Summarizing Small Data Workloads

Anonymous authors

ABSTRACT

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. First, we describe a clustering scheme where we analyze the query logs to group the SQL queries by semantic similarity. Then, we introduce a session identification technique for mobile query workloads where we identify bursts of activities created by the user actions. Finally, we elaborate on using these clusters and session information to model common behaviors and unusual patterns. We demonstrate that 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 indicate bugs or inefficiencies in the app.

Keywords

Benchmark, Database, Workload, Mobile Systems

1. INTRODUCTION

Many modern smartphone apps and services need to persist structured data. For this task, developers typically turn to embedded database like SQLite, which are available as a part of most modern smartphone operating systems. SQLite and databases like it are fully-featured relational databases, and so mobile apps and operating systems represent a new class of database clients. Embedded databases play a significant role in the performance of smartphone apps [26]. Unfortunately, the needs of apps are hard to characterize: In contrast to server-class workloads, each individual app has a dedicated database and is typically only used by one user, for only short periods at a time. As a consequence, the database workload created by a typical app is bursty, variable, and hard to summarize as a simple distribution of queries [11].

In this paper, we propose using a two-level hierarchy for summaries of query workloads created by smartphone apps: We treat app workloads as collections of tasks, or bursts of database activity typically triggered by self-contained user activities such as checking a Facebook feed or composing an email. Concretely, we tackle the problem of reducing a log of per-app query activity down to a set of task categories. Using recently collected traces of smartphone query activity in the wild [11], we are able to show that our approach is able to cluster queries into meaningful tasks. These summaries can in turn be used by embedded database developers to better understand app requirements, as well as app developers to better tune app performance. Summaries can also be used as the basis for synthetic workloads representative of individual apps.

Our solution addresses three specific challenges. First, we assume that the app's query log arrives as one large sequence of queries. We explore how to segment the log into sub-sequences of queries, each corresponding to one task. Although there are no explicit cues from the user that signal that a task has started or completed, we show that it is possible to use query inter-arrival rates to identify a time threshold between tasks.

We next consider linking these sub-sequences together into self-similar clusters. Every cluster corresponds to one category of task. The underlying question is what makes two clusters similar. Query similarity has already received significant attention [2, 1, 15]. A common approach is to describe queries in terms of features extracted from the query — Common features include which columns are projected or what selection predicates are used. Although our underlying approach is agnostic to how these features are extracted, we experiment with a variety of feature definitions and adopt a feature extraction method proposed by Makiyama *et al.* [15].

In principle each sub-sequence of queries could be described by the features of those queries. As we show, clustering directly on the query features is neither scalable nor reliable. Instead, we add an intermediate step where we first cluster individual queries, allowing us to first link related queries. These cluster labels then serve as the basis for the task-similarity metric.

Concretely, in this paper we: (1) Identify the challenges posed by mobile app workloads for query log summarization, (2) Propose a two-level, task-oriented summary format for mobile app workloads, (3) Design a process for creating task-oriented summaries from query logs, and (4) Evaluate our summarization process, showing that it efficiently creates representative summaries.

This paper is organized as follows. We first describe the moving parts of the system in Section 3. Then, we give a detailed description of our framework and our methods in Section **??**. In Section 7, we introduce a sample dataset for workload characterization, and we evaluate our proposed techniques using this dataset. Finally, we conclude in Section 8, and identify the steps needed to deploy our methods into practice in Section 9.

2. PRELIMINARIES

The usage pattern of databases in smartphones differs significantly from the traditional database server and web application workloads. Most modern day smartphones typically rely on a web service to help a mobile application deliver the desired functionality of the application to the user, as opposed to keeping the actual data in the database. Thus, a mobile database typically serves as a cache to hold recently accessed data to be used in case of a connection failure, and data required for the business logic of the mobile application. This enables the application to defer getting updates of recent changes, and downloading of live data until connection is restored. Also, this helps asynchronously fetching data from the web services while still being able to show the user a consistent state of the information.

Mobile applications that work on smartphones are the sole owners of their dedicated databases. None of the other applications, or the operating system can access to an application's database. When a user starts to interact with an application, the application starts to generate queries for the user's information needs. Since the database acts as a cache for the application, there are a lot of repetitive queries that brings the most recently fetched information to the user interface. This behavior creates a unique workload for the mobile databases where a user's activity fires up a *burst* of queries in a very short time while little to no activity when the user is not actively engaging with the application. A sample workload gathered from Facebook application can be seen in Figure 1.

The effects of these characteristics are threefold: (1) the workload optimizations does not require considering the behavior of many other users' workloads, and should focus on the individual behavior of the user, (2) the workloads are bursty; they do not spread over time as traditional database workloads do, and (3) the bursts are usually focused around a limited number of activities that can be explained with the user's actions, or the governance needs of the application itself.

2.1 Session Identification

In traditional databases, a *session* is defined as a connection between a user or an application, and the user [17]. Basically, every session has (1) a user or an application as the owner, (2) a start time where the user connects to the database, and (3) an end time where the user disconnects from the database.

This basic understanding of a session is not always enough for specific purposes such as prefetching predicted queries, or automatic query suggestion to the user. Yao *et al.* [9] define a session as a sequence of queries issued to the database by a user, or an application to achieve a *certain task* where they adopt the some of the methods described from their previous work on web logs [8].

