visual purposes, we clustered the activities of only one user in Figure 2. For this specific user's case, there are 84273 rows of activities in the log. There are 8795 parsable select queries, however, there are only 59 unique queries among them. Keep in mind that PocketData dataset is an anonymized dataset where most of the constant values are replaced with "?", which reduces the number of distinct queries greatly. The dendrogram we created using Makiyama method and hierarchical clustering can be seen in Figure 2.

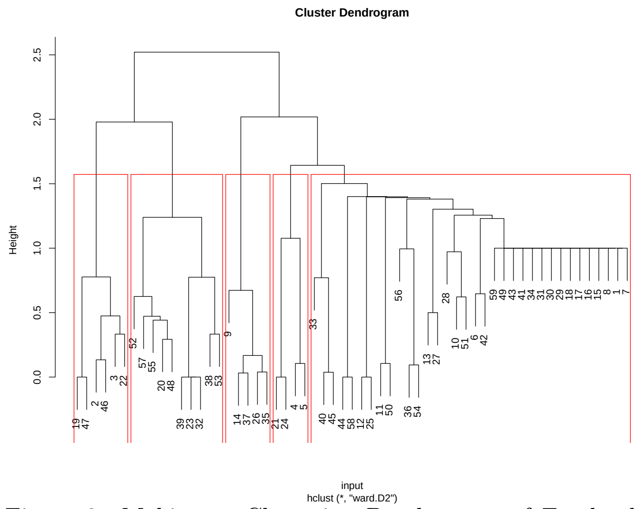

Figure 2: Makiyama Clustering Dendrogram of Facebook usage for a user

As random examples, here are some the queries that are put in the same clusters:

```
Cluster 1:
19 -> SELECT seen_state,
             updated,
             cursor,
             cache_id,
             dashing,
             icon_uri,
             photo_uri,
             profile_picture_uri,
             summary_graphql_text_with_entities,
             short_summary_graphql_text_with_entities,
             notif_id,
             star_rating
      FROM gql_notifications
      WHERE (recipient_id=?)
      ORDER BY updated DESC LIMIT ?

3 -> SELECT cursor
     FROM gql_notifications
     WHERE (recipient_id=?)
     ORDER BY updated DESC LIMIT 1

Cluster 4:

4 -> SELECT timestamp,
            data,
            fetchreason
     FROM cache
     WHERE cachekey= ?

21 -> SELECT value, timestamp
      FROM cache
      WHERE (name='MFacewebVersion:MRootVersion')
      ORDER BY name DESC
```

As seen in Figure 2, there are 5 different clusters of queries when clustered with Makiyama method. In Table 1, we provide the tasks performed by the queries within the corresponding cluster.

Table 1: Makiyama lustering results

Cluster	Explanation
1	New notification check
2	Prefetch and retrieve notification
3	Fill home feed
4	Cache operations
5	Housekeeping

Also, for the n-gram approach, when we choose n to be 2, we created the clustering shown in Figure 3.

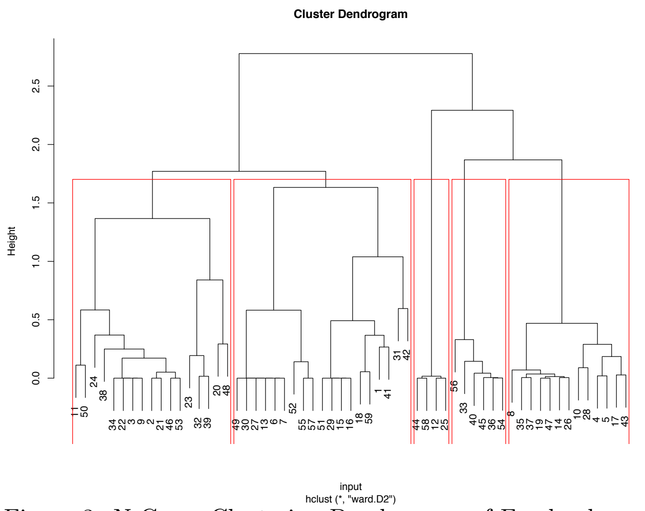

Figure 3: N-Gram Clustering Dendrogram of Facebook usage for a user

In Table 2, we provide the explanations for the queries according to the clusters they got appointed with n-gram feature extraction scheme.

