Improving Mobile Search User Experience with SONGO

Emmanouil Koukoumidis[†], Dimitrios Lymberopoulos[‡], Jie Liu[‡], Doug Burger[‡]

[†] Electrical Engineering
Princeton University
Princeton, NJ 08544

[‡] Microsoft Research
One Microsoft Way
Redmond, WA 98052

ABSTRACT

SONGO (**S**earch **ON** the **GO**) is a web search caching system for mobile devices that allows most of the submitted queries to be served locally without having to use the 3G link. Initially, a community-based search cache is generated by mining the most popular queries and links from the mobile search logs. This cache is updated daily making sure that the latest popular information is always available locally on the mobile device. Over time, the community-based cache is incrementally updated with the queries and links that the individual user submits and visits respectively. An analysis of 200 million queries shows that, on average, 66% of the search queries submitted by an individual user can be locally served by caching 2,500 links at the expense of 1MB of flash and 200KB of RAM space. A prototype implementation of SONGO in Windows Mobile demonstrates that a cache hit results into 16x faster responses and 23x energy savings when compared to querying through the 3G link.

1. INTRODUCTION

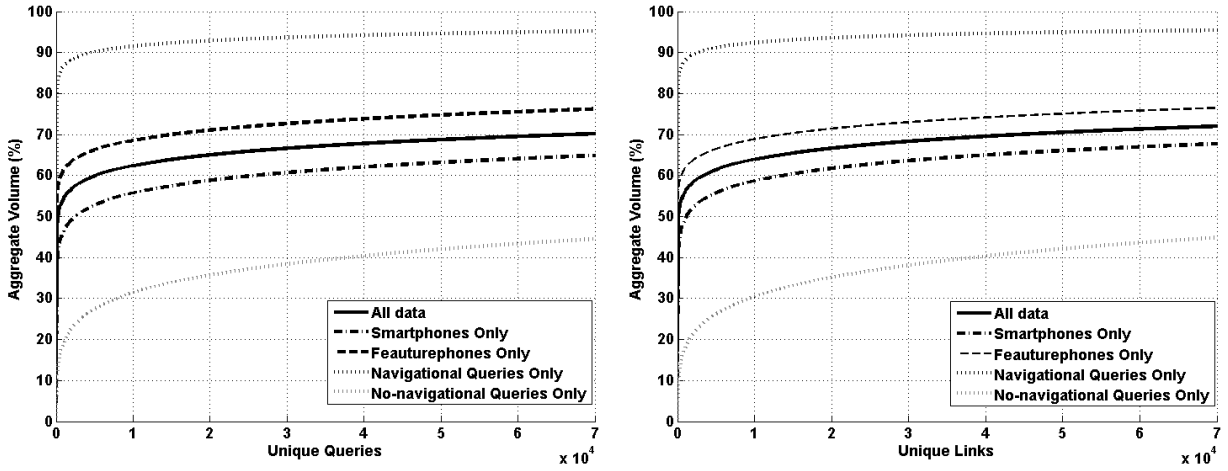
The wide availability of internet access on mobile devices, such as phones and personal media players, has allowed users to search, locate and access web information while on the go. Currently, there are 54.5 million mobile internet users and market analysis shows that this number will increase to 95 million by 2013 [12]. By that time, half of the overall search volume will correspond to queries submitted from mobile devices. As web search becomes a fundamental service for mobile devices, the need to design these devices to better support search services becomes critical.

Currently, mobile web search works in the same way as desktop web search. For every query that is submitted on a mobile device a TCP/IP connection to the datacenter is established, the query is transmitted over the radio link, and the results are computed and sent back to the device over the radio link. Even though this server oriented architecture provides the necessary information to the end user, it is inefficient because it ignores fundamental characteristics and properties of mobile devices and mobile users.

Conversely to the desktop domain where the TCP/IP sessions are established over very fast internet connections, TCP/IP sessions on mobile devices are usually established over slow 3G or Edge links. For instance, the overall time it takes to answer a web search query on a typical high-end smartphone with a 3G connection will on average vary between 5 and 10 seconds. In addition, when the 3G radio is not connected or only Edge connectivity is available, the overall time to serve a search query can be doubled or even tripled. These delays can become frustrating and create a major bottleneck for users that need to quickly access web information in order to make informed decisions while on the go. In addition, mobile devices are usually battery operated and the radio links used to establish TCP/IP connections are the most power hungry components. Actively transmitting or receiving data over the 3G radio can double the power consumed by the phone and thus, it can significantly reduce its battery lifetime. Given the rapidly increasing power consumption of mobile devices and the slow increase in battery capacities, this power bottleneck is likely to remain in place in the future.

On the other hand, the traditional memory bottleneck of mobile devices is rapidly fading away. Phones and personal music players currently support up to 64GB of flash memory and future flash technologies promise capacities that can theoretically reach 2TB in the next two years [1]. The increasingly available memory resources on these devices can really transform the way specific services are structured. For instance, in the case of web search, a large part of the web index could be stored on the phone enabling users to instantly access search results locally without having to reach the search engine.

In this paper we present a new mobile web search cache for mobile devices that takes advantage of the continuously increasing memory resources on mobile devices and the unique way mobile users search the web to overcome, when possible, the performance bottleneck introduced by the 3G link. First, we analyze 200 million queries to understand how mobile users search on their phones. Our data analysis indicates two major observations: (i) there is a small set of queries that is frequently submitted across all mobile users and (ii) individual users frequently repeat queries they have submitted in the past. Given these two observations, we design and implement SONGO, a web search cache that lives on the phone and allows most of the submitted queries to be served locally



(a) (b)

Figure 1: (a) Cumulative query volume as a function of the most popular queries (b) Cumulative link volume as a function of the most popular links.

without having to use the 3G link.

Using a prototype implementation of SONGO and real search query streams extracted from the search logs of `m.bing.com`, we show that SONGO is able to successfully serve, on average, 66% of the web search queries submitted by an individual user. In these cases, the search results are displayed to the user within 400ms which is 16 times faster and 23 times more energy efficient when compared to querying through the 3G link. The resources used by SONGO are approximately 200KB of RAM and 1MB of Flash, less than 1% of the memory resources available on most smartphones today.

Not only does SONGO improve the speed and energy efficiency of mobile search but it also improves the quality of the search results by enabling *personalized ranking* on the phone. Since SONGO lives on the phone, it is able to monitor *what*, *when* and *how many times* the specific user clicks on a link after a given query. This information, in association with the information mined from the search logs, is used by SONGO to provide personalized ranking of search results.

The rest of the paper is organized as follows: Section 2 presents our findings on how mobile users search the web based on the analysis of several consecutive months of web search logs. Section 3 provides an overview of the SONGO architecture and a detailed description of its different components. Section 4 presents the evaluation of SONGO with real user search query streams extracted from the mobile search logs. Section 5 provides an overview of the related work and Section 6 concludes the paper.

2. MOTIVATION

Caching search results on mobile devices offers the obvious advantage of serving search queries extremely fast. However, the real impact of such a caching system depends on the fraction of the query volume that can be successfully served locally on the device. A small fraction would suggest that the value of such a cache is negligible. Of course, how small or large this fraction is depends on how mobile internet users search the web.

To answer these questions we analyzed 200 million queries, submitted to m.bing.com, over a period of several consecutive months in 2009. The query volume consisted of web search queries submitted from mobile devices such as phones and personal music players. Every entry in the search logs we analyze contains the raw query string that was submitted by the mobile user as well as the link[1] that was selected as a result of the submitted query. No personal information (i.e. location, carrier, phone model etc.) is included in the logs.

Our current analysis and the proposed cache focus on web search queries only. Local search queries where users try to find information about businesses around them are not taken into account in this work. The reason is that the vast majority of mobile search queries today (approximately 70%) are web search queries. Our future work will focus on the smaller but still important part of local search queries.

2.1 The Mobile Community Effect

First, we look at the community of mobile users as a whole. From the web search logs, we extract the most popular queries submitted and the most popular links clicked by the mobile users as a result of any query submission. Figures 1(a) and 1(b) show the cumulative query and link volume as a function of the number of most popular queries and links respectively. When looking across all data, it turns out that the 6000 most popular queries and the 4000 most popular links are responsible for approximately 60% of the query and link volumes respectively. *In other words, there is a small set of queries and links that is extremely popular across all mobile users*. This suggests that if we store these 6000 queries and

[1]A *link* uniquely identifies a *search result* that also contains a brief summery of the web page. In this paper, we consider the terms *link* and *search result* interchangeable.