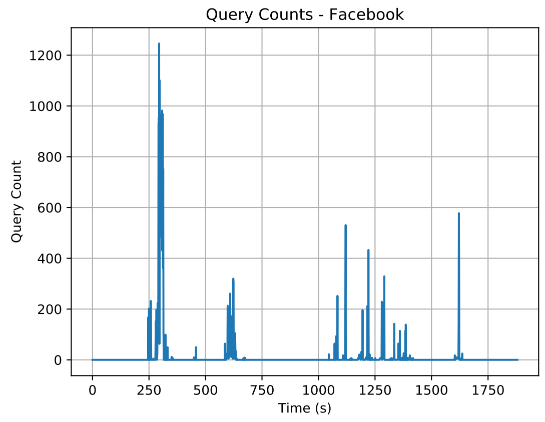

Figure 1: Sample Facebook Workload

The research on how to make use of database sessions using database query logs approximates to session identification in web search engine query logs, and it focuses on three approaches: (1) connection time approach, (2) timeout based approach, and (2) semantic segmentation of topics approach.

Connection time approach. This approach assumes that all the activity (and inactivity) belongs to one session as long as the user or the application is connected to the system.

This approach is not suitable for mobiles systems since the database user sessions are local to the database. They do not correspond to the events of applications' connection and disconnection to the database.

Soikkeli *et al.* [21] 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.

Timeout based approach. This approach is based on identifying a time-out value that is ideal for the given scenario to detect session boundaries. The queries that are issued between two boundaries belong to the same session.

Semantic segmentation based methods. This approach focuses on the content of the queries in order to understand the context change in the query workload [10, 9, 6]. The assumption is that, if two queries are semantically close to each other, they should be placed in the same session, and if there is a shift in the query interest, there should be session boundary between these queries. Naturally, this raises the question of *what makes two queries similar?* The research on this question focuses on various motivations, such as database performance optimization [2], workload exploration [15], and security applications [12].

Yao *et al.* [9] report that sessions can be identified by studying change in information entropy. They use a language model which is chacterized by an order parameter and a threshhold parameter. The order parameter determines the granularity of the n-gram model which would be used to break the query log into smaller sequences of queries. The conditional probability of occurence of those sequences is calculated from the training data consisting of queries with sessions already identified beforehand. Using these probabilities, a running count of entropy is calculated for all sessions. The entropy parameter determines a threshold value at which a session can't accomodate any more new queries. These queries form part of another session and contribute of its entropy. If a sequence of queries has been observed to be occuring close to each other before, their entropy value will be low. This indicates presence of some kind of link between them and hence supports the case of them being in the same session. However, when a completely unrelated querie is being considered to be part of a particular session, the entropy value of the session will rise. The system would place the query in a different session. This approach is not dependent on time intervals. Even though this approach is very intuitive, it falls short of addressing specific issues in smartphone database sessions. As already described, there is no clear way to identify start and end of a session. As a result, it is not possible to obtain a training set of queries with sessions already labelled. Hence, this approach doesn't work for session identification in our case.

Hagen *et al.* [7] present an interesting approach to session identification which doesn't need simultaneous evaluation of all features of queries in the log. They propose the Cascade method which processes features in different steps each with increasing computational cost. When a computationally cheaper feature can make a reliable decision about session identificatio, features with higher computational cost and runtime are not processes. Additional features are evaluated only when computationally cheaper features don't provide a reliable decision. The Cascade method is designed on the assumption that time is not a good indicator of session boundaries. The scenario described in their work deals with online web searches. So, users stop working and resume their logical sessions after arbitarary amounts of time. They also refer to the state-of-the-art geometric method by Gayo-Avello [5]. Like the geometric method, the cascade method uses the time and lexical similarity of queries to decide session boundaries. But for query pairs that are chronologically very close and lexically very different, the decision of the geometric method are not reliable. Hence, the cascade method invokes explicit semantic analysis and search result comparison to help decide session boundaries.

2.2 Session Similarity

Once the the database user sessions have been identified, we need to identify sessions which perform similar logical activities. A session can include one or more activities, and activities consist of a bag of queries as illustrated in Figure 2. Exploring the similarities of sessions could be used to identify repeating patterns.

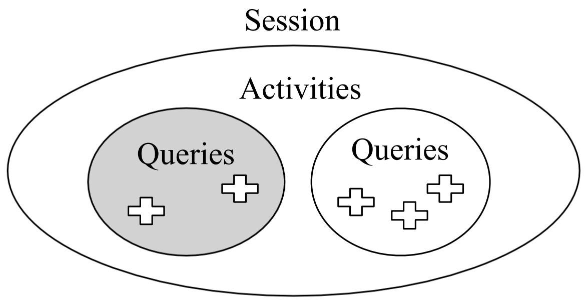

Figure 2: Session - Activity - Query relationship

Aligon *et al.* [1] report that there are 4 approaches in the literature for computing session similarity: (1) Edit-based approach, (2) Subsequence-based approach, (3) Log-based approach, and (4) Alignment-based approach.

Given two sessions A and B, and a treshold θ, any approach given below first creates two sequences of placeholders for queries by comparing queries in both sessions to label each query pair that has a similarity score greater than θ as the same, and appoints distinct labels for other queries.

Edit-Based Approach. It finds the Levenshtein distance between the resulting sequences.

Subsequence-Based Approach. It computes the dice coefficient of n-grams of resulting sequences.