We also created a tanglegram to show how similar clusterings these two methods created in Figure 4. As can be seen in the figure, there is little to no similarity between these clusterings which is not unexpected since the two feature extraction mechanisms completely have different strategies and targets different features.

Table 2: N-Gram clustering results

Cluster	Explanation
1	Key-Value lookups
2	No filter or multiple row lookups
3	Lookup in a provided list
4	Complex queries
5	Top-k row queries

4.3 Session Identification

The first set of experiments were directed towards coming up with an idle time tolerance. We ran our user sessions segmentation routine for query logs of 3 users for the timeline of a month. The idle time tolerances used were 10ms, 100ms, 1s, 5s, 10s, 1min, 2min and 5min. We noticed that the number of user sessions generated converged to around 10000 milliseconds of idle time tolerance as can be seen in Figure 5. So, we chose an idle time of 10 seconds for further experiments.

While the idea of a low idle time tolerance might be enticing because it enables us to look into the query log at a more granular level, we decided that this approach was more suitable. When the number of user sessions became too high, neighboring sessions started to become very similar to each other. This might lead to a myopic view of the data. Also, it is reasonable to hypothesize that the general usage pattern of smartphones is in bursts. The user would pickup the smartphone for a few minutes, perform a bunch of tasks and then keep it away. During these bursts of activity, the high similarity among smaller user sessions could be because of the fact that if a user is checking the Facebook feed for 5 seconds, it is highly probably that they will keep doing that for the next many seconds. However, we are able to deal with this myopic view with larger idle time tolerances. Also, higher idle time tolerances lead to lower number of user session windows. The time complexity of similarity calculation operation is $O(n^2)$. Higher idle time tolerances fit the general usage patterns, as well as, reduce the computational complexity of calculating the average similarity vector.

4.4 Common patterns

Extracting meaningful patterns from user data is central to a reliable characterization of the smartphone database workload. We proposed a robust similarity measure which takes into the account the considerations and constraints that are imposed by the problem domain. We believe that our experimental results support the dependability of this approach for mobile applications.

We applied the idle time treshold as 10 seconds over the Facebook query set for a month of usage for 11 users as we indicated in the previous section in order to determine how many sessions there are. This operation revealed that there were 15820 sessions initiated in the dataset according to the session definition. Among these 15820 sessions, 5184 of them had 90% or higher average similarity with all other sessions and there are 25 sessions that had 25% or less average similarity with all other sessions as show in Figure 6.

We believe that a random session selection among the 5184 sessions can provide us with a representative query set of the workloads for all users since 90% average similarity means the query set represents 90% of all the sessions in the dataset. The average, minimum and maximum session lengths are given in Table 3.