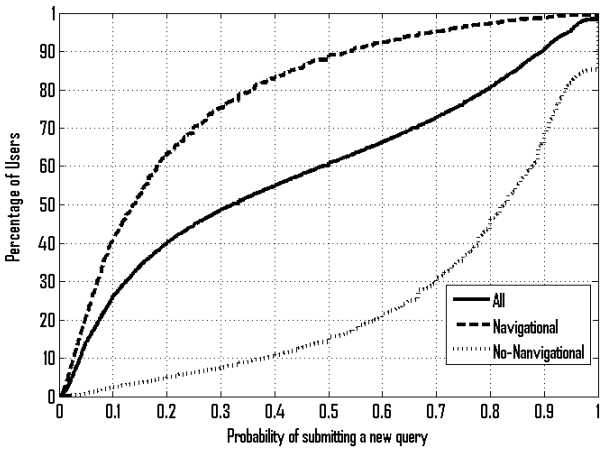


Figure 2: Repeatability of mobile search queries. For every unique user, a 1-month long search query stream was extracted from the search logs and used to compute the repeatability of queries.

4000 links locally on the phone we could theoretically answer 60% of the overall queries submitted by mobile users without having to use the radio link.

We further divide the queries in two different categories, navigational (e.g. "youtube" or "facebook") and no-navigational (e.g. "michael jackson") and we study the same trends in isolation. As Figure 1 shows, both query types follow the same trends but navigational queries are significantly more concentrated compared to no-navigational queries. For instance, the first 5000 navigational queries are responsible for 90% of the navigational query volume while the same number of no-navigational queries accounts for less than 30% of the no-navigational query volume.

Another interesting observation comes from comparing the results between Figures 1(a) and 1(b). Note that in order to achieve a cumulative volume of 60%, 50% more queries are required compared to the number of links (6000 queries vs 4000 links). By a more careful investigation and manual inspection of the search logs, we concluded that this is due to the fact that users search for the same web page in many different ways. For instance, it turns out that mobile users tend to very often either misspell their queries because of the small and inaccurate keyboards they have to interact with (i.e. "yotube" instead of "youtube") or purposely change the query term so that they can still get to the result they want without having to type a lot of characters (i.e. "utube" instead of "youtube"). However, even though a misspelled or altered query is submitted, the search engine is intelligent enough to provide the user with the correct search result and thus a successful click trough is recorded. Because of this, a popular webpage is, in general, reached through multiple search queries.

Figures 1(a) and 1(b) also show the same information when considering the queries and links that were submitted by feature-phone[2] and smartphone users in isolation. Even though the exact same observations hold in both cases, queries and links that are accessed over feature-phones are in general about 10% more concentrated when compared to smartphones. Of course, this is an artifact of the limited user interfaces found on these devices. The lack of a full browser makes accessing the web a challenging task, limiting user interaction with the phone.

2.2 The Individual Mobile User Effect

Figure 1 shows that there is a very small set of queries and links that is popular across all users and it is responsible for approximately 60% of the cumulative mobile query volume. However, there is another 40% of the mobile query volume that seems to be spread across a very large number of queries and links. Even though queries in this 40% might not be statistically significant for all mobile users, they might be extremely important for individual users.

To find out how we can possibly achieve this, we studied how often individual users repeat queries. We call a query a *repeated query* only if the user submits the same query *and* clicks on the exact same link. Figure 2 shows the percentage of individual mobile users as a function of the probability of submitting a new query within a month. Approximately 50% of mobile users will submit a new query *at most* 30% of the time. *In other words, at least 70% of the queries submitted by half of the mobile users are repeated queries*. Figure 2 also shows this trend for both navigational and no-navigational queries. Approximately 80% of mobile users repeat a navigational query *at least* 65% of the time. In the case of no-navigational queries, 20% of the users repeat the same query *at least* 40% of the time.

Consequently, knowing what the user searched in the past provides a very good indication of what the user will search for in the future. In essence, users consider the search box as the most natural interface for accessing the web. Current techniques for providing quick access to web pages on mobile devices, such as the browser address bar and bookmarks, are not used by mobile users. The search box today is replacing those interfaces. By simply monitoring the queries submitted on a phone, one can automatically populate the bookmarks of the mobile user and instantly display these bookmarks while the user is entering a search query.

3. SONGO ARCHITECTURE

Given the mobile search trends that the analysis of the web search logs highlighted, we designed and implemented SONGO, a mobile search cache that lives on the phone and it is able to answer queries locally without having to use the 3G link. SONGO consists of two discrete but strongly interrelated components (Figure 3). First, the community-

[2]Feature-phones are devices with limited internet access capabilities in the sense that they don't have a fully capable web browser such as smart-phones like iphone, android and windows mobile devices.

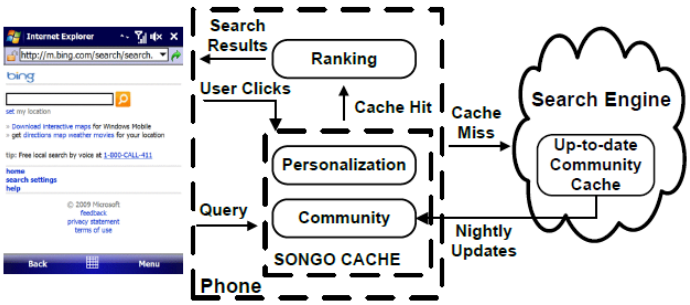


Figure 3: Overview of the SONGO web search cache.

based part of the cache is responsible for storing the small set of queries and links that are very popular across all mobile users. This information is automatically extracted from the search logs and is updated overnight every time the mobile device is recharging, making sure that the latest popular information is available on the mobile device. The personalization part of the cache monitors the queries entered as well as the links clicked by the user and performs two discrete tasks. First, it expands the cache to include all those queries and links accessed by the user that do not initially exist in the community part of the cache. In that way, the cache can take advantage of the repeatability of the queries submitted by the mobile users to serve as many queries as possible locally on the mobile device. Second, it collects information about user clicks, such as when and how many times the user clicks on a link after a query is submitted, to customize ranking of search results to user's click history.

When a query is submitted, SONGO will first perform a lookup in the cache to find out if there are locally available search results for the given query. In the case of a cache hit, the search results are fetched from the local storage, ranked based on the past user access patterns recorded by the personalization part of the cache, and immediately displayed to the user. In the case of a cache miss, the query is submitted to the search engine over the 3G radio link.

Realizing the architecture shown in Figure 3 on an actual mobile device poses several challenges:

Content Generation: A methodology is required to decide which and how many queries as well as which and how many search results for every query should be included in the cache. Ideally, we would like to maximize the number of queries that can be served by the cache while minimizing its impact on the memory resources of the mobile device.

Storage Architecture: An efficient way to store and quickly retrieve the search results on the mobile device is needed. Memory overhead, in terms of both RAM and flash, should be minimized to prevent performance degradation on the device and provide ample storage space for user's personal files. At the same time, SONGO should be able to quickly locate and retrieve the search results to minimize user response time.

Cache Management: A scalable mechanism for regularly updating the cache contents is required. The set of popular queries and links that appear in the search logs changes over time. Having available the most up-to-date set of popular queries and links on the phone is necessary for maximizing the number of queries that can be served by SONGO.

Query	Link	Volume
myspace	`http://mobile.myspace.com`	1,000,000
myspace	`http://www.myspace.com`	950,000
facebook	`http://m.facebook.com`	900,000
...	...	...
youtube	`http://m.youtube.com`	500,000
yotube	`http://m.youtube.com`	300,000
...	...	...
weather	`http://m.weather.com`	200,000
...	...	...
Total Volume		100,000,000

Table 1: A list of query-link pairs sorted by their volume is generated by processing the mobile web search logs over a time window(i.e. a month). The volume numbers used in this table are hypothetical.