Log-Based Approach. It calculates the tf-idf value of the two sequences. This approach penalizes the queries that repeats a lot but is not distinctive.

Alignment-Based Approach. It considers the ordering of the queries along while comparing n-grams of resulting sequences. It finds the best alignments of n-grams to maximize the similarity.

2.3 Mobile Workload Analysis

There are various widely used benchmarking systems on traditional relational databases [22, 23, 24, 16]. DWEB [4], which is a data warehouse benchmark, also provides parameterization of data in terms of number of tables, and tuples; and parameterization of queries in terms of number of queries, and attributes in a query. These benchmarks create scalable fixed data, and homogenous query workloads. They focus on throughput and response time as the performance metric.

NoSQL database benchmarks [3, 25], on the other hand, appear to be more suitable to mobile database systems by providing support for configuring the query workload in terms of density of inserts, updates, deletes, and selects. However, the queries are still pre-set in TPCx-IoT [25]. YCSB [3] only measures performance of key-value stores, and requires to be extended in order to process more complex queries. Additionally, the workload created is still homogenous, and sequential.

There are also some mobile system database micro benchmarks such as AndroBench [14], which was designed to evaluate the storage performance of the device, and not the database management system itself.

Even though a few of the mentioned benchmarks can satisfy the requirements for mobile database systems, none of these benchmarks provide an accomplished mobile database management system evaluation, since there is a lack of understanding the behavior of mobile apps, and mobile operating systems as a class of database clients [11].

3. SYSTEM OUTLINE

In order to create a representative sample of the workload, we need to identify patterns in the query log. The task of identifying patterns directly from the log is difficult. A naive approach would be to consider individual queries as atomic components of the query logs and cluster them by directly extracting features. The most important difference between a smartphone database workload and that in database servers are the users' bursts of activity. Most benchmarks like the TPC-C focus on emulating homogenous query workloads of an OLTP system. Their goal is to analyse throughput fpr these homogenous workloads. But it is not correct to truly emulate smartphone query workloads without emulating the inermittent bursts of query activity. These bursts can only be detected by looking at the chronological attributes like query timestamp and query interarrival time. This approach does not consider the chronological ordering of the queries. Hence, This naive clustering is not meaningful. Another level of abstraction is needed to extract meaningful patterns from the query log.

We introduce the concept of sessions to solve this problem. A user would interact with their smartphone multiple times a day for small intervals of time. These bursts of intermittent activity are captured in database user sessions. These form the atomic units of the query log which can be used for detecting patterns.

A logical task, called an *activity*, performed by a user on a smartphone, such as checking for new email, might produce multiple queries to the database. Since smartphone applications keep switching between foreground and background, these queries could be arbitrarily spaced out in time. Hence, one database user session might contain one or more logical user tasks. A logical user task might be spread across multiple database user sessions. These sessions are useful in capture subset of logical tasks which are repetitive. Since there is no discrete indicator of the start and end of a database user session in smartphones, we use a heuristic to help define one. If queries in a log are apart in time beyond a threshold, we consider them to be a part of different sessions.

After partitioning the query log into *sessions*, we create a statistical summary of *similar* activities by providing frequencies of each pattern detected. This could be done in two ways: (1) Supervised approach, and (2) Unsupervised approach.

3.1 Supervised Approach

We present the supervised approach as a method that requires human intervention, and extensive effort since it requires the activity data to be collected and labeled.

In this approach, we define a set of atomic activities that are possible to be performed on a mobile app, and look for them every time a user uses the phone. The frequency of the activities that appear together or individually in these *sessions* provide an outlook of how a user utilizes an application. The process is illustrated in Figure 3.

[[Task: Go on. Describe the figure.]]

3.2 Unsupervised Approach

Our main contribution in this paper is the unsupervised approach, which eliminates the human intervention and effort required. The criteria for this approach to be successful is to be able to have comparable performance with the supervised approach.

With this approach, we take the workload as the only input, and look for the behavior of the user when they take the phone in their hands every time on a typical day. The frequency of certain types of queries that appear together or individually in a user session provides a summary of the expected user activity. This process is illustrated in Figure 3.

[[Task: Go on. Describe the figure.]]

Both the supervised and unsupervised approaches described in the previous section has three blackboxes: (1) Session identifier, (2) Profiler, and (3) Analyzer. In this section, we describe the methods that we applied to implement these black boxes.

4. SESSION IDENTIFIER

In our framework, a *database session* is a logical unit of user interaction. It spans over a period of time and comprises of sequential queries. If two sequential queries are more than t seconds apart, we consider them to be in different sessions. Parameter t is called the Idle Time Tolerance.

Since the number and content of these sessions depend on the idle time parameter T, it is important to identify the best idle time. 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 the following 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.

We incrementally iterate different idle time tolerances t and look at the corresponding number of sessions that are obtained. The optimum value of t is obtained by locating the knee or trade-off point. The non-optimum values of t would require an unfavorably large change in one of the quantities to gain a small amount in the other. [19] This method can implicitly adapt to workloads with different characteristics since the trade-off point is choosen by running through the actual query log.

