FindAll: A Local Search Engine for Mobile Phones

Aruna Balasubramanian
University of Washington
arunab@cs.washington.edu

Niranjan Balasubramanian
University of Washington
niranjan@cs.washington.edu

Samuel J. Huston
UMass Amherst
sjh@cs.umass.edu

Donald Metzler
USC
metzler@isi.edu

David J. Wetherall
University of Washington
djw@cs.washington.edu

ABSTRACT

We present the design and evaluation of *FindAll*, a local search engine that lets users search and retrieve web pages, even in the absence of connectivity. Our user study with 23 users show that mobile users often search for web pages that they have previously visited, known as re-finding. This re-finding behavior makes the case for a local solution. *FindAll* goes beyond caching and using keyword search, and instead, implements a full blown search engine. The key challenge in *FindAll* is in designing a search engine, which is both memory- and energy-intensive, on the constrained phone environment. To this end, *FindAll* balances the cost of running the search engine with the expected benefits of serving a web page locally. *FindAll* estimates the benefits of local search, by learning the re-finding behavior of users. We implement *FindAll* on Android by adapting a publicly available search engine. Our evaluations, based on the traces collected from our user study, shows that *FindAll* reduces search latency by two-folds for users who re-find often, and reduces 3G data usage by up to 100 MB a month.

Categories and Subject Descriptors

C.2 [**Computer Communication Network**]: Network Architecture and Design— *Wireless Communication*

General Terms

Algorithms, Design, Performance

Keywords

Web search, Latency, Energy savings, Local search, Refinding, Measurement, User study

1. INTRODUCTION

Many popular smartphone apps, including web search, maps, yelp, and others, solely rely on the cloud for its operations. The apps fail completely during periods of no connectivity and perform poorly during periods of bad connectivity. This reliance on the cloud can severely degrade user experience, especially on cellular networks—cellular networks have high round trip delays [14], offer poor connectivity [27], and are often unavailable [7]. Even with good cellular connectivity, a web search session is an order of magnitude slower on mobile phones than on desktops [15, 25]. 3G data is also becoming increasingly expensive, and as cellular providers introduce tiered data plans [21], users are becoming more aware of their 3G data usage.

Improving the local availability of content can help mitigate these issues. Mobile storage is advancing at a rapid pace, and technology trends suggest that storage capacity will continue to rise and will remain cheap [15]. Thus, trading local storage for connectivity is an attractive alternative to *always* relying on the cloud.

In this work, we focus on using local storage to improve the performance of *web search*. To this end, we leverage the *re-finding* behavior of web search users; users often search for web pages that they have previously visited [23]. Re-finding queries constitute nearly 80% of the total queries issued by more than half the population of mobile users [18] . If users can effectively search through their local caches, a significantly large fraction of searches queries can be answered locally.

Imagine that a mobile user Alice is browsing some product web pages on her laptop or her phone. She wants to re-find a specific product page on her phone when she goes to a store, but the connectivity to 3G is poor. How should her smartphone support local search? Manually browsing through the browser history is cumbersome [4]. Web caches require that Alice has access to the URL link, which is usually long, not easy to remember, and is rarely used for re-finding [10]. Search engines such as Google [20] and Bing [19] store user history in the cloud. However, using the search engine's cloud-history feature requires network connectivity, and further, storing the user history in the cloud raises privacy concerns.

We present *FindAll*, a system that collects all web pages that the user visits on all of her devices, caches them locally, and indexes [1] them. *FindAll* then implements a full scale search engine over this local index to allow users to effectively search over their local history.

A local search engine will allow Alice to re-find desired web pages quickly, privately, and without depending on network connectivity. The key challenge in *FindAll* is the high resource cost of running a conventional search engine on the phone. Building search engine indexes is both a memory- and an energy-intensive operation.

The *FindAll* design is motivated by a user study in which we logged the mobile search and browsing patterns of 23 participants

[1]An index is a data structure used by search engines for fast and efficient retrieval.

CoNEXT'12, December 10–13, 2012, Nice, France.

for 30 days. We also logged the corresponding desktop search patterns for a subset of the users. We find that, similar to other mobile studies, re-finding was common in our dataset with 52% of the queries being re-finding queries. We leverage two key findings from our study to design *FindAll*. First, almost 45% of the re-finding queries were submitted within 50 minutes of the original request. This suggests that we need to locally index web pages near when they are browsed, rather than indexing pages in the cloud and periodically downloading them. Second, there is a large diversity in re-finding behavior across users, but each user tends to be relatively consistent in their level of re-finding. This suggests that *FindAll* should adapt to the user's re-finding patterns, so that the index computation, an energy intensive operation, is performed only when it is beneficial to the user.

FindAll seeks to balance two goals: maximizing the local availability of web pages, while ensuring that the energy consumption for *FindAll* operations is no more than that of default search. The central question in the *FindAll* design is: When should *FindAll* index web pages? If web pages are indexed frequently they are more available in the local cache, but the energy to index the web pages increases. If the web pages are indexed less frequently, they will not be available locally when the user tries to re-find the web page.

The *FindAll* algorithm indexes web pages when the energy to index is lower than the expected cost of not indexing. By not indexing a web page locally, *FindAll* incurs a cost, because the web page will have to be downloaded from the Internet. However, *FindAll* incurs this cost *only if* the user searches for the web page. *FindAll* estimates the probability that a user will search for a web page by learning user behavior; it uses this probability estimate to compute the benefits of indexing. As a result, *FindAll* aggressively indexes when users are expected to re-find often, and indexes more conservatively when users are not expected to re-find.