Personalized Ranking: The user's access patterns can provide important information about the individual user and its interests. Initially, the cache contains search results obtained by mining the web search logs. Over time, individual users will access a portion of this information and of course add more search results in the cache based on the specific queries they submit. Recording and leveraging this information over time to personalize the search experience provides a great opportunity and challenge.

In the next sections we describe how SONGO addresses these challenges in detail.

3.1 Cache Content Generation

The search results stored in our cache are extracted directly from the mobile search logs in a data-driven way. The goal of this process is to identify the most popular queries and search results that are of interest to the whole mobile community.

Table 1 shows an example of the type of information extracted from the search logs. A set of triplets of the form: $< query, link, volume >$ is generated and sorted based on *volume*. The term *query* corresponds to the exact query string submitted to the search engine, the term *link* corresponds to the search result that was selected after entering the particular query, and the term *volume* represents the number of times in the search logs that the specific *link* was selected after entering the query string *query*. For instance, the first row in Table 1 can be interpreted as follows:

In the last month, there were 1 million searches where the link `http://mobile.myspace.com` *was selected after the query "myspace" was submitted.*

Ideally, we would like to use all the triplets in Table 1 to generate our cache contents. This approach would maximize the query volume that can be served locally on the phone. However, the unique *query-link* pairs in the mobile web search logs can be in the order of tens or even hundreds of millions. Storing all of them would require significant

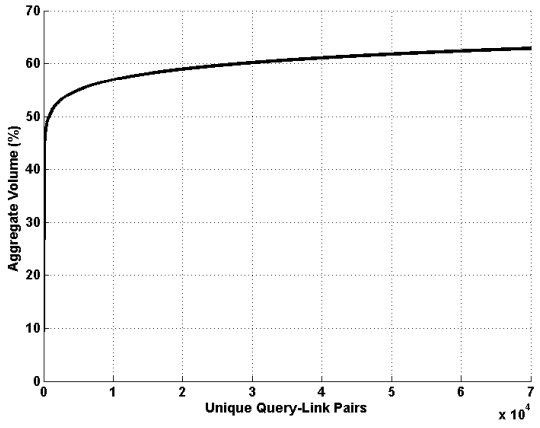


Figure 4: Cumulative query-link volume as a function of the most popular query-link pairs.

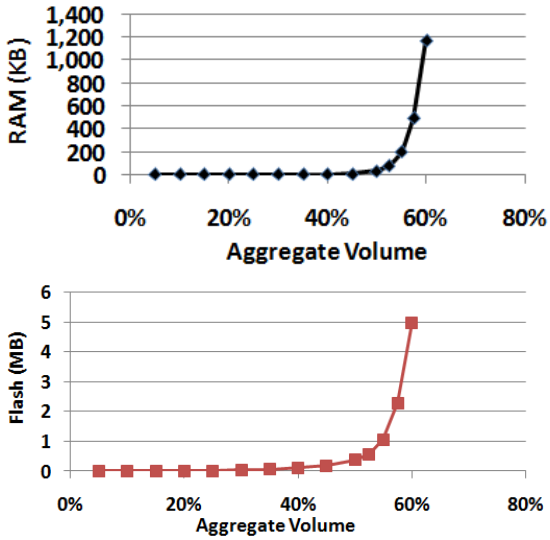


Figure 5: Memory and flash overhead introduced by SONGO for different *query-link* aggregate volumes.

memory resources that a phone might not be able to provide or a user might not be willing to sacrifice. Thus, we have to carefully decide *which* and *how many* entries in Table 1 will be cached on the phone.

Deciding *which* entries to store is straightforward. To maximize the query volume that can be served by our cache, we should always store the most popular *query-link* pairs or, in other words, the top entries of Table 1. The higher the volume of a *query-link* pair the higher the probability that a given user will try to access this *query-link* pair on his phone.

Deciding *how many* of the most popular *query-link* pairs to store is a more complicated process. We select the number of *query-link* pairs to cache based on either a memory or cache saturation threshold.

Memory (Flash or RAM) Threshold: Starting from the top entry in Table 1, we run down though its entries and continuously add *query-link* pairs until a specific flash or RAM memory threshold M_{th} is reached. This threshold represents the maximum memory that can be allocated to our cache and it can be set by either the phone itself based on its available memory resources or by the user, depending on how much storage space and memory he is willing to sacrifice for SONGO. The memory and storage requirements can be easily computed for a given set of entries in Table 1 based on the number of unique queries and links they contain (see Section 3.2).

Cache Saturation Threshold: Starting from the top entry in Table 1, we run down though its entries and continuously add *query-link* pairs until we reach a *query-link* pair with a *normalized volume* lower than a predetermined threshold V_{th}. The *normalized volume* of a *query-link* pair is generated by dividing this pair's volume by the total volume of all *query-link* pairs in the search logs. For instance, the *normalized volume* of the first query-link pair in Table 1 is equal to: $10^6/10^8 = 0.01$. Since the entries in Table 1 are sorted based on their volume, the *normalized volume* of *query-link* pairs is monotonically decreasing.

The value of the cache saturation threshold is illustrated in Figure 4. From the empirical CDF of the volume of all *query-link* pairs in the search logs, it is apparent that the value of adding *query-link* pairs to our cache quickly diminishes. In particular, after having cached approximately 20000 *query-link* pairs, even marginally increasing the aggregate volume requires a large number of *query-link* pairs. For instance, slightly increasing the aggregate volume from 58% to 62% requires to double the amount of *query-link* pairs from 20000 to 40000.

In practice, our mobile web search logs analysis has shown that the cache saturation threshold will be quickly reached before SONGO stretches the memory or storage resources available on the phone. This can be clearly seen in Figure 5 that shows the size of main memory and flash required by SONGO as a function of the aggregate *query-link* volume represented by the *query-link* pairs stored in the cache. It is clear that the saturation point of the cache is quickly reached when the most popular *query-link* pairs that correspond to approximately 55% of the cumulative *query-link* volume have been cached. At this point, our cache requires approximately 1MB of flash and 200KB of main memory, which accounts for less than 1% of the available memory and storage resources on a typical smartphone.

Independently of which threshold is used (memory or cache saturation), this methodology identifies the n top entries in Table 1. Before using these entries to build the SONGO cache, we use the volume information for every *query-link* pair to produce its ranking score. In essence, the score for each *query-link* pair is produced by normalizing the volume across all links that correspond to the query. For instance, in the case of query "myspace" in Table 1, the ranking score for the link `http://mobile.myspace.com` is equal to $10^6/1.95 * 10^6 = 0.513$ and the score for `http://www.myspace.com` is $95 * 10^4/1.95 * 10^6 = 0.487$.

The generated $< query, link, score >$ triplets can now be used to build the cache on the phone.

3.1.1 Advantages of the approach

Extracting SONGO's cache contents directly from the mobile search logs provides several advantages.

First, not only do we store the most popular queries across all mobile users, but we also store only the most popular search results for every query. Even though there might be tens or even hundreds of search results available for a given query, we only cache these search results that are popular across all mobile users. This approach constraints the amount of memory resources required. For instance, for a typical cache (Section 4) we store on average 1.5 search results per query. This can result into 3 to 5 times lower memory overhead when compared to simpler approaches where the top 5 or 10 search results for every popular query are cached.

Second, each query and search result pair extracted from the search logs is associated to a volume. By caching the volume (or the normalized volume across all pairs in the cache) of every query-link pair, we enable the phone to rank search results for a given query locally without the need to reach the search engine over the 3G link.

Third, by processing the mobile search logs we automatically discover the most common misspellings of popular queries. Given the cumbersome text input interfaces on phones, mobile users tend to frequently misspell their queries. These frequent misspellings appear in the search logs as popular queries, enabling SONGO to cache search results for all these query misspellings. As a result, queries such as "yotube", "facebok", "utube", and "yaho" can now be served locally on the phone.

3.2 Storage Architecture

The set of $< query, link, score >$ triplets identified with the methodology described in the previous section must be efficiently stored on the phone. Storage efficiency is defined in two ways. First, the main memory and flash resources required to store the search results should be minimal. Second, the time it takes to retrieve, rank and display search results after the user enters a query should be as low as possible.