Algorithm 1 Euclids algorithm

```
In:
    $a \geq 0$, $b \geq 0$
Out:
    the gcd of $a$ and $b$
1: function GCD($x, y$)                ▷ Where $a \geq 0$, $b \geq 0$
2:     if $b = 0$ then return $a$
3:     else return GCD($b, a \bmod b$)
4:     end if
5: end function
```

5. PROFILER

In order to be able to perform a similarity assessment between queries, activities, and sessions, we need to be able to extract features out of SQL queries. 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
```

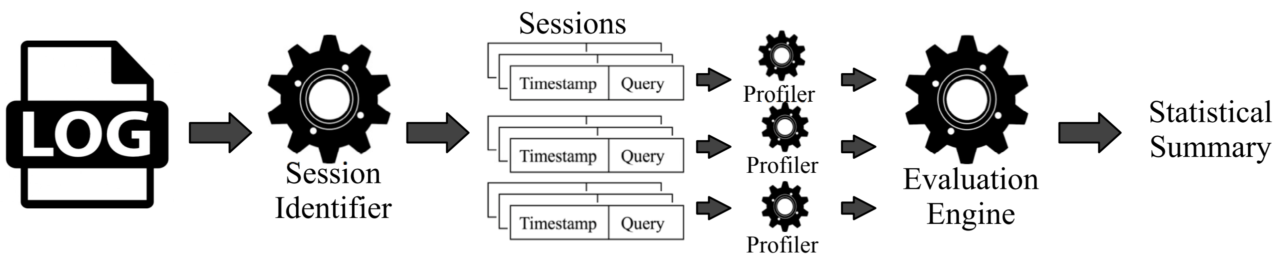

Figure 3: Abstract views of inputs and outputs of both approaches

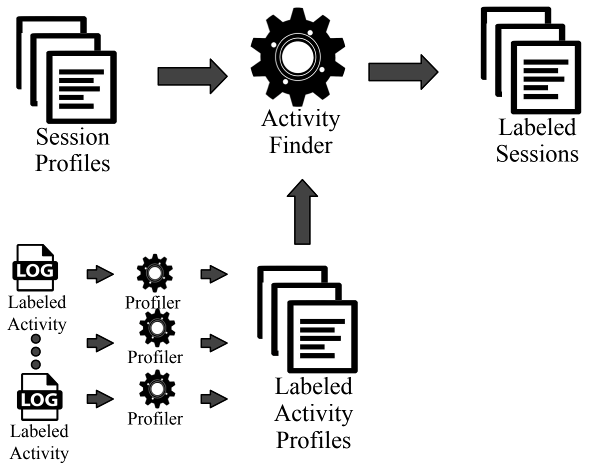

Figure 4: Analysis box of the supervised approach

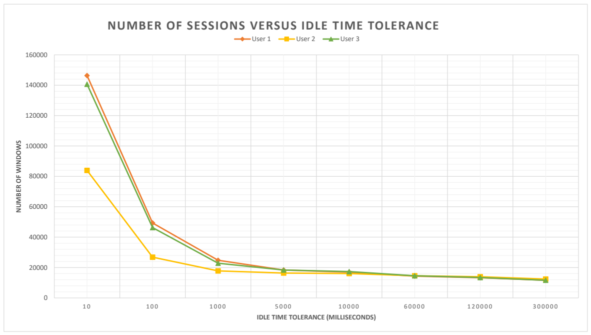

Figure 5: [[Task: Regenerate]]Number of user sessions generated with varying idle time tolerances

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*, and *order-by*.

Makiyama *et al.* [15] 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.

The *profiles* to compare activities and sessions can be performed in three ways: (1) Feature based sets which can be compared with Jaccard Index, (2) KL-Divergence entropy using feature appearance frequencies, and (3) Jaccard Index using the clustering appointments of queries.

Feature sets based Jaccard Index. 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 S_i and S_j, we compare the feature sets of these sessions that are extracted from the constituent queries.

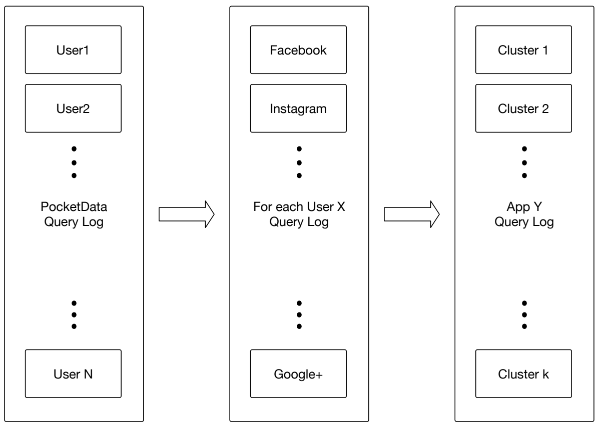

Figure 6: The workflow for clustering process

$$J(S_i, S_j) = \frac{[S_i \cap S_j]}{[S_i \cup S_j]}$$

If S_i and S_j are both empty, we define $J(S_i, S_j) = 1$. Also, this guarantees that $0 \le J(S_i, S_j) \le 1$.

KL-Divergence entropy. Each query $Q_u^{t_i}$ is processed with the methodology given above, and denoted as:

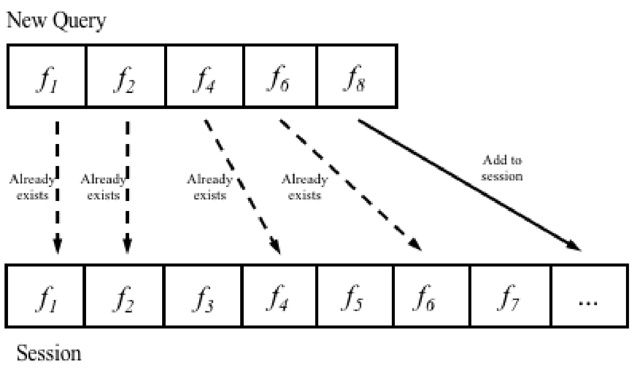

Figure 7: Creating a feature set based session profile

$$Q_u^t = (f_1^t(c_0), f_2^t(c_1), ..., f_m^t(c_n)) \tag{1}$$

where t is the timestamp that the query Q was issued, u represents the username of the query owner, f_i is the feature extracted, and c_j denotes how many times the feature f_i was observed in the query.

A session S is represented by a user $u \in U$, where U is the set of all users, for the time period T that starts from t_0 and goes on for Δt, and the set of queries Q performed by u within T. Formally,

$$S_u^T = (Q_u^{t_0}, Q_u^{t_1}, ..., Q_u^{t_n}) \tag{2}$$

where $Q_u^{t_i}$ represents a query Q_{t_i} issued at time t_i by user u.

The *session profiles* are created with the accumulation of these features from the beginning to the end of the session. Using the appearance frequency of these features, we calculate the appearance probability of each harvested feature. This multinomial probability distribution of the features for each session constitutes the *session distribution*. A session distribution ϕ is formally denoted as:

$$\phi_u^T = (P(f_0)_u^T, P(f_1)_u^T, ..., P(f_n)_u^T) \tag{3}$$

where $P(f_i)_u^T$ represents the probability of encountering feature f_i within the timeframe T among all the operations performed by user u.

We compute the difference between distributions with KL-Divergence [13]. Comparison of a session with the other sessions using KL-Divergence gives the difference denoted as follows:

$$KL(\phi_u^{T_1} || \phi_u^{T_2}) = \sum_i \phi_u^{T_1}(i) log_2 \frac{\phi_u^{T_1}(i)}{\phi_u^{T_2}(i)} \tag{4}$$

KL-Divergence is used for comparing two probability distributions, P and Q; and it ranges between 0 and ∞. $D_{JS}(P||Q)$ essentially represents the symmetric information loss when P distribution is used to approximate Q.

Note that when $P(i) \neq 0$ and $Q(i) = 0$, $D_{JS}(P||Q) = \infty$. For example, suppose, we have two distributions P and Q as follows: $P = \{f_0 : 3/10, f_1 : 4/10, f_2 : 2/10, f_3 : 1/10\}$ and $Q = \{f_0 : 3/10, f_1 : 3/10, f_2 : 3/10, f_4 : 1/10\}$. In this case, since f_3 is not a part of Q, the result would be ∞, which means these two distributions are completely different.