We implement *FindAll* in Android by adapting Galago [2], a publicly available search engine, to mobile phones. We evaluate our implementation using traces from real mobile users. We find that *FindAll* significantly improves web search performance by increasing local availability of web pages by 40%. By improving availability, *FindAll* reduces search latency by a factor of two for the users who re-find often. By reducing Internet downloads, *FindAll* reduces 3G data usage by up to 100 MB a month for users who re-find often.

Importantly, these improvements do not come at an energy cost: *FindAll* reduces the overall energy consumption by 30% for users who re-find often, and does not increase the overall energy consumption for users who do not re-find often. Adapting to the re-finding behavior of users provides significant benefits. Non-adaptive strategies do not change their indexing policies according to users re-finding patterns; as a result, they index more often than needed for users who do not re-find, resulting in up to 50% increase in energy consumption compared to default web search. On the other hand, they index less often than needed for users who do re-find often, resulting in 39% lower availability compared to *FindAll*.

We make three key contributions in this work. First, our measurement study presents the day-to-day browsing and re-finding behavior of real mobile users and highlights implications for mobile search. Second, we develop a local search engine that is suitable for constrained phone environments: *FindAll* indexes web pages within the memory constraints of phones and according to the energy implications of indexing. Third, we show using an extensive evaluation, that *FindAll* improves both latency and availability of mobile search.

2. BACKGROUND

2.1 The Re-finding problem

Re-finding is a well-studied topic in web search, where users issue search queries for web pages that they have previously seen. Studies have shown that 40% to 60% of all search queries from desktops are re-finding queries [22, 10]. Even on mobile phones, 50% of mobile users issue more than 80% re-finding queries [15]. In our own user study, 52% of search queries were re-finding queries (§3).

Because of the dynamic nature of Internet content, re-finding is not always easy– search results change constantly [5, 22], users often do not remember the exact query they typed originally [6], and the target web page may not be among the top web pages returned by the search engine even when the same query is issued repeatedly [24].

Poor cellular connectivity further exacerbates the re-finding problem. Users need to first download the search results page, search for the link to the specific web page, and then re-download the web page. High latency in cellular networks can make this experience tedious [15]. Even if web pages are cached locally, searching through the history is cumbersome [4], and users seldom look through their search history for re-finding [10].

A database lookup, consisting of a <query,webpage> map, could make searching through the local history easier [15]. A database requires that the original search query and the re-finding query be the same, since the query is used as the lookup key. However, the re-finding query is often different from the original query [23]. In our study, the query used to re-find a web page was different from the original query for 24% of the cases (§3). As a result, *FindAll* was 27% more effective in searching web pages, compared to a database lookup (§7).

Further, a database solution can only help retrieve web pages that are found through search; if a user visits a web pages through other sources such as social media and email, these web pages will not be stored in the database.

2.2 Search engines

FindAll uses a state-of-the-art information retrieval system to efficiently and effectively search through locally stored content.

Given locally stored content, a simple alternative to using a search engine is to use keyword matching; i.e., every web page that contains the keyword is retrieved, but results are not ordered in any manner. Keyword matching is not effective for search. Imagine that the user has tens of web pages about hotels in Anchorage, but the user is looking for a particular hotel called *Anchorage Hotel*. A keyword search on *Anchorage Hotel* will return all the web pages, in no particular order. Finding the relevant web page from this unordered list is cumbersome, especially in small form-factor phones that display 3-4 search results per page.

Instead, search engines and most modern information retrieval systems go beyond simple boolean search and *rank* web pages in response to a user query. Search engines leverage several types of features such as the frequency of the query words within the document, the proximity of the query words, and their distribution in various sections of the document (URL, title, body etc). Our evaluations (§7) show that the *FindAll* is 33% more effective in returning the relevant web page as one of the top results, compared to keyword matching.

To retrieve and rank web pages efficiently, search engines first index all of the web pages. The index comprises of a mapping between words and the list of web pages in which they occur, and allows fast retrieval. The core of any indexing algorithm is to sort the <word, web page> tuple. Sorting is usually implemented as an

external-merge sort because the entire list of tuples is too large be held in main memory [26]. However, this task is both memory- and energy-intensive.

Finally, with the growing variety of mobile apps, re-finding is likely to become a common phenomenon across other apps, such as news readers, social media, maps, etc. As the number of re-found content scales, a search engine solution is likely to become essential. As a first step, we focus on re-finding *web pages*. This is because the need for re-finding is well-established for web search, and browsing remains a frequent activity for many mobile users; traces of mobile browsing activity give us a basis for evaluating our system.

3. USER STUDY

To measure day to day browsing and re-finding patterns of real mobile users, we conduct a user study. We built a logging application that records anonymized user browsing history. We use the data that we record during the study to drive the evaluation of *FindAll*.

3.1 Re-finding Defined

A re-finding query is the set of keywords entered by a user, used to retrieve a previously viewed web page. We call the corresponding web page that is visited through the query as the re-finding or the re-found web page.

Knowing if a query is a re-finding query is a hard problem since it depends on user intent. In the past, researchers have focused on *re-finding URLs*. In contrast, in *FindAll*, we are interested in knowing if the web page itself (i.e., the *content* of the URL) is re-found. Given this goal, we define *re-finding* to be revisiting a previously visited web page when:

- The web page is revisited via a search query.
- The content of the web page remains unchanged from the previous visit.