Figure 6 provides an overview of SONGO's storage architecture. It consists of two components, a hash table and a custom database of search results. The hash table lives in main memory and its role is to link queries to search results. This is done in three steps. First, given a submitted query the hash table can quickly identify if there are cached search results for the query and thus, if we have a cache hit or a cache miss. Second, in the case of a cache hit, the hash table provides pointers to the database where the search results for the submitted query are located. Third, along with each search result pointer, the hash table provides its ranking score, enabling SONGO to properly rank search results on the phone before displaying them to the user.

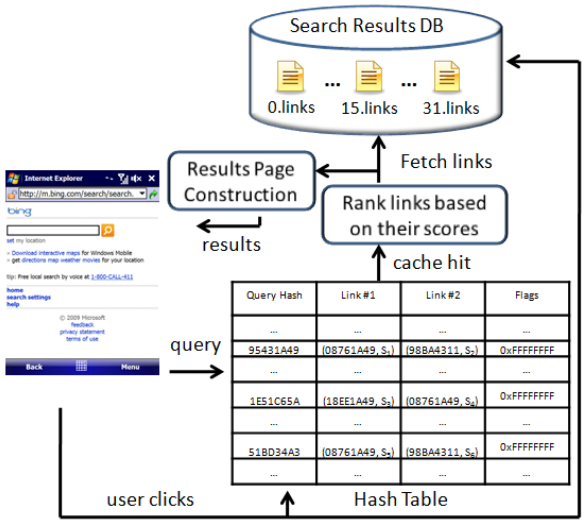


Figure 6: Overview of SONGO's storage architecture. The hash table lives on the memory and it links queries to search results. The search results are stored in a custom database stored in flash.

The custom database of search results lives in flash and its role is to store all the available search results so that they occupy the least possible space and they can be quickly retrieved. The information stored in the database for each search result includes all the necessary information for generating the same search user experience with the search engine: the actual web address, a short description of the website and the human readable form of the web address.

Over time and as the user submits queries and clicks on search results, SONGO updates both the hash table and the database of search results. First, every time the user clicks on a search result, its ranking score is properly updated in the hash table. In addition, if a new query or a new search result is selected that does not exist in the cache, both the hash table and the database are properly updated so that this query and search result can be retrieved form the cache in the future.

3.2.1 Query Hash Table

Figure 7 shows the structure of the hash table used to link queries to search results. Every entry in the hash table corresponds to one and only one query and has 4 fields. The first field of every entry contains the hash value of the query string that this entry corresponds to. The next two fields are of identical type and represent two search results associated to the query (Link #1 and Link #2 in Figure 7). Each search result in the hash table is represented by a pair of numbers. The first number corresponds to the hash value of the web address of the search result. This value is used to uniquely identify a search result and, as it will be described in the next section, is used as a pointer to retrieve the information associated to the search result (short description, web address etc.) from the database. The second number corresponds to

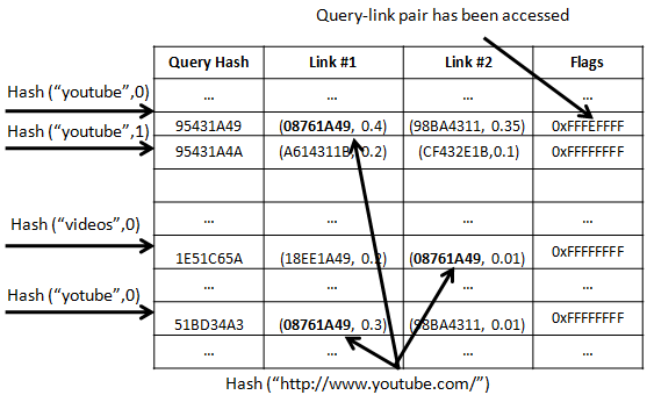


Figure 7: The hash table data structure used to link queries to search results.

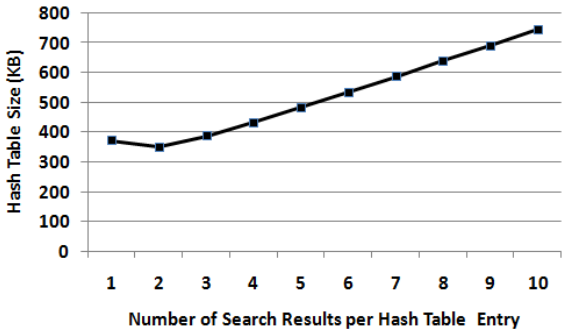


Figure 8: The memory footprint of the hash table as a function of the number of search results per hash table entry. For all data points we used exactly the same $< query, link, score >$ triplets that correspond to 60% of the cumulative query-link volume (Figure 4).

the ranking score of the search result. The ranking score of every *query-link* pair is represented by its normalized volume. The last field of each entry in the hash table is a 64-bit number that is used to log information about the two search results in this entry(the 32 most significantly bits correspond to the first search result while the 32 least significant bits correspond to the second search result). Currently, we use only one bit to indicate if the user has ever accessed the specific *query-link* pair. The rest of the flag bits are reserved for future purposes.

For instance, consider the first entry in the example hash table shown in Figure 7. This entry corresponds to the query "youtube" and thus, its first field contains the hash value of the string "youtube". The most popular search result for this query points to the web address `http://www.youtube.com` and therefore the second field of this entry contains the hash value of this web address along with its ranking score. Similarly the third field contains the same information for the next most popular search result for this query. From the value of the bit flags, it is also clear that the user has already submitted the query "youtube" on his phone and he has only clicked on the first search result immediately after.

In general, given a set of $< query, link, score >$ triplets the hash table is generated as follows. For every unique query in the set of triplets we identify all the links associated to this query. An entry is created in the hash table for the query and search results are added in descending order of score. If more than two links are associated to the same query, additional entries are created in the hash table by properly setting the second argument of the hash function (i.e. "youtube" query in Figure 7).

This approach of linking queries to search results highlights two important design decisions that were influenced by the properties of the $< query, link, score >$ triplets extracted from the mobile web search logs. First, the number of search results linked to a query in a given hash table entry is critical. If most of the queries are associated to a large number of search results, then using a small number of search results per hash table entry could lead to a large number of hash table entries and thus to higher memory requirements. On the other hand, choosing a hash table entry size that can support a large number of search results when most of the queries are associated to a few search results, could also lead to wasting memory resources. These trends are illustrated in Figure 8 that shows the memory footprint of the hash table for different numbers of search results stored per hash table entry. Note that the smallest memory footprint is achieved when two search results are stored per hash table entry. The reason is that every query in the $< query, link, score >$ triplets corresponds, on average, to 1.5 links.

Second, the way queries are linked to search results can impact the storage requirements of the database of search results. The simplest and fastest approach to retrieving and displaying search results to the user would be to store them in a single HTML file. Even though this approach would simplify the structure of our hash table, it would significantly increase the flash memory required to store them.

The reason is that most of the search results are shared across a large number of queries. For instance, in the example hash table shown in Figure 7, the search result that corresponds to the web address `https://www.youtube.com` is linked to three different queries. In general, our analysis indicates that only 60% of the search results that appear in the $< query, link, score >$ triplets are unique. If a single search result page were to be stored for every query, then 40% of the search results would have to be stored at least twice. To avoid wasting flash resources, we opted to store each search result once and then link individual queries to each search result independently. The overhead of this approach is that the search result web page for every query has to be constructed on the fly after parsing the hash table to retrieve all the search results for the query. However, as it will be shown in Section 4, the overhead introduced by this approach is in the order of tens of milliseconds and thus does not impact the overall user experience.