To get past this problem, we can apply *smoothing* (i.e., Laplace/additive smoothing), which is essentially adding a small constant *epsilon* to the distribution, to handle zero values, without significantly impacting the distribution. After we apply smoothing, $D_{KL}(P||Q)$ becomes 1.38.

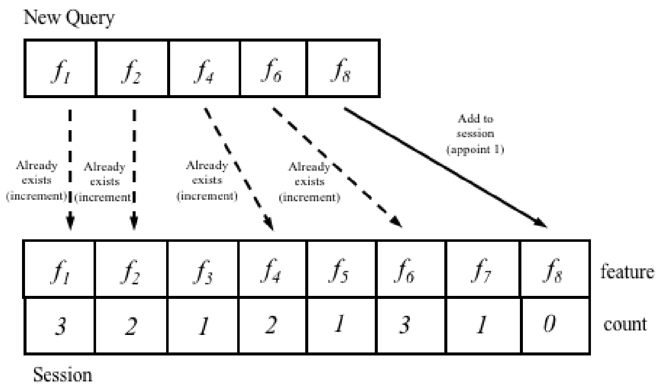
New Query
f_1
f_2
f_4
f_6
f_8
Already exists (increment)
Already exists (increment)
Already exists (increment)
Already exists (increment)
Add to session (appoint 1)
f_1
f_2
f_3
f_4
f_5
f_6
f_7
f_8
feature
3
2
1
2
1
3
1
0
count
Session

Figure 8: Creating an probability distribution based session profile

Jaccard Index using the clustering of queries. In this strategy, the profiler takes the preprocessed clustering appointments of queries as input instead of the features.

Clustering the queries in the workload narrows down the space of possible patterns that could be detected. This facilitates easier and more accurate understanding of the workload [18]. The main goal of this step is to group queries into classes that exhibit similar interests over database attributes. We consider two queries to exhibit similar interests over database attributes if they are similar in semantic structure. In the clustering process, we first filter the queries belonging to the app of our interest without distinguishing which user the activity belongs to. Then, we create clusters using all the queries belonging to that specific app. The workflow for the PocketData dataset is illustrated in Figure 6.

We use hierarchical clustering which takes the distance matrix as input, and outputs a dendrogram – a tree structure which shows how each query can be grouped together. Furthermore, a dendrogram is a convenient way to visualize the relationship between queries and how each query is grouped in the clustering process.

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.

In this form of session profiling, we embed query cluster appointments for all the queries within the session to the *session profile*, which would be used to define the session for the rest of the process. An illustration of how the sessions are represented is given in Figure 9.

[[Task: Show how to calculate the session similarity.]]
[[Task: Add a representative figure.]]

6. ANALYZER

[[Task: To be written.]]

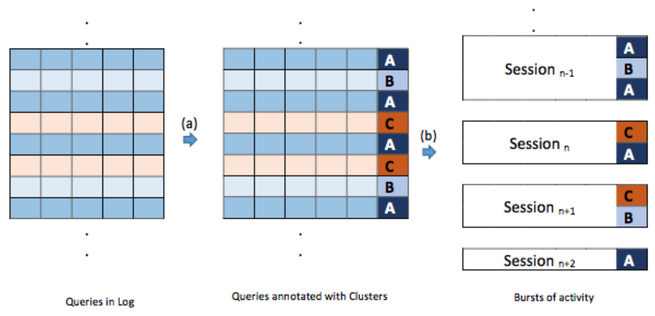

Figure 9: Session representations with query clustering

7. EXPERIMENTS

In this section, we describe the datasets, the environment we performed our experiments in, and the experiment designs along with their results.

7.1 Environment

In our experiments, we used a machine with 2.5 GHz Intel Xeon i7 processor, 128GB RAM, Java 1.8 SE Runtime Environment and R v3.3.2 on Ubuntu 16.04 operating system.

7.2 Datasets

We used three real world datasets to evaluate the performance and accuracy of our design.

7.2.1 Dataset 1 - PocketData Dataset

As the mobile workload dataset, we use an extended version of the PocketData [11] dataset which provides handset-based data directly collected from smartphones for multiple users. It includes 20 day's trace of SQLite activity of [[Note: 60]] PhoneLab [20] smartphones running the Android smartphone platform. University at Buffalo's PhoneLab is a smartphone platform testbed consisting of 175 participants drawn from the university faculty, students and staff. They were provided a discounted Sprint service and a Google Nexus 5 smartphone to use as their primary device. In exchange, they ran a modified Android platform image containing custom instrumentation.

A 2013-14 survey indicates that PhoneLab participants are well distributed among genders and age brackets. [20] Because PhoneLab is open to any UB affiliate , the participants are in many different departments on campus, providing a reasonable level of on-campus spatial coverage.

This dataset consists of various information about the usage patterns across a wide variety of apps, and is a best-effort anonymized dataset. Most of the private information is irreversibly concealed but there are still some constants in the queries.