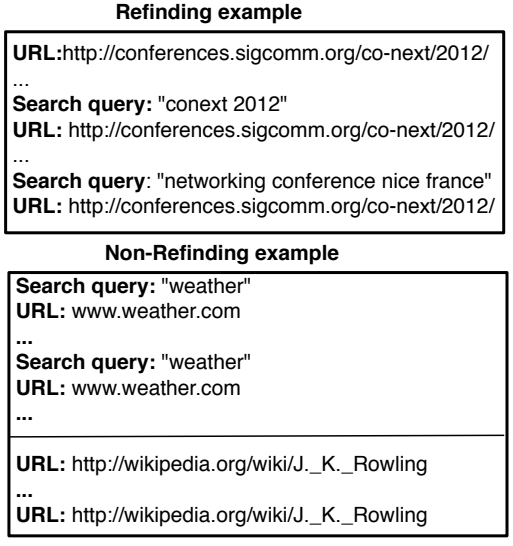

Figure 7: Refinding and Non-refinding examples

Figure 7 shows examples of re-finding and non re-finding behavior. For example, the CoNext 2012 web page is re-found twice. Note that the search query is different for each re-find request. On the other hand, the web-page for the URL "www.weather.com" is visited twice but is not marked as re-finding even though it was found via a search query. This is because the webpage is likely to have changed since the previous visit. Finally, the wikipedia entry for J. K. Rowling is not marked re-find because the web page is not visited via a search query.

3.2 Measurement methodology

3.2.1 Logging software

The *FindAll* logger application records the user's browsing activity on their phones and personal computers. To encourage user participation, the logger anonymizes the browsing logs of the user. URLs of web pages are hashed using a user-specific key[2], such that two identical URLs are hashed to the same value. Similarly, search queries are also hashed using a user-specific key.

On the phone, the application logs the following items: (i) Time (ii) URL hash (iii) Hash of the search queries (iv) Size of the web page (iv) Connection status (3G/WiFi). The logger uses the Android bookmark database to log browser activity. We ensure that the logs do not contain spurious URLs due to client-side redirections or due to the user pressing the "back" button.

On the computer, the application logs the following items: (i) Time (ii) URL hash (iii) Hash of the search queries. The logger periodically polls the browser's sqlite database to obtain browser information. The same hash key is used to hash the content of the user's phone and user's computer.

3.2.2 Data collection

We logged mobile browsing history from 23 users for 1 month. In addition, we collected the computer browsing history from 7 of the 23 users for a month to give us cross-device browsing history. Table 1 shows the user characteristics. All except two of the participants were undergraduate or graduate students from two different universities.

We note that 9 of the 23 users either had limited 3G plan or did not have 3G plan at all on their phones. A system such as *FindAll* is especially useful for such users.

	Users		Users
Undergraduate	13	Graduate students	8
Computer science majors	12	Other majors	9
No 3G plan	4	Limited 3G plan	5

Table 1: Characteristics of user study participants.

During post-processing of the collected logs, we mark a query and the corresponding web page visited through the query as re-finding, if it satisfies the definition (§3.1). Note that we can identify repeated requests to the same web page since the URLs are hashed to the same value. However, we cannot access the content of the web page because of anonymization; therefore, we cannot identify web pages whose content has changed since the last visit. This is a difficult problem.

However, to alleviate this problem, we created a list of URLs whose content change frequently. We obtained the top 500 most popular web pages from Alexa [1]. We then identify the web pages that are "not cacheable", as marked in their HTTP headers. The *FindAll* logger software marks all web pages in the top 500 popular pages that are "not cacheable", and these web pages are automatically marked original (or non re-find).

[2]We employ one way hashing which does not allow original data to be recovered even with the user-specific key.

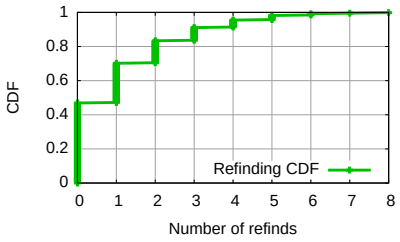

Figure 1: CDF of re-finding pages for all 23 users. Over 52% of the web pages are re-found.

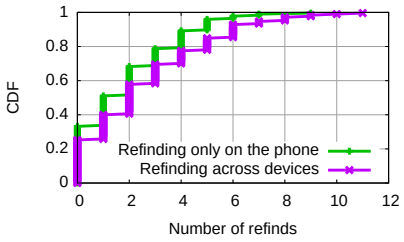

Figure 2: Re-finding when cross-device indexes are used, for the 7 users from whom we collect cross-device indexes.

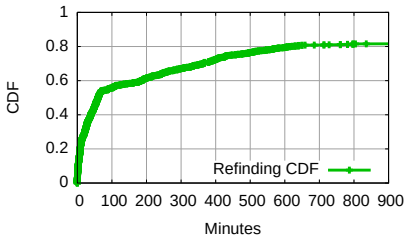

Figure 3: CDF of the time between when a web page is re-found and the first time the page is visited.

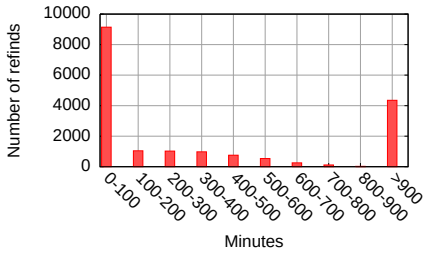

Figure 4: PDF of the time between when a web page is re-found and the first time the page is visited.

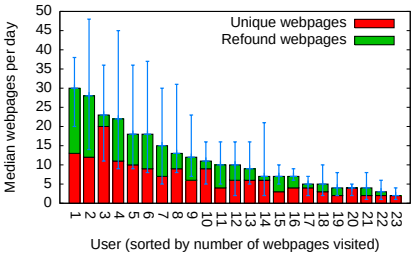

Figure 5: Browsing and re-finding statistics per user. The percentile bar shows the 25th and 75th percentile.