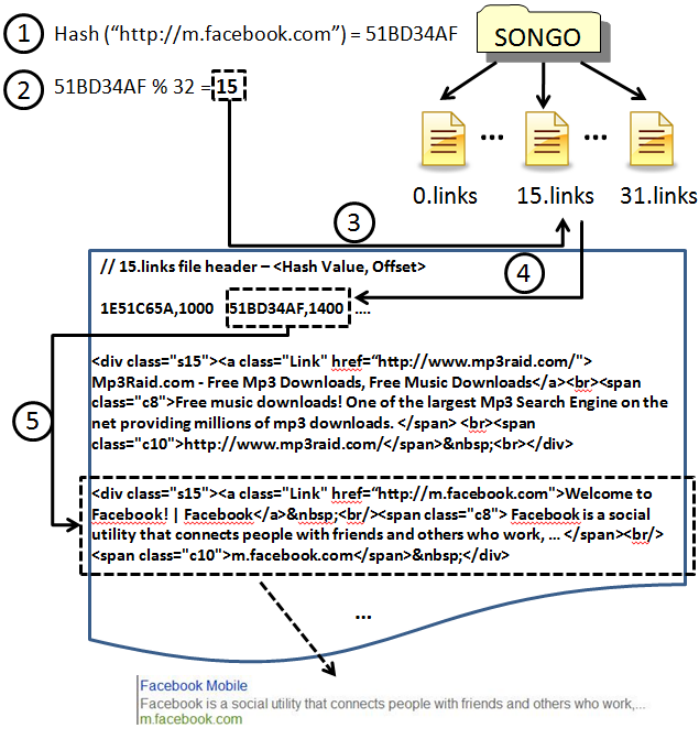


Figure 9: Search result database structure and illustration of the process of retrieving a search result when $N = 32$ files are used.

3.2.2 *Search Results Organization*

The custom database of search results lives in flash and stores all the links that appear in the $< query, link, score >$ triplets extracted from the mobile web search logs. For every search result, we store its title that serves as the link to the landing page, a short description of the landing page and the human readable form of the link (Figure 9). This allows SONGO to display a search result in exactly the same way as the search engine would and thus, to provide a transparent user experience.

The amount of memory required to store this information for a search result is on average 500 bytes. However, the actual memory space required might be significantly higher due to the internal structure of flash chips. Flash memories are organized in blocks of fixed size that are usually equal to 2KB, 4KB or 8KB depending on the size of the chip. This means that even small files with size less than the size of a flash block, will appear to occupy a full block. For instance, if we store a 500 bytes file containing a single search result in flash memory, then this file will occupy 4, 8 or 16 times more flash space (when 2KB, 4KB or 8KB of block size is used respectively) than its actual size.

In order to avoid flash fragmentation, multiple search results should be aggregated and stored into as few files as possible. However, storing a large number of search results into a single file could increase the time it takes SONGO to locate and retrieve a search result and, thus it could hurt the response time of our cache. As a result, the way search results are aggregated into files and organized within a file is critical for minimizing both, flash fragmentation and cache response time.

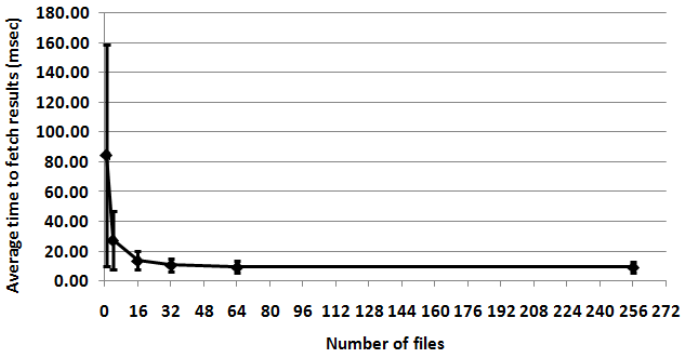


Figure 10: Average time to retrieve two search results from the database as a function of the number of files used to store the search results. The vertical bars represent the deviation of the access time over 10 consecutive experiments.

For now, assume that the number of database files used in SONGO is N. Figure 9 shows how search results are organized within a database file. Each search result is assigned to one of the N files based on the hash value of its web address. In particular, the remainder of the division of the hash value with the number of files in our database (a number between 0 and $N - 1$) is used to identify the file where the search result should be stored (Figure 9).

The first line in each of the N database files contains pairs of the form $(hash\,value,\ offset)$. The *offset* represents the actual offset from the beginning of the file where the information for the search result represented by the *hash value* is located. By parsing the first line of a database file we can identify where each search result stored in this file is located. Note that search results within a database file are not sorted based on their hash value or ranking score. Instead, whenever the user clicks on a search result that is not already cached, SONGO will add the search result at the end of the database file and the header of this file is augmented with the $(hash\,value,\ offset)$ pair for this search result. This allows SONGO to easily update the search results database over time.

To determine the best number N of database files that can efficiently balance cache's response time and flash fragmentation, we measured the performance of our database architecture for different number of files on a cache that stores approximately 6000 search results. Figure 10 shows the average time it takes to retrieve two search results from the database when different number of files is used. In general, the smaller the number of files used, the lower the impact of flash fragmentation will be. Note, that when the number of files is anywhere between 32 and 256 the cache's response time is low and almost identical. However, when less than

32 files are used, the average time to fetch two search results and its deviation seems to increase exponentially. This indicates that 32 files is the smallest number of files that can be used to store search results without increasing the overall user response time. When using 32 files, the amount of memory that can be wasted due to flash fragmentation can never exceed 64KB, 128KB, or 256KB depending on the flash block size used (2KB, 4KB and 8KB respectively).

3.3 Cache Management

The contents of the personalization and community parts of the cache are updated over time. First, the personalization part monitors user clicks over time and adjusts the ranking score of individual search results to reflect the history of user clicks. Second, the community part of the cache periodically connects to the server to obtain the latest set of popular queries and links in the mobile search logs, making sure that the most up-to-date information is always available locally.

3.3.1 Personalized Ranking

By monitoring user clicks over time, the personalization part of the cache is aware of *when* and *how many times* the user selects a link after a given query is submitted. SONGO uses this information to incrementally update the ranking score of the cached search results to offer a personalized search experience.

Assume that for a query Q there are two search results R_1 and R_2 available in the cache. Every time the user submits the query Q and clicks on the search result R_1, SONGO updates the scores S_1 and S_2 for the two search results R_1 and R_2 respectively, as follows:

$$S_1 = S_1 + 1 \quad (1)$$

$$S_2 = S_2 * e^{-\lambda} \quad (2)$$

The ranking score of the selected search result is increased by 1, the maximum possible score of a search result extracted from the mobile search logs, (Equation (1)). In that way, we always favor search results that the user has selected. Note that if this search result did not initially exist in the cache (selected after a cache miss), then a new entry in the hash table is created that links the submitted query to the selected search result and its score becomes equal to 1. At the same time, the ranking score for the unselected search result is exponentially decreased [3] (Equation (2)). This enables SONGO to take into account the freshness of user clicks. For instance, if search result R_1 was clicked 100 times one month ago and search result R_2 was clicked 100 times during last week, then the ranking score for R_2 will be higher.

Using Equations (1) and (2), the ranking score of the search results, at any given time, reflects both the number and freshness of past user clicks. In practice, any personalization ranking algorithm could be used [19],[18] with the proposed cache.

[3]The parameter λ is used to control how fast the ranking score is decayed.

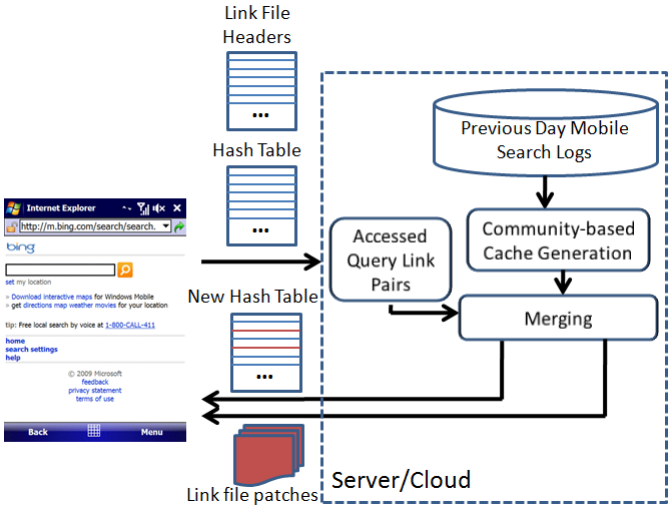


Figure 11: The community part of the cache is periodically updated with the latest set of popular queries and links in the mobile search logs.

3.3.2 Cache Updates

Figure 11 provides an overview of the mechanism used to update the community part of the cache. The phone transmits to the server its current version of the hash table along with the headers of the 32 files in the database of search results. The server first runs through the hash table and removes all the *query-link* pairs that have not been accessed by the user in the past. This can be easily done by examining the *flags* column in the hash table (Figure 7). The *query-link* pairs that have been accessed by the user in the past are never removed from the cache because of the high repeatability of mobile queries.