The older version of the dataset that includes a month's trace of SQLite activity of 11 users is available online [11]. However, the data that we used in this paper is only available by request, and the researchers has to follow the IRB requirements of their institute to be able to use, and publish using this data.

7.2.2 Dataset 2 - Controlled Dataset

We collected this dataset in a controlled environment where one user's activities for one hour are recorded along with the time of each activity, and the activity the user performed.

Table 1: Dataset

Application	**# of queries**
Complete Dataset	45,090,798 (tentative)
Facebook	1,272,779 (tentative)
Google+	2,040,793 (tentative)
Hangouts	974,349 (tentative)
Google Play Services	14,813,949 (tentative)
Media Storage	13,592,982 (tentative)

To create this dataset, we selected *Facebook* application simply because it actively utilizes the mobile database, and we could define the descriptive activities such as opening a profile page, scrolling down on the home feed, and so on.

During the data collection period, the user was free to use the any other application along with Facebook, or not to use the phone at all. However, the user was informed about the experiment was being performed on Facebook application, and they should narrate what they are doing while doing it to be recorded.

This dataset contains 60 minutes of queries that Facebook application issued to the database. There are [[Task: about 5000]] parsable [1] queries recorded, labeled with what activity they were associated with, timing information of each query, and lastly the user labeled session which is roughly described to the user as *"Session in an application is the period between a user starts to dedicate their attention to the application, and loses their interest."* Note that this description is ambiguous, and aims to lead the experiment participant to form their understanding of a session.

7.2.3 Dataset 3 - Controlled Activity

This dataset contains queries that are generated when the user interacts with the mobile application for a specific purpose. While collecting these activity logs, everytime the user completes an operation, the set of queries issued is labeled with that certain activity.

For instance, on Facebook application, the user clicks on another profile on the feed page. This activity makes the application fire a sequence of queries. We capture these queries, and label them with the actual activity.

7.3 Experiment Design

The core expected output of the system presented in this paper is an expected query workload created based on the user's past activity without any supervision. However, it is very difficult to answer the question of *"what is the accuracy of such an output?"* without providing a comparison. Therefore, we compare our session identification methodology with the state of the art of web query session identification, and also measure the accuracy of our session similarity method, we designed a supervised approach given in Section 3.1 to be able to show the unsupervised approach presented has comparable results.

To achieve this, we designed the following experiments as building blocks to verify the accuracy and consistency of the methods that we use in each step.

7.3.1 Experiment 1 - Clustering Accuracy

[1]by JSqlParser

In this experiment, we aim to measure the placement performance of clustering methodology described in Section 5 over all of the queries in the query log.

We prepare ground truth cluster labels by manually inspecting all the unique queries within the query log for Facebook app. The accuracy of the clustering result is measured by comparing the query placements to the clusters to the ground truth.

Table 2: Clustering accuracy

	Facebook	**Google+**
# of queries	#	#
# of unique queries	#	#
# of detected clusters	#	#
# of accurately placed queries	#	#
# of inaccurately placed queries	#	#
Accuracy	#	#

For our experiments, we selected Facebook to be our example app. For visual purposes, we clustered the activities of only one user in Figure 10. 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 10.