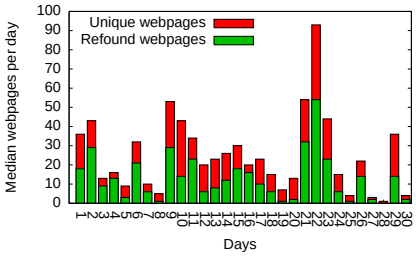

Figure 6: Browsing and re-finding statistics of one user across all the days of the experiment.

3.3 User Study Results

3.3.1 *Re-finding across users*

More than half of all visited web pages are re-found at least once. Figure 1 shows the CDF of the number of times web pages are re-found by the user; 0 indicates that the page is not re-found, 1 indicates that the page is re-found once, and so on. The figure shows that 52% of the web pages are re-found at least once, and 25% of the web pages are re-found two or more times. This finding is consistent with previous studies that showed re-finding is wide-spread among web search users [23, 15].

Re-finding increases when considering cross-device browsing history. In Figure 2, we compare the cross-device re-finding characteristics. For measuring cross-device re-finding, a web page visited on the phone is marked as re-found if it satisfies the re-finding definition, and is previously viewed either on the user's computer or the user's phone. For the subset of 7 users with cross-device history, the percentage of re-found web pages on the phone increases from 58% to 72%, when the web pages visited on the computer are also included in estimating re-finding. This suggests that leveraging cross-device history can provide substantial benefits for *FindAll*.

Most re-find occur either within the first 50 minutes or after 10 hours. Figure 3, shows the time difference between the first visit to a web page and subsequent visits to the same web page. About 45% of the time, re-finding occurs within the next 50 minutes; a further 20% of the time, web pages are accessed no sooner than 900 minutes after the web page was first visited. (Note that the figure is cropped at 900 minutes.) Figure 4 shows the probability density function of the same data. Taken together, the figures show that web pages need to be indexed relatively soon after a user visits them, to improve availability.

3.3.2 *Diversity in browsing and re-finding*

Next, we focus on the re-finding behavior of individual users. Figure 5 shows the browsing and re-finding characteristics for each of the 23 users. The figure shows the median number of web pages browsed by the user.

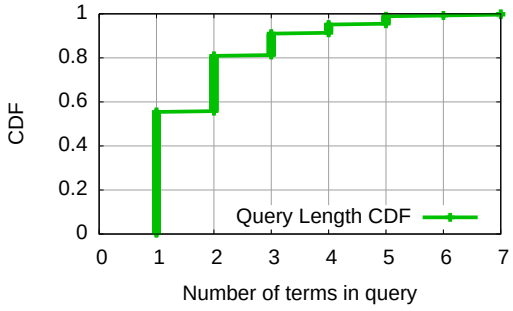

Figure 8: The CDF of the number of terms in a user query.

Three important characteristics emerge from this graph: 1) Users have varied browsing characteristics. While 33% of the users browse more than 15 web pages a day, 30% of the users browse less than 3 web pages per day. 2) The re-finding characteristics vary for different users. For 8 of the 23 users, nearly half of the web pages are re-found web pages, but for 6 of the 23 users, less than 10% of the web page visits are re-finding visits. 3) Even for a given user, browsing behavior varies across different days. The percentile bars shown in Figure 5 are the 25th and the 75th percentile of browsing behavior for a user, and their difference shows that there is significant variability.

3.3.3 *Per user browsing and re-finding patterns*

Re-finding percentage is consistent for a given user, despite daily variability in browsing patterns. Figure 6 shows the browsing and re-finding behavior across the 30 days for a single, example user. While there is a large variability in browsing patterns for a single

user, the re-finding percentage does not vary greatly across days. For the user in Figure 6, the average re-finding over the 30 days is 47% with a standard deviation of only 9%. Analyzing our data, we find that the re-finding percentage is fairly consistent across days for most users.

3.3.4 Query characteristics

Figure 8 shows a CDF of the number of terms in each user query. 40% of the queries contain more than 2 terms, with a maximum of 7 terms in a query.

More interestingly, we find that for 24% of the re-find queries, the original query and the re-find query were different (not shown in figure); two queries are different if at least one term in the query do not match. Two queries with the same terms but in different order are considered to be the same.

This finding is consistent with previous studies on the search characteristics of desktop users, which also showed that the original query and re-find query are often different [23].

3.4 Summary

We find two key characteristics of mobile re-finding behavior that have implications for the design of *FindAll*:

1. *Nearly 45% of web pages are re-found in the first 50 minutes.* This motivates the need for a local search engine solution that indexes web pages as soon as they arrive, to improve availability.

2. *Users have diverse but consistent re-finding patterns*. Since some users may seldom re-find web pages, *FindAll* learns the re-finding pattern of users, so that the energy-intensive indexing operation is performed only if there is associated benefit.

We note that the aggregate findings from our user study are consistent with findings of a much larger re-finding study using 8000 Bing mobile users [18]. The Bing study also shows that: (1) mobile users exhibit high re-finding behavior, and (2) users have diverse re-finding characteristics. In addition, our study characterizes non-aggregate day-to-day behavior of mobile search users. For example, we learn that user's show consistent re-finding behavior across days, that a large fraction of re-finding occurs soon after the original search, and that user's often re-find across devices. We also characterize the query term distribution of mobile search queries.