At the same time, the server periodically (e.g. daily or every several hours) extracts the most popular queries and links from the mobile search logs as described in Section 3.1, and adds them to the hash table [4]. During this process, conflicts might arise in the sense that a *query-link* pair that already exists in the hash table (previously accessed by the user) might re-appear in the popular set of queries and links extracted on the server. The conflict is caused when the ranking score stored in the hash table is different from the new ranking score computed on the server based on the search log analysis. SONGO resolves these conflicts by always adopting the maximum ranking score.

After the hash table has been updated, the server uses the 32 header files to create the necessary patch files for the database files that live on the phone. The new hash table and the 32 patch files are transmitted to the phone and the new cache becomes available to the user. Note that the amount of data exchanged between the phone and the server will usually be less than 2.5MB given that SONGO requires, on av-

[4]The community part of the cache is generated on the server once for all users. Conversely, the merging phase in Figure 11 is repeated for every user.

Figure 12: The graphical user interface of our prototype implementation. In this example, cached search results for the query "facebook" were displayed in 369ms.

erage, approximately 200KB for storing the hash table and 1MB for storing the search results (Figure 5). This amount can be further reduced by leveraging data compression techniques.

4. EVALUATION

Our evaluation focuses on answering two major questions about the proposed mobile web search cache: (i) *How fast and energy efficiently can SONGO serve search queries?* and (ii) *What fraction of the query volume that a typical user submits can be served by SONGO?*

First, we use our prototype implementation to quantify the amount of time and energy required to serve a search query through SONGO and compare its performance to that of the different radio links available on the phone. Second, we extract anonymized search query streams from the mobile web search logs and run them against the SONGO cache to quantify what fraction of the query volume of an actual user can be served locally on the phone.

4.1 Cache Hit Performance

All of the measurements presented in this section were acquired using our prototype SONGO implementation on a Sony Ericsson Experia X1a cell phone [17] running Windows Mobile 6.1 (Figure 12). An embedded internet explorer object allows us to display search results provided by either the phone (cache hit) or the search engine (cache miss). Also, since the internet explorer object is managed by the SONGO application, it allows us to intercept user clicks after a query is submitted and thus, properly personalize the web search cache on the phone.

To measure how fast and energy efficiently search queries are served using our cache and the different radios available on the phone, we randomly selected 100 different queries for which cached search results were available. In every experiment, each of the 100 queries was submitted 100 times through a lightweight test script we embedded in our SONGO application. For every query submission the user response time and the overall energy dissipation of the phone during that time were recorded.

User response time is defined as the elapsed time from the moment that the query is submitted (search button is clicked in Figure 12) to the moment that the embedded internet explorer object in our application has completed rendering the search results web page. User response time was measured using the sub-millisecond timing mechanisms available in Windows Mobile. The energy consumed due to every query submission was measured by connecting the phone to a Monsoon power monitor device [13].

The same experiment was repeated once for every radio available on the phone (Edge, 3G and 802.11g) and of course for the SONGO cache. In each experiment all the queries were served in one and only one way (SONGO cache, 3G, Edge or 802.11g). In the experiments where a radio was used to serve search queries, we made sure that this radio was always properly associated to a cell tower or an access point, so that no additional delays were introduced by radio connection times. In the experiments where the SONGO cache was used to serve queries, a cache containing all the *query-link* pairs that account for 55% of the cumulative *query-link* volume over a period of several months was used (Figures 4 and 5). This cache included approximately 2500 search results occupying 1MB of flash space as described in Section 3.2.2.

4.1.1 User Response Time

Figure 13 shows the average user response time per query when the SONGO cache or one of the radios on the phone is used. *On average, SONGO is able to serve a query 16 times faster than 3G, 25 times faster than Edge and 7 times faster than 802.11g*. Note that even though 802.11g can provide a low user response time that is slightly higher than 2 seconds, it has a major drawback. Due to its high power consumption, 802.11g will be rarely turned on and connected to an access point on a continuous basis. As a result, in practice, 802.11g is not immediately available and extra steps that introduce significant delay and unnecessary user interaction are required.

Table 2 shows the breakdown of SONGO's user response time in the case of a cache hit. From the 378ms it takes SONGO to serve a query, 96.7% of the time is spent at the browser while rendering the search results web page. The time it takes our cache to locate and retrieve search results is less than 20ms and it accounts for only 3.3% of the overall user response time.

Furthermore, the time it takes SONGO to look up its hash table and determine if a query is a cache hit or a cache miss is only $10\mu s$ (Table 2). As a result, in the case of a cache miss, the overall user response time will be increased by $10\mu s$. This is an infinitesimal increase given that any radio on the phone requires several seconds to serve a search query.

4.1.2 Energy Consumption

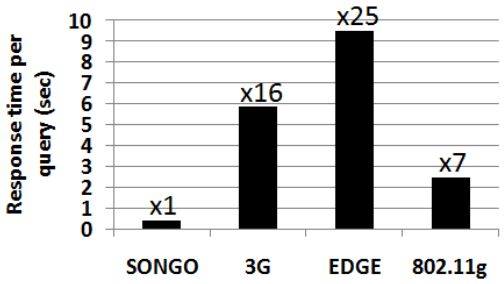


Figure 13: Average user response time per query when SONGO and different radio links are used to serve search queries.

Operation	Average Time (ms)	Percentage
Hash Table Lookup	0.01	$\approx 0\%$
Fetch Search Results	10	2.7%
Browser Rendering	361	96.7%
Miscellaneous (function calls etc.)	7	1.7%
Total	378	100%

Table 2: SONGO's user response time breakdown

Figure 14 shows the average energy consumed by thephone per query when the SONGO cache or one of the radios on the phone is used. *SONGO is on average 23 times more energy efficient than 3G, 41 times more energy efficient than Edge and 11 times more energy efficient that 802.11g*. Note that the gap in the energy efficiency between SONGO and the different radios on the phone is larger than the corresponding gap in the user response time shown in Figure 13. This is because SONGO conserves energy in two ways. First, no data is being transmitted or received in the case of a cache hit, and thus the overall power consumption of the phone remains low. Second, since SONGO achieves a user response time that is an order of magnitude lower compared to when the radios on the phone are used, the per query energy dissipation is significantly lower for SONGO.

4.2 Cache Hit Rate Performance

To quantify the cache hit rate achieved by SONGO for a typical user, we used anonymized search query streams from the mobile web search logs. To ensure a representative and unbiased selection of search query streams, we classified users in 4 different classes based on their monthly query volume. Table 3 shows the different user classes and the percentage of users in the mobile search logs that belongs to each class. Note that we ignore users that submit less than 20 queries per month. This is done for two reasons. First, SONGO is targeting users that frequently access the internet and search the web. Second, as higher-end smartphones with advanced browsing capabilities become more and more available, the average monthly query volume submitted by individual users will increase beyond the threshold of 20 queries per month.

For the experiments described in this section, we randomly

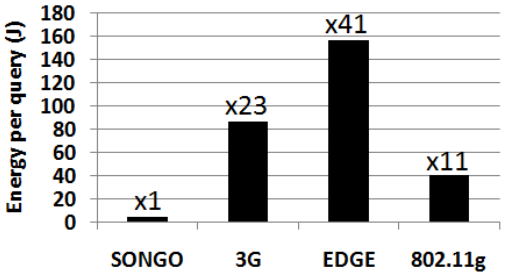


Figure 14: Average energy per query when SONGO and different radio links are used to serve search queries.

User Class	Monthly Query Volume	% of Users
Low Volume	[20,40)	55%
Medium Volume	[40,140)	36%
High Volume	[140,460)	8%
Extreme Volume	[460,∞)	1%

Table 3: Classes of users and their characteristics. 100 anonymized search query streams from each class of users were used to quantify the cache hit rate achieved by SONGO.

selected 100 anonymized users from each class shown in Table 3 and extracted their search query streams from the mobile web search logs over a period of one month. Each of the 400 search query streams was replayed against the SONGO cache that was generated using the mobile search logs over the preceding month [5]. The resulting cache contained approximately 2500 links that corresponded to 55% of the cumulative *query-link* volume in the search logs.