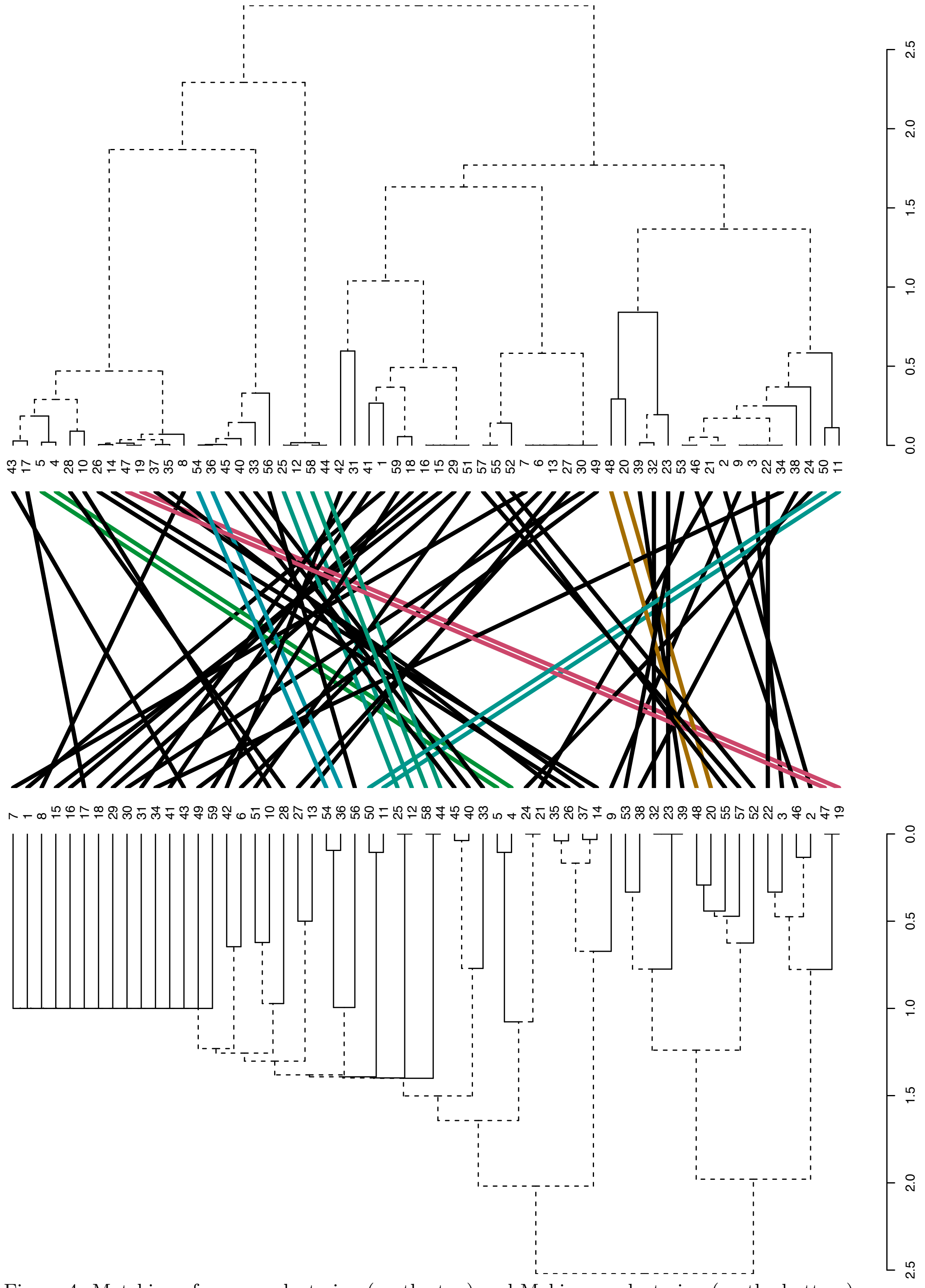

Figure 4: Matching of n-gram clustering (on the top) and Makiyama clustering (on the bottom)

As for outlier detection, the 25 sessions that have 25% or less average similarity is a very moderate number that should be inspected to understand the structure of extraordinarily different sessions.

5. CONCLUSION

The focus of this paper is to identify common behaviors and unusual patterns in user activities on mobile databases with the hypothesis that making use of this information can