4. FINDALL

FindAll is a local search engine designed specifically for mobile phones.

4.1 Goals

The two main goals of *FindAll* are: (i) to increase local availability of web pages, and (ii) to operate within the memory and battery constraints of phones.

We say that a web page is available locally, if the web page has been indexed locally and can be retrieved from the local cache when the user issues a re-find request. Increasing local availability provides multiple benefits: (1) It enables users to re-find more web pages easily by searching through a smaller collection of locally available web pages. (2) It reduces search latency and reduces the use of expensive 3G data, by avoiding repeated downloads of web pages from the Internet.

In terms of resource constraints, memory and energy are the two most limited resources on mobile phones. We design *FindAll* such that it consumes no more energy than default; the default in our case is Internet-based web search. With respect to memory, we design the *FindAll* indexing algorithm to write to disk frequently, to avoid overflowing available memory.

Next, we discuss the *FindAll* architecture and challenges.

4.2 Architecture

Figure 9 shows the FindAll architecture. *FindAll* stores web pages as a user downloads them, and then indexes them in blocks. *FindAll* stores both the index and the cached web pages on its external memory card.

When a query is submitted to *FindAll*, it runs the retrieval algorithm on the index. The algorithm returns a results page with a set of links, similar to search engines such as Google or Bing. When the user clicks on one of the links, the corresponding web page is fetched from the local cache. If the results page does not contain any link or if none of the links are relevant to the user, the user switches to Internet-based search.

FindAll also indexes web pages on all other devices that the user browses, such as their laptop and desktop computers. When the phone is being charged, *FindAll* downloads the cross-device indexes on to the phone and merges them with the existing index. *FindAll* also downloads the corresponding web page cache to the phone.

4.2.1 Partial indexes

Traditionally, search engines use static indexing, which are designed to index large collections of documents in one-go. However, static indexing approaches cannot effectively handle the frequent updates necessary for maintaining high-availability in *FindAll*'s local search engine.

Therefore, we implement a dynamic indexer that is designed for handling frequent updates [17]. Our implementation builds partial indexes of smaller blocks of web pages in main memory. Partial indexes allow for web pages to be indexed as they arrive, increasing the local availability of a web page. The partial indexes are then merged periodically.

Maintaining the partial index in memory is more energy efficient, as merging partial indexes that are in-memory requires less energy and compute resources. However, because of the severe memory constraints in phones, *FindAll* writes the partial indexes to disk as soon as they are created. To illustrate the memory constraint in phones, consider the following example. The HTC thunderbolt phone, released in 2011, has around 800 MB RAM. However, only 240 MB is available for applications. In our experiments, indexing 50 web pages and maintaining the index in-memory resulted in the phone running out of memory. In this instance, no other application was using the memory. In the presence of multiple background applications, the number of web page indexes that can be maintained in memory becomes even less.

As a consequence of not maintaing in-memory partial indexes, *FindAll* incurs higher energy costs due to merging. We note that it is essential that the partial indexes be merged; as the number of partial indexes grow, the retrieval latency increases, because the retrieval algorithm needs to search through each partial index. Our experiments show that, as the number of partial indexes grow, the retrieval latency becomes higher than the latency of downloading from the Internet, thus reducing the benefits of a local search engine.

4.2.2 When to index?

FindAll buffers downloaded web pages and indexes them as a block. The central question in *FindAll*'s architecture is in deciding when to index the buffered web pages.

On the one hand, indexing in smaller block sizes improves local availability and consumes less energy by reducing number of Internet downloads. On the other hand, indexing in smaller block sizes consumes more energy during indexing due to the need for frequent merging of the partial indexes.

To understand the trade-off between indexing small block sizes vs indexing large block sizes, we conduct a simple experiment.

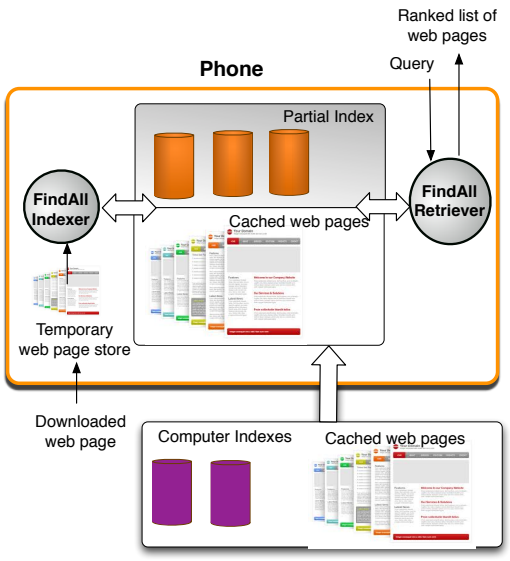

Figure 9: *FindAll* local search engine

We index 90 web pages obtained from the Microsoft Search logs, with a total size of 12.3MB. We adapt a publicly available search engine called Galago [2] to index the web pages. We conduct all our energy measurement using the Monsoon Power Monitor [3] using a Motorola Droid phone. All network activities are conducted on the 3G interface.

Figure 10 shows the trade-off between the index energy and availability. The x axis shows the block size; a block size of 15 implies that indexes are created when 15 web pages are downloaded. The left y axis shows the energy required to index the web pages; indexing all the 90 web pages together consumes only 52 Joules, while indexing in block sizes of 15 web pages require 140 Joules, a nearly 3-fold increase. This is because of the additional energy cost incurred due to merging.