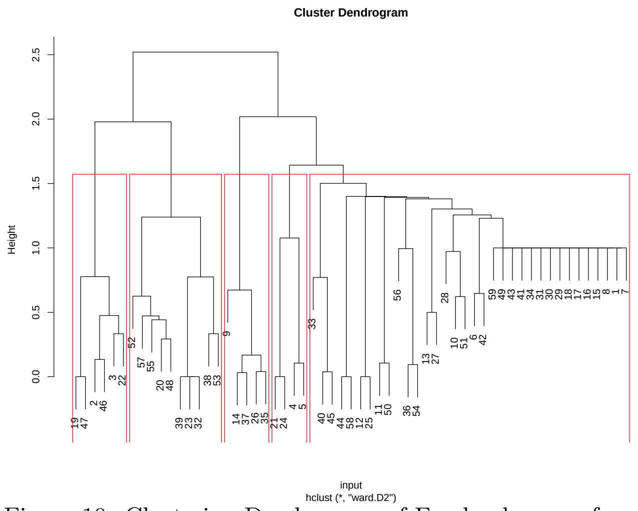

Figure 10: Clustering Dendrogram of Facebook usage for a user [[Task: Renew]]

As seen in Figure 10, there are 5 different clusters of queries when clustered with Makiyama method. In Table 3, we provide the tasks performed by the queries within the corresponding cluster.

We also created a tanglegram to show how the automated clusterings, and manual cluster assignments match in Figure 11.

7.3.2 *Experiment 2 - Idle Time Tolerance*

Table 3: Makiyama clustering results

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

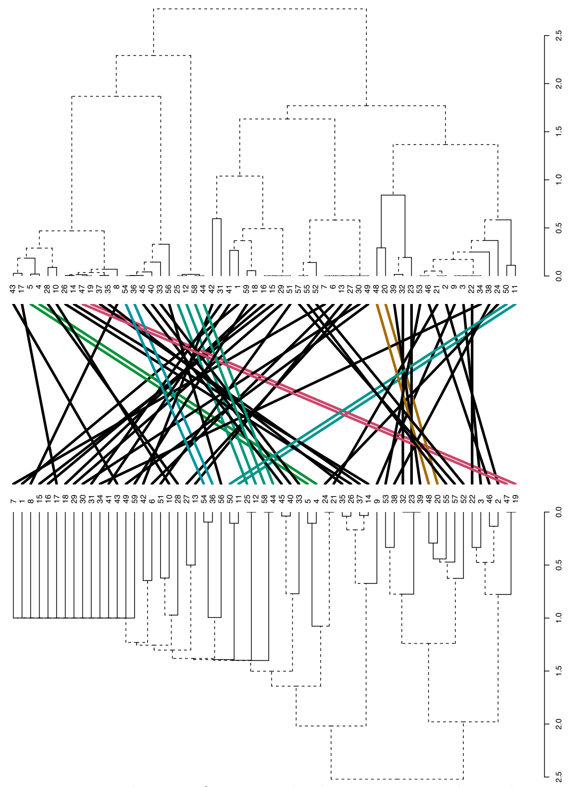

Figure 11: Matching of manual clustering and Makiyama clustering [[Task: Renew]]

This set of experiments were directed towards coming up with an idle time tolerance for each user. We ran our user session segmentation routine described in Section 4 for query logs of all users for all the query workload they created.

The idle time tolerances used were 10ms, 100ms, 1s, 5s, 10s, 1min, 2min and 5min. So, we chose an idle time specific to each user for our further experiments.

[[Task: Create new table for all users with the new data. Show cutoff points clearly.]]

7.3.3 *Experiment 3 - Session Identification on Controlled Dataset*

The controlled dataset is created by monitoring and labeling every activity performed on the phone. Hence, both applying the state of the art search log session identification technique described in Section 2.1 and our heuristic. [[Note: Of course, this experiment here cannot be used to conclude our method is better than the cascade method. It is just an indicator that it is more suitable for mobile workloads.]]

The controlled dataset includes [[Task: 13]] labeled sessions. Our methodology identified [[Task: 12]] sessions, where one of the sessions was labeled to be the tail end of the previous session. Hagen *et al.* [7] methodology, on the other hand, identified only [[Task: 3]] sessions, where [[Task: 2]] of these sessions were correctly identified, but the [[Task: one]] of the sessions identified covered [[Task: 11]] user labeled sessions. This result confirms that although the state of the art method for web search query session identification method is successful in the area it is developed for, it is not suitable for mobile phone database session identification.

7.3.4 *Experiment 4 - Activity Recognition Performance*

The aim of this experiment is to measure the accuracy of various activity detection techniques using labeled data. We collected one hour of Facebook query log where the user recorded every activity they performed freely on the phone with corresponding time data as described in Section 7.2.2. We then collected the query output of a set of activities individually.

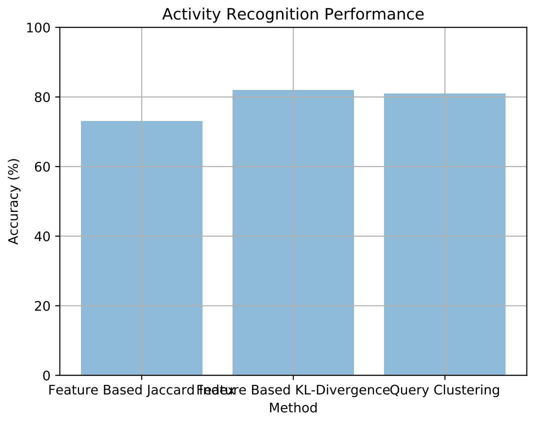

Figure 12: Activity recognition performance for different profiler methods [[Task: Renew]]

Figure 12 shows that feature based KL-Divergence method and query clustering methods have comparable accuracy results.

[[Task: We can test different tresholds and add a ROC graph for each method. It would make this claim stronger, and eliminate any questions that the reviewers can raise.]]

7.3.5 *Experiment 5 - Supervised vs Unsupervised Common Pattern Recognition on PocketData Workload*

[[Task: Warning! This subsection will be filled with new results!]]

Extracting meaningful patterns from user data is central to a reliable characterization of the smartphone database workload. We proposed a robust session 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 specific for each user over the Facebook query workload as we indicated in the previous section in order to determine how many sessions there are. [[Task: Rewrite this: 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 13.]]

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 4.

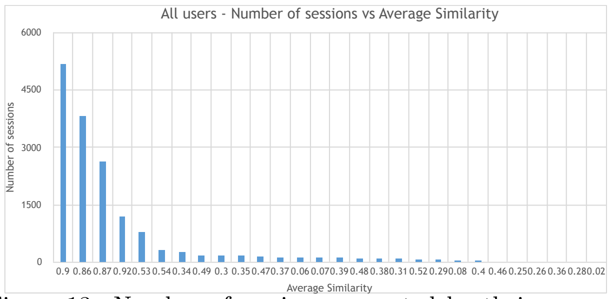

Figure 13: Number of sessions separated by their average similarity [[Task: Will be replaced with supervised and unsupervised method performance comparison graph]]