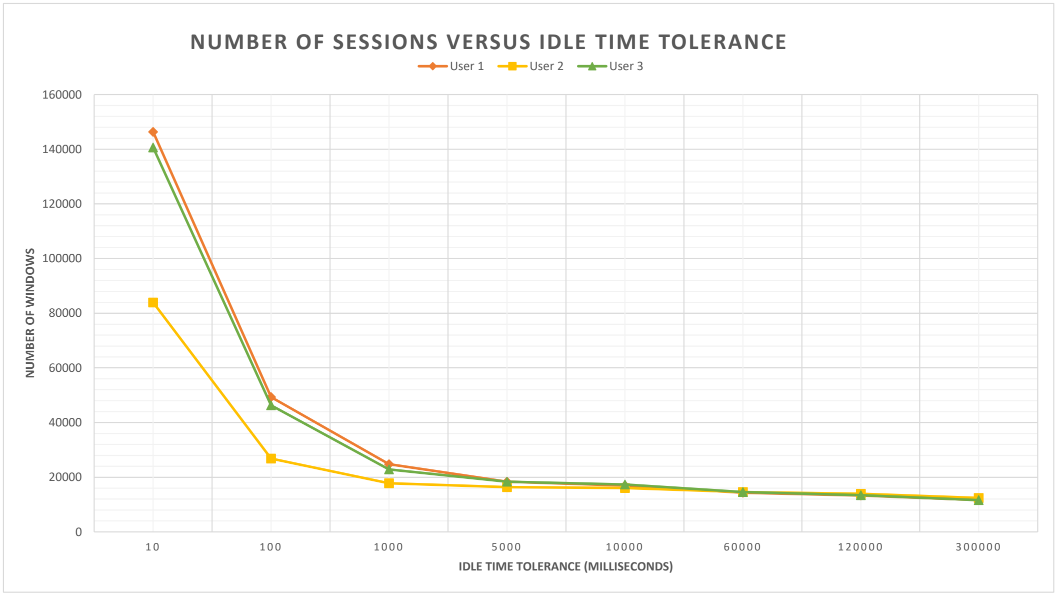

Figure 5: Number of user sessions generated with varying idle time tolerances

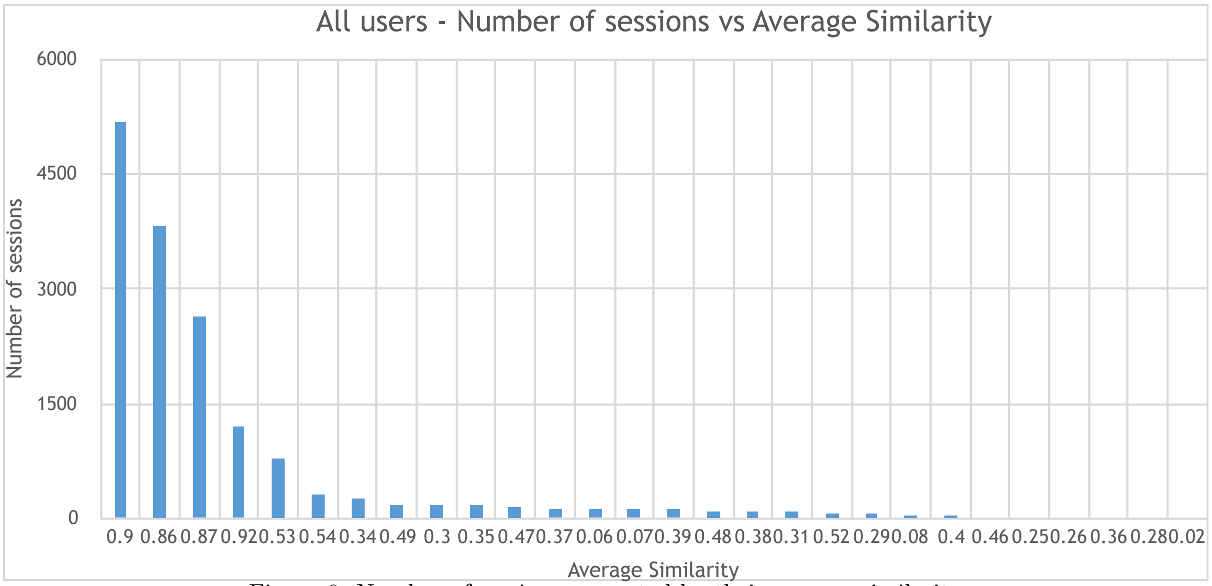

Figure 6: Number of sessions separated by their average similarity

open up a lot of opportunities for mobile apps. Identifying the common behaviors are essential for creating a *small data* benchmark which compares the performance of mobile database management systems under different workloads. Another usage scenario of this information is to find out bugs and unnecessary function calls by identifying repeating queries on data that has not change the last reading. To achieve this, we analyze PocketData dataset which consists

Table 3: Session length

	Average Session Length	Minimum Session Length	Maximum Session Length
Sessions with 90% or higher similarity	3474.32 ms	10 ms	159320 ms
Sessions with 25% or less similarity	2402.91 ms	4 ms	136110 ms
All sessions	5065.02 ms	4 ms	218959 ms

of SQL queries posed by different applications for 11 users for a month. We utilize different query similarity methods to identify important features out of these queries, form feature vectors out of them, and cluster the similar queries together by their feature vectors with hierarchical clustering. Finally, we discuss on how we can make use of this clusters; we appoint an integer label to each cluster, and whenever there is an incoming query from a user, we identify which cluster the new query belongs to. These labels create a sequential array of integers for that specific user, in which we explore interesting and repeating patterns.

6. FUTURE WORK

This paper represents the first steps for developing a *small data* benchmarking tool for apps running on mobile devices. We plan several extensions as future work.

First, our efforts, until now, focused on how users access data instead of their total utilization of database system capabilities: inserts, updates, and deletions along with select statements. This will require us to modify the current query comparison methods since their specifications do not support them. We will continue to expand the scope of our analysis through understanding more statement types and their effects on the query load.