We then assume that 20% of the search queries are re-finding queries, uniformly; note that this is a conservative estimate. The right y axis shows the energy cost of not indexing (i.e., the cost to download the search result from the Internet) for different block sizes. For example, if we wait for all 90 web pages to be downloaded before indexing, 18 re-found web pages will need to be downloaded from the Internet because they are not available locally. Clearly, there is an optimal block size that balances the two opposing goals.

In the next section, we describe how *FindAll* estimates when to index a block of web pages, to balance its twin goals of availability and energy.

5. FINDALL INDEXING ALGORITHM

The *FindAll* indexing algorithm decides when to index a set of buffered web pages. The goal of the indexing decision is to maximize local availability, while ensuring that *FindAll* consumes no more energy than default search.

Figure 11 shows the *FindAll* indexing algorithm. *FindAll* makes indexing decisions by comparing the expected energy cost of indexing web pages ($E[I]$) and the expected cost of not indexing web pages ($E[\neg I]$).

- The cost of indexing web pages ($E[I]$) includes the cost of indexing the current block of web pages, as well as a penalty based on expected cost of indexing future web pages. The penalty is designed to penalize indexing of small block sizes.
- The cost of not indexing web pages ($E[\neg I]$) is simply the energy required to search and download each web page from the Internet; in other words it is the energy required for default web search.

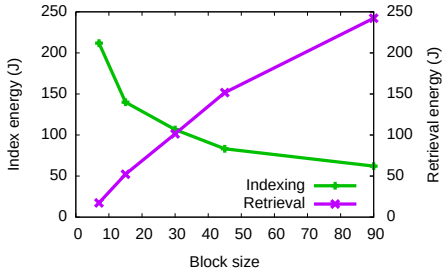

Figure 10: Experiment that shows the trade-off between availability and indexing energy.

FindAll will decide to index a block of web pages, if the cost of indexing ($E[I]$) is lower than the cost of using default web search ($E[\neg I]$). Next, we describe the estimation of $E[I]$ and $E[\neg I]$.

5.1 Expected cost of indexing ($E[I]$)

We estimate indexing cost as the sum of the current indexing cost and a penalty. The penalty is the estimated impact of the current indexing decision on future indexing operations.

The penalty is designed to discourage indexing small blocks, which can lead to more merge operations in the future. Specifically, if *FindAll* decides to index a block of B web pages, it estimates the cost of indexing future web pages, as if all future pages are also indexed in blocks of size B. The penalty is then computed as the energy to index future web pages normalized over the total number of blocks. Our evaluations (§7.4) show that this simple heuristic for penalty estimation provides a good balance between availability and energy, compared to alternate indexing strategies that do not penalize indexing in small blocks.

5.1.1 *Energy for indexing and merging*

The energy spent in indexing a block of B web pages is the sum of the energy to index the block and then merge the partial index with the existing index. Let the number of bytes in the block of B web pages be denoted by $|B|$ and number of bytes in the existing index be denoted by $|P|$. The value of $|P|$ is known. Then,

$$\begin{aligned}\text{Expected energy to index/merge } B \text{ web pages} = \\ \text{Energy to index } |B| \text{ bytes} \\ +\text{Energy to merge indexes of sum size } |B|+|P| \text{ bytes}\end{aligned} \quad (1)$$

We use a simple linear model to estimate the energy to index and merge web pages of a certain size. To this end, we index web pages of various sizes (without merging) from 0 to 5MB in 200K intervals, and fit a linear model to the data. This allows us to compute the energy to index web pages for a given number of bytes. Similarly, we build a linear model to estimate the energy to merge two indexes of a given sum size.

5.1.2 Penalty

To penalize energy decisions that index in small blocks, we assume that the remaining web pages for the day will also be indexed using a block size of B.

Suppose that there are W additional pages that the user will download during the day [3]. Then the penalty for indexing the remaining W web pages in blocks of B is given by:

$$\text{Penalty} = \frac{B}{W} \times \{\frac{W}{B} \times \text{Energy to index |B| bytes} + \text{Energy to merge } \frac{W}{B} \text{ partial indexes}\} \quad (2)$$

We note that $\frac{B}{W}$ is the normalization/amortization factor.

The expected cost of indexing a block of B web pages is then given by:

$$E[I] = \text{Expected energy to index/merge } B \text{ web pages + Penalty} \quad (3)$$

5.2 Expected cost of not indexing ($E[\neg I]$)

To estimate the cost of not indexing a block of web pages, *FindAll* first estimates the probability that a web page will be re-found. We first describe how *FindAll* estimates this probability, and then describe the estimation of $E[\neg I]$.

5.2.1 Predicting re-finding probability

FindAll uses the re-finding patterns of users to estimate the probability that a web page will be re-found. To this end, we build an *online* classifier, which when given a web page and the time of evaluation, estimates the probability that the web page will be re-found in the next T time units. The variable T is the same time interval used by the *FindAll* indexing algorithm (Figure 11).

Features:.

We use three features to train our classifier:,

- Base re-find probability – The fraction of user requests that are re-find requests. This feature encodes the intuition that if the user re-finds often in the past, then she is likely to re-find often in the future. Recall that the user's refinding patterns remain consistent across different days (§3.3.3), and is therefore a consistent signal for re-finding. This feature takes fraction values between 0 and 1.
- Session features – Is the user currently in a browsing session? The intuition for this feature is that, if the user is not currently in a browsing session, then it is unlikely that the user will be browsing, and in turn, it is unlikely that the user will re-find webpages. The session feature takes a binary value of 0 or 1, depending on whether the user is in a browsing session or not.
- Download history of the page – Has the web page been re-found recently? The intuition is that if the user recently re-found a web page, then it is likely that the user will re-find the web page again [23]. The download history feature takes a binary value of 0 or 1, depending on whether the web page has been re-found recently or not.