4.2.1 Hit Rate Results

Figure 15 shows the average hit rate for each user class described in Table 3. On average, 65% of the queries that an individual user submits are cache hits. As a result, 65% of a user's query volume can be served 16 times faster. By comparing the cache hit rate across the 4 user classes, it is apparent that the cache hit rate seems to increase with the monthly query volume of the user. SONGO achieves a cache hit rate of approximately 60% for the low volume class which immediately jumps to 70% for the medium volume class and to 75% for the high and extreme volume user classes.

To better understand how the community and personalization components of the SONGO cache contribute to the overall cache hit rate, Figure 15 also shows the average cache hit rate for every user class in the cases where SONGO is using only either the community or personalization part of the cache. When SONGO uses only the community-based part of the cache, new queries and search results selected by the user are not cached over time and therefore, the cache cannot take advantage of the repeatability of mobile queries. When SONGO uses only the personalization part of the cache, the

[5]Note that no data from the one month period from which the 400 query streams were extracted was used when building the SONGO cache.

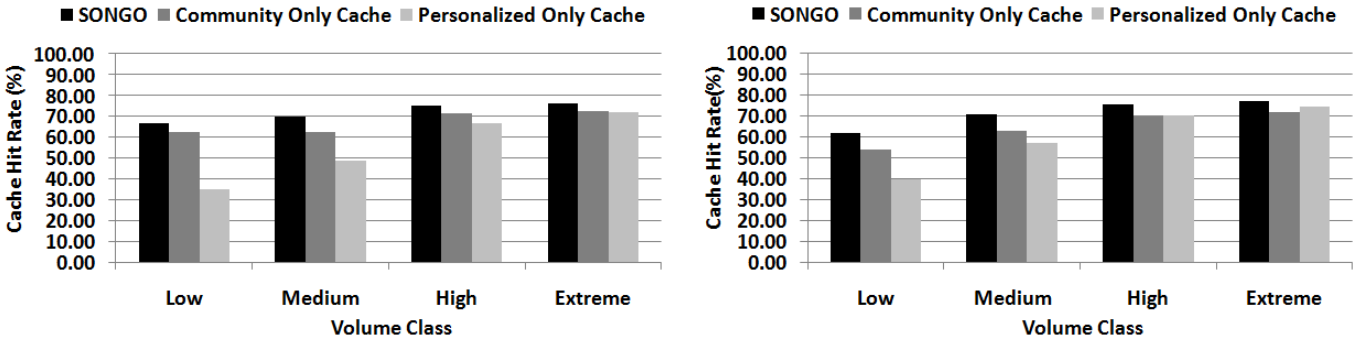


Figure 16: SONGO's average cache hit rate across the 4 different classes of users for (a) the first week and (b) the first two weeks of the month.

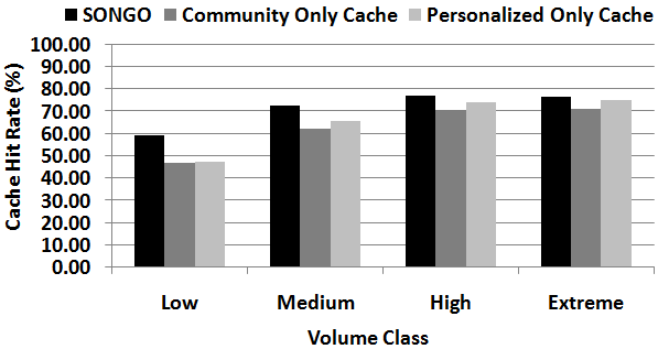


Figure 15: SONGO's average cache hit rate across the 4 different classes of users.

cache is initially empty and therefore cache hits are achieved only from repeated queries.

As Figure 15 illustrates, when only the community part of the cache is used, the average hit rate across all user classes is reduced from 65% to 55%. What is even more interesting is the fact that the hit rate seems to increase monotonically with the monthly query volume. Even though the exact same cache is used across all classes (since personalization is not used), the users that submit more queries seem to also experience higher hit rates.

When only the personalization part of the cache is used, the average hit rate across all user classes is reduced from 65% to 56.5%. Note, that for every user class, the personalization part of the cache achieves the same or higher hit rate compared to the case where only the community part of the cache is used. This is another indication of the high repeatability of mobile queries (demonstrated by our search log analysis in Section 2) that the personalization part of our cache is able to capture. In addition, the fact that the cache hit rates due to the personalized part of the cache increase for users with higher query volumes, indicates that users with higher query volumes tend to repeat the same queries more often.

Even though users repeat mobile queries frequently, the community part of the cache is still very important for the overall user experience. Figure 16 shows the average hit rate for the different user classes during the first week (Figure 16(a)) and first two weeks (Figure 16(b)) of the one-month long query streams. Note, that after the first week, the hit rate of the personalization part of the cache remains lower than that of the community part of the cache. In particular, in the case of the low and medium volume classes, the hit rate for the personalization part of the cache is significantly low. The less queries a user submits, the more time it takes the personalization part to warm up and be able to take advantage of the repeated queries. However, even during the first week, SONGO cache is able to provide the same hit rate with the one achieved in Figure 16 after a month. This is due to the community part of the cache that provides a warm start for SONGO and the best possible out of the box search user experience.

The breakdown of the queries that result into a cache hit can be seen in Figure 17. On average and across all user classes, 70% of the cache hits are navigational queries (i.e. facebook, youtube, hi5, etc.) and the rest 30% are no-navigational queries (i.e. michael jackson etc.). Even though navigational queries are dominant, note that for both the high and extreme volume classes the no-navigational hit rates are significantly increased or even doubled when compared to the medium volume class. This trend indicates that higher volume users tend to submit more diversified queries. However, even for this type of users SONGO is able to achieve high hit rates by taking advantage of the repeatability of mobile queries with its personalization-based part.

4.2.2 Daily Cache Updates

To understand how much the set of popular queries and links changes over time and to study how these changes affect SONGO's cache hit rate, we repeated the same experiments while updating the SONGO cache on a daily basis. In particular, we replayed the 400 monthly query streams extracted from the search logs, but at the end of every day, we updated the cache using the mechanism described in Section 3.3. Note that during the update process, the queries and links in the cache that have been accessed by the user at least

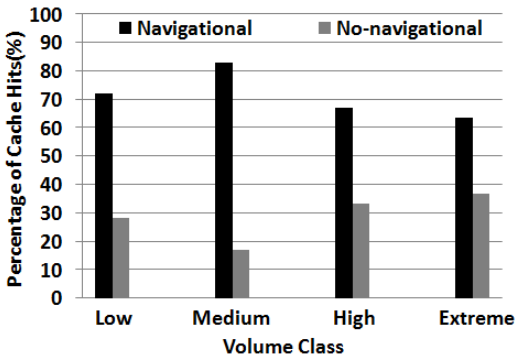


Figure 17: Breakdown of SONGO's cache hits into navigational and no-navigational across the 4 different classes of users.

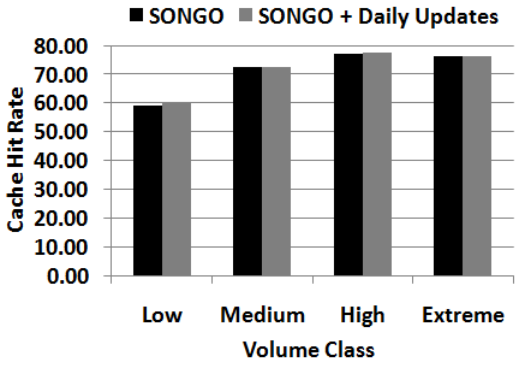


Figure 18: SONGO's average cache hit rate across the 4 different classes of users when daily cache updates are used.

once are not removed. As a result, the daily cache updates can only impact the hit rate that comes from the community part the cache.

Figure 18 shows the average cache hit rate across the different user classes when daily cache updates are used. On average, across all user classes SONGO achieves a cache hit rate of 66% when daily updates are used, an incremental increase of only 1.5% (66% vs 65% hit rate). The small increase could be due to the fact that the popular set of queries and links did not change significantly over the one month period we used in our evaluation. However, as Figure 19 shows, on average 40% to 50% of the approximately 2500 search results in the cache changed across days. This indicates that the search results that are responsible for the majority of hits in the community part of the cache are among the top 1000 most popular search results that always remain popular across different days.