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.

[[Task: Go on here, add results...]]

8. 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 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 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.

9. 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

Table 4: 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

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.

10. REFERENCES

[1] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia. Similarity measures for OLAP sessions. *Knowledge and information systems*, 2014.

[2] K. Aouiche, P.-E. Jouve, and J. Darmont. Clustering-based materialized view selection in data warehouses. In *ADBIS*, 2006.

[3] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *SOCC*, 2010.

[4] J. Darmont, F. Bentayeb, and O. Boussaïd. Dweb: A data warehouse engineering benchmark. *Data Warehousing and Knowledge Discovery*, pages 85–94, 2005.

[5] D. Gayo-Avello. A survey on session detection methods in query logs and a proposal for future evaluation. *Information Sciences*, 179(12):1822–1843, 2009.

[6] M. Hagen, B. Stein, and T. Rüb. Query session detection as a cascade. In *CIKM*, 2011.

[7] M. Hagen, B. Stein, and T. Rüb. Query session detection as a cascade. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 147–152. ACM, 2011.

[8] X. Huang, F. Peng, A. An, and D. Schuurmans. Dynamic web log session identification with statistical language models. *Journal of the Association for Information Science and Technology*, 55(14):1290–1303, 2004.

[9] X. Huang, Q. Yao, and A. An. Applying language modeling to session identification from database trace logs. *Knowledge and Information Systems*, 2006.

[10] R. Jones and K. L. Klinkner. Beyond the session timeout: Automatic hierarchical segmentation of search topics in query logs. In *CIKM*, 2008.

[11] O. Kennedy, J. Ajay, G. Challen, and L. Ziarek. Pocket data: The need for tpc-mobile. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 8–25. Springer, 2015.

[12] G. Kul, D. Luong, T. Xie, P. Coonan, V. Chandola, O. Kennedy, and S. 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.

[13] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[14] D. Liu, T. Wang, Y. Wang, Z. Shao, Q. Zhuge, and E. H.-M. Sha. Application-specific wear leveling for extending lifetime of phase change memory in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1450–1462, 2014.

[15] V. H. Makiyama, M. J. Raddick, and R. D. Santos. Text mining applied to SQL queries: A case study for the SDSS SkyServer. In *SIMBig*, 2015.

[16] Open Source. Open source database benchmark OSDB specification, v0.17. *Published at http://osdb.sourceforge.net*, 2004.

[17] Oracle Corporation. Oracle 9ias toplink foundation library guide: Understanding database sessions. *Published at https://docs.oracle.com/*, 2002.

[18] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. 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.

[19] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan. Finding a " kneedle" in a haystack: Detecting knee points in system behavior. In *Distributed Computing Systems Workshops (ICDCSW), 2011 31st*

International Conference on, pages 166–171. IEEE, 2011.

[20] J. Shi, E. Santos, and G. Challen. Why and how to use phonelab. *GetMobile: Mobile Comp. and Comm.*, 19(4):32–38, Mar. 2016.

[21] T. Soikkeli, J. Karikoski, and H. 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.

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

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

[24] Transaction Processing Performance Council. TPC-DS benchmark specification, v2.5. *Published at http://www.tpc.org/tpcds/*, 2017.

[25] Transaction Processing Performance Council. TPCx-IoT benchmark specification draft, v1.5. *Published at http://www.tpc.org/tpcx-iot/*, 2017.

[26] S. Yang, D. Yan, H. Wu, Y. Wang, and A. Rountev. Static control-flow analysis of user-driven callbacks in Android applications. In *International Conference on Software Engineering*, pages 89–99, 2015.