Classifier training:.

To train the classifier, we use half of the data we collect from each user in our user study. Starting from the beginning of the trace, we gather all web pages that were downloaded in each T time unit. We assign labels to each web page in this set: *re-find* if the web page was indeed re-found in the next T time units; *not-refind* otherwise.

[3] *FindAll* keeps track of the average web pages browsed per day to estimate W.

1. If new web page is downloaded by the user, store in temporary buffer B
   (a) If this is the first page in the buffer, set a timer T
2. When timer goes off
   (a) If $E[I] < E[\neg I]$, where $E[I]$ is the expected indexing energy, and $E[\neg I]$ is the expected cost of not indexing in the next T minutes, then
      - Index buffer B, empty the buffer, and cancel the timer.
3. If phone is being charged && B is not empty
   (a) Index buffer B, empty the buffer, and cancel the timer.

Figure 11: *FindAll* indexing.

The process is repeated for each T interval to obtain a set of labeled instances. Each of these labeled instances are represented using the the three features described above.

We learn a logistic regression classifier; the goal of the classifier is to minimize classification errors on this training data. On its completion, the learning algorithm outputs a set of weights corresponding to each feature. At each time instance, the re-finding probability of a web page is computed as the weighted sum of each feature value at that time instance. As previously noted, the weights for the weighted sum computation are obtained from the learning algorithm.

The classifier only needs to be trained once a day during periods of inactivity or when the phone is being charged. Since the prediction only involves computing a weighted sum of feature values, there is little energy consumption for the prediction.

Our prediction algorithm used only three simple sets of features; our evaluations show that *FindAll* is able to use the predictions to avoid indexing web pages for low re-find users (§7.4). Due to the privacy preserving nature of the data collection, we did not use any text based features in our prediction algorithm. An actual implementation of the system, however, will have access to the queries, the URLs and the text of the web pages. Such a system can use several textual features, further improving the prediction.

5.2.2 Estimating cost of not indexing

Given our estimates of the re-finding probability, we first make a simplifying assumption that the re-finding probabilities of each web page is independent. Then, given a block of B web pages, the cost of not indexing each web page $w \in B$ is the energy to search and download web page w, if w were re-found. Therefore,

$$E[\neg I] = \sum_{w \in B} \text{Energy to search and download } w \times \text{Probability(w will be re-found)} \quad (4)$$

To estimate the energy to search and download w, we build a simple model, similar to the one described in [11]. We download web pages of varying sizes over 3G, and measure the power consumption using the Power Monitor [3]. We use the measurements to build a linear model of energy consumed to download a web page or a search result page of a given size. Similarly, we model the energy consumption over WiFi links. *FindAll* determines the energy consumption to download the search results page and the web page, based on the network in which it is currently operating.

6. IMPLEMENTATION

We implement *FindAll* on the Android operating system. The search engine is adapted from the publicly available Galago search engine [2]. We implement dynamic indexing in Galago to allow blocks of web pages to be indexed into partial indexes. We use the machine learning tool Weka [13] to perform our classifications.

Although we implement *FindAll* over Galago, the indexing strategy is agnostic to the search engine. *FindAll* provides a strategy for creating partial indexes of <word, web page> tuples and merging the indexes in a constrained environment. Any dynamic indexer will need to perform similar indexing and merging activities.

The *FindAll* indexer is implemented as a service on Android, and it periodically indexes web pages, according to the algorithm in Figure 11. We set the time interval T to 5 minutes; i.e., *FindAll* makes indexing decisions every 5 minutes. This ensures that web pages frequently get a chance to be indexed.

Both the cached web pages and the indexes are stored in the memory card. When the user issues a query to *FindAll*, it searches over its local index and returns a search result page with links. When a user clicks on the link, the locally cached web page is retrieved. If the web page is not indexed, *FindAll* falls back to default search, and retrieves the search results from the Internet.

FindAll also indexes web pages on the user's computers. When the phone is being charged, the *FindAll* software on the phone downloads the computer indexes (and the corresponding cached web pages) on to the phone.

In the current version of our implementation, we assume that the user specifies if the query is a re-find query. If it is a re-find query, then the *FindAll* system retrieves the web pages locally; else the web pages are retrieved from the Internet. A mechanism for automatically determining if a query is a re-find request will further enhance the usability of *FindAll*.

We conduct all our evaluations using this *FindAll* implementation using a Motorola DroidX phone. The phone has both a WiFi and a 3G interface, and uses Verizon as the cellular provider.

7. EVALUATION

Our evaluations aim to show that

- *FindAll* improves local availability of web pages and as a result, decreases search latency and 3G data usage.
- *FindAll*'s local availability benefits do not come at an energy cost.
- Compared to alternate indexing strategies that either always index or use the cloud for indexing, *FindAll*'s indexing strategy is more energy efficient and improves availability.
- *FindAll*'s index-based search significantly improves search effectiveness compared to keyword matching or using a database lookup.

7.1 Evaluation methodology

We use the browser logs collected during the user study to evaluate *FindAll*. Our user study collects the time when the user issues a query or requests a web page, the hash values of the query/web page, and the size of the subsequent download. Recall (§3.2.1) that we perform an offline analysis to mark each query/web page as *re-find* or *not re-find*. For our experiments, we assume that this marking represents the user specifying whether a query is a re-find query or a first-time query. All direct web page visits, that are not reached through a search query, are always marked *non re-find*.