Second, the PocketData dataset contains the time which the query took to execute itself. We have not used this measure in our analysis yet. This could prove to be a valuable aid in uncovering further characteristics of the data.

Finally, we will investigate how to emulate a workload utilizing the findings in this paper automatically. This step is essential for creating a benchmarking tool. This includes concentrating our focus on both emulating the queries and generating data for the mobile application we are evaluating.

PocketBench will be a decision support benchmark that will consist of a series of queries that are created from real user query logs and concurrent mobile data manipulation operations. The queries and the data populating the database will be realistically emulated from the common patterns and sessions that we identified in this work. This benchmark will

1. Examine realistic amounts of data in a mobile database

2. Execute queries with complexities proportional to the mobile app produces

3. Give answers to real-world mobile application performance questions

4. Simulate generated queries

5. Generate realistic activity on the mobile database under test

6. Be implemented with constraints that real production line mobile databases have.

Acknowledgments. We would like to thank Dr. Oliver Kennedy and Dr. Lukasz Ziarek for their support and guidance for our project. We also would like to thank the CSE662 students for their helpful comments and suggestions.

7. REFERENCES

[1] Jyrki Ali-Yrkkö, Matias Kalm, Mika Pajarinen, Petri Rouvinen, Timo Seppälä, Antti-Jussi Tahvanainen, et al. Microsoft acquires nokia: Implications for the two companies and finland. *ETLA Brief*, 16(3), 2013.

[2] Felix Gillette, Diane Brady, and Caroline Winter. The rise and fall of blackberry: An oral history. *Bloomberg Business Week. Bloomberg*, 5, 2013.

[3] Jason Wei. *Android database programming.* Packt Publishing Ltd, 2012.

[4] Grant Allen and Mike Owens. *The definitive guide to SQLite.* Springer, 2010.

[5] Oliver Kennedy, Jerry Ajay, Geoffrey Challen, and Lukasz Ziarek. Pocket data: The need for tpc-mobile. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 8–25. Springer, 2015.

[6] Transaction Processing Performance Council. TPC-H benchmark specification. *Published at http://www.tpc.org/tpch/*, 2008.

[7] Transaction Processing Performance Council. TPC-C benchmark specification, v5.11. *Published at http://www.tpc.org/tpcc/*, 2010.

[8] Je-Min Kim and Jin-Soo Kim. Androbench: Benchmarking the storage performance of android-based mobile devices. In *Frontiers in Computer Education*, pages 667–674. Springer, 2012.

[9] Jean-Pierre Dijcks. White paper: Big data for the enterprise. Technical Report 519135, Oracle Corporation, 500 Oracle Parkway Redwood Shores, CA 94065 U.S.A., June 2013.

[10] Tapio Soikkeli, Juuso Karikoski, and Heikki Hammainen. Diversity and end user context in smartphone usage sessions. In *2011 Fifth International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 7–12. IEEE, 2011.

[11] Jinghao Shi, Edwin Santos, and Geoffrey Challen. Why and how to use phonelab. *GetMobile: Mobile Comp. and Comm.*, 19(4):32–38, March 2016.

[12] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. Self-driving database management systems. In *CIDR 2017, Conference on Innovative Data Systems Research*, volume 10, pages 707–722. VLDB Endowment, 2017.

[13] Gokhan Kul, Duc Luong, Ting Xie, Patrick Coonan, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. Ettu: Analyzing query intents in corporate databases. In *Proceedings of the 25th International Conference Companion on World Wide*

Web, pages 463–466. International World Wide Web Conferences Steering Committee, 2016.

[14] Gokhan Kul, Duc Luong, Ting Xie, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. Towards effective log summarization. *Published at http://odin.cse.buffalo.edu/papers/2017/EDBT-SummarizingSQL-submitted.pdf*, 2016.

[15] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. Clustering-based materialized view selection in data warehouses. In *ADBIS*, 2006.

[16] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Similarity measures for OLAP sessions. *Knowledge and information systems*, 2014.

[17] Vitor Hirota Makiyama, M Jordan Raddick, and Rafael DC Santos. Text mining applied to SQL queries: A case study for the SDSS SkyServer. In *SIMBig*, 2015.

[18] Ameera Jadalla and Ashraf Elnagar. Pde4java: Plagiarism detection engine for java source code: a clustering approach. *International Journal of Business Intelligence and Data Mining*, 3(2):121–135, 2008.

[19] Rui Xu, Donald Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.