5. RELATED WORK

There have been several research efforts on understanding mobile search behavior through detailed search log analysis [6],[7],[8],[9],[10],[20],[3],[4]. These efforts have analyzed search query volumes that vary from hundreds of thousands to several tenths of millions of queries. Their main focus has been on understanding mobile query characteristics (e.g. average query length etc.) and comparing them to typical desktop query characteristics, on demonstrating the locality of mobile queries across the community of mobile users and on providing a detailed breakdown of the different types of queries that mobile users submit (e.g. entertainment, shopping, etc.).

The work presented in this paper differs in three fundamental ways. First, we analyze 200 million mobile search queries, a query volume that is at least one order of magnitude larger than the query volume used in any other mobile search log study. Second, besides reporting similar observations on the locality of queries across the community of mobile users, we also study in detail the repeatability of mobile queries for individual users. Third, previous work has

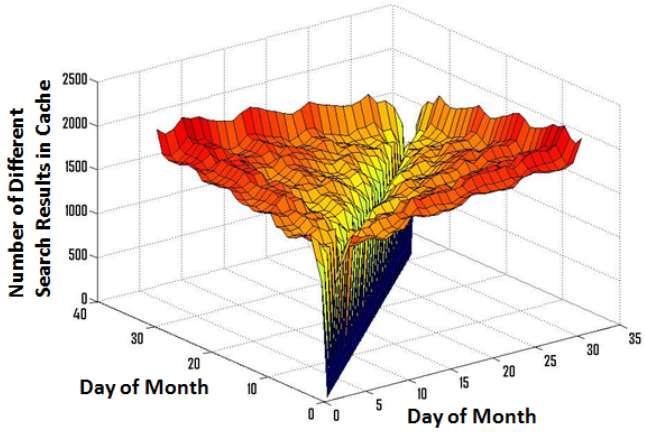


Figure 19: Number of different search results in the community part of the cache for every pair of days over the one month period used for evaluation.

only used the search log analysis findings to provide keyword auto-completion services on mobile devices [9]. In contrast, we leverage our mobile search behavior observations to design and implement a mobile search cache capable of instantly providing *search results*.

The concept of caching information on mobile devices has also been applied in the past in the context of web caching and prefetching [15],[2],[11],[14],[16]. Our work is complimentary to these efforts, in the sense that SONGO focuses on caching search results and not web pages. However, SONGO could be used along with web caching techniques to provide an instant browsing experience on the phone.

Advertisement caching was also recently proposed. In [5], a scheme for storing ads on the PC is proposed for performance, privacy and profit. However, the work in [5] focuses on desktop machines and thus, does not address any of the design and implementation details of building such a cache system on a mobile device.

6. DISCUSSIONS AND CONCLUSION

We have presented SONGO, a mobile search cache for mobile devices. Using our prototype implementation and

query streams of real users extracted from the mobile search logs, we demonstrated that SONGO can serve $2/3$ of the queries that an individual user submits on his phone, 16 times faster and 23 times more energy efficiently compared to when using the 3G radio. We believe that the combination of the SONGO architecture and the continuously increasing memory resources on mobile devices can transform the mobile user experience in the context of mobile web search and far beyond.

Search Interface: Since retrieving search results from the local cache in SONGO only takes tens of milliseconds, one can combine SONGO with query auto-complete to improve mobile search user interface by reducing the number of query key strokes. A mobile phone can have a dictionary of popular search queries. While a user types search query characters, the system looks in the dictionary to find the most popular queries that start with the contents in the search textbox. The corresponding search results can be immediately displayed in the main window, without user pressing the search button. For example, while typing "f" in the search box, the search box autocompletes it as "facebook" *and the search result (i.e. http://mobile.facebook.com) is already shown in the result panel of the page*. If the user does not find the search results after finish typing a query term, she can then click the search button and use the online service. So SONGO is completely transparent to the user.

Content Caching: A caching architecture like SONGO is not limited to improving mobile search experience. As the storage and processing capabilities of mobile devices continue to improve, other frequently accessed contents such as local business listing, coupons and promotions, blogs and reviews, popular web sites, as well as certain contents on personal computers can be cached on the devices for fast user access on the go. While storing gigabytes of contents on a phone is not expensive even with current flash storage capacity, how to enable users to quickly find the right information is nontrivial. Search is natural user interface for such interaction. But one must solve the challenges of building indexes and ranking the contents on low power devices with limited user interface.

Personalization and Privacy: A key advantage of performing content search and ranking locally on a mobile phone is personalization. The personalized hash table in SONGO is an example of it. In general, the phone can learn from past user activities and contextual information to better rank and present contents that fits the preference of the particular user. Performing personalization on the mobile phone rather than in the cloud also preserves user privacy better. For example, the phone can filter and aggregate user access logs and activities before sharing it with the community data mining.

Accounting: A practical concern of deploying SONGO is how to enable third parties, such as comScore, to correctly account for mobile search traffic, which is a critical market research number and directly links to advertisement revenue. If on average 66% of search traffic are not visible on the Internet, then the accounting cannot be accurate. One solution is to send a "one-way" search query in the background. The query asks the search engine not to return any results, but the traffic can be tracked by third party.

7. REFERENCES

[1] 2TB memory cards coming soon, `http://www.wired.com/gadgetlab/2009/01/two-terabyte-sd/`.

[2] A. Balasubramanian, B. N. Levine, and A. Venkataramani. Enhancing interactive web applications in hybrid networks. In *Proceedings of MobiCom*, New York, NY, USA, 2008.

[3] K. Church, B. Smyth, K. Bradley, and P. Cotter. A large scale study of european mobile search behaviour. In *Proceedings of MobileHCI*, New York, NY, USA, 2008.

[4] K. Church, B. Smyth, P. Cotter, and K. Bradley. Mobile information access: A study of emerging search behavior on the mobile internet. *ACM Trans. Web*, 1(1):4, 2007.

[5] S. Guha, A. Reznichenko, K. Tang, H. Haddadi, and P. Francis. Serving ads from localhost for performance, privacy, and profit. In *Proceedings of Hotnets*, 2009.

[6] M. Kamvar and S. Baluja. A large scale study of wireless search behavior: Google mobile search. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, New York, NY, USA, 2006.

[7] M. Kamvar and S. Baluja. Deciphering trends in mobile search. *Computer*, 40(8):58–62, 2007.

[8] M. Kamvar and S. Baluja. The role of context in query input: using contextual signals to complete queries on mobile devices. In *Proceedings of MobileHCI*, 2007.

[9] M. Kamvar and S. Baluja. Query suggestions for mobile search: understanding usage patterns. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, New York, NY, USA, 2008.

[10] M. Kamvar, M. Kellar, R. Patel, and Y. Xu. Computers and iphones and mobile phones, oh my! In *18th International World Wide Web Conference*, pages 801–801, April 2009.

[11] E. P. Markatos and C. E. Chronaki. A top-10 approach to prefetching on the web. In *Proceedings of INET*, 1998.

[12] Mobile Search Trends Report, `http://www.marketingcharts.com/interactive/mobile-local-search-ad-revenues-to-reach-13b-by-201`

[13] Monsoon Solutions, Power Monitor, `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[14] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *IEEE Trans. on Knoweledge and Data Engineering*, 15(5):1155–1169, 2003.

[15] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *SIGCOMM Comput. Commun. Rev.*, 26(3):22–36, 1996.

[16] J. Pitkow and P. Pirolli. Mining longest repeating subsequences to predict world wide web surfing. In *Proceedings of USENIX* , 1999.

[17] Sony Ericsson Xperia X1a Mobile Phone, `http://www.sonyericsson.com/x1/`.

[18] J. Teevan, E. Adar, R. Jones, and M. A. S. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *Proceedings of SIGIR*, New York, NY, USA, 2007.

[19] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of SIGIR*, New York, NY, USA, 2005. ACM.

[20] J. Yi, F. Maghoul, and J. Pedersen. Deciphering mobile search patterns: a study of yahoo! mobile search queries. In *Proceedings of WWW*, New York, NY, USA, 2008.