Since the traces only contain hashes of the search queries and visited web pages, we map these hashes to real queries and web pages. For each <query hash, web page hash> pair in our logs, we pick an appropriate <real query, real web page> pair from the Microsoft search query logs obtained from Microsoft Live Labs [4]. To perform the mapping, we keep track of the different re-find queries used to search the same web page, both in our traces and in the Microsoft search logs. This allows us to map different variations of queries in our log to different variation of the real queries; recall that 24% of the re-find queries in our logs were different from the original query. Clearly, the same query/web page hash in our logs are mapped to the same query/web page in the Microsoft logs.

To emulate the search session, we replay the log traces. For web pages in the trace that are not reached through a search request, we directly download the web page from the Internet. For other requests, we do the following:

Default search: For each search query, we submit the query to Google, download the search results page, and subsequently download the web page(s) visited by the user through the search query.

FindAll: For each search query marked *re-find*, we submit the search query to *FindAll*. *FindAll* searches the local cache and returns a search results page. If the *FindAll* search is successful, then the results page will contain links to the web pages subsequently visited by the user. These are then fetched from the local cache. If *FindAll* is unsuccessful, either because the page is not indexed/cached yet or if the search algorithm could not retrieve the relevant web page, then we fall back to default search. If the search query is marked *non re-find*, we use default search to retrieve the web pages.

As the search session is played, *FindAll* indexes web pages according to the implementation (§6). The first half of the user's log is used for training *FindAll*'s prediction algorithm. We conduct our experiments using the second half of the logs. We assume that the users use the 3G network for all their interactions. Unless specified, we do not use the cross-device indexes and only use the mobile indexes for search.

7.2 FindAll benefits

FindAll trades storage for Internet connectivity to improve local availability. Figure 12 shows that *FindAll* often retrieves web pages from the local cache, without requiring Internet connectivity: For 6 of the 23 users, *FindAll* retrieves over 40% of the web pages locally. We call these users high re-find users. *FindAll* improves local availability by an average of 20% for the remaining users.

Improving local availability reduces the need for downloading web pages from the Internet, resulting in two key benefits: (1) reduced search latency, and (2) reduced 3G data usage.

FindAll reduces search latency by 2-folds for high re-find users. Figure 13 shows that at its best, *FindAll* reduces per request retrieval latency from 3.35 seconds to 1.65 seconds. As can be expected, *FindAll* does not provide uniform improvements for all users; *FindAll* provides less than 10% latency improvement for 5 of the 23 users, due to the low re-finding behavior of these users.

FindAll reduces search latency by over 3-folds when using cross-device indexes. Figure 14 shows that when browsing logs from all of the user's devices are cached and indexed, the local availability increases even further, reducing the search latency by over 3-folds compared to default search, for certain users.

FindAll reduces 3G data usage by up to 100 MB a month for high re-find users. Figure 15 shows the savings in 3G data usage compared to default web search. On an average, *FindAll* saves 80 MB of 3G data a month for the 6 high re-find users. For one user, the 3G savings is 100 MB a month. If the user has a 500 MB limit on their 3G data plan, a 100 MB savings translates to 20% of the

[4]The search logs are available to research labs, but are not available online.

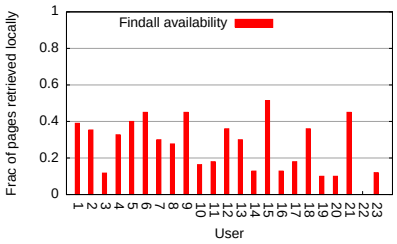

Figure 12: Benefits: *FindAll* increases availability by retrieving web pages from the local cache for over 30% of search queries for 10 of the 23 users.

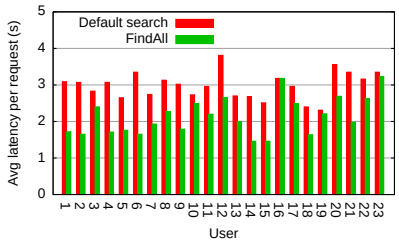

Figure 13: Benefits: Increasing availability results in lower search latency compared to default search. At its best case, *FindAll* reduces latency from 3.35 to 1.65 seconds.

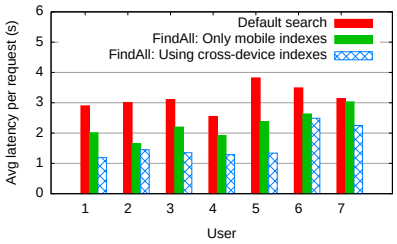

Figure 14: Benefits: Using cross-device indexes that combines browser history from all devices of a given user, further reduces search latency by up to 3-folds compared to default search.

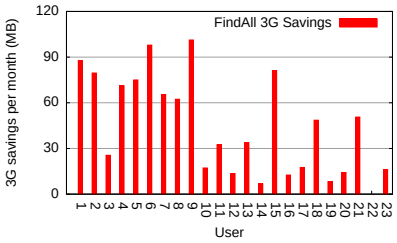

Figure 15: Benefits: Improving local availability results in considerable savings in 3G data usage, that is especially important given tiered data plans.

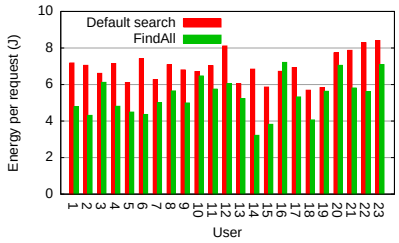

Figure 16: Cost: *FindAll*'s latency and 3G savings does not come at an energy cost. In fact, *FindAll* reduces energy consumption by 30% for high re-find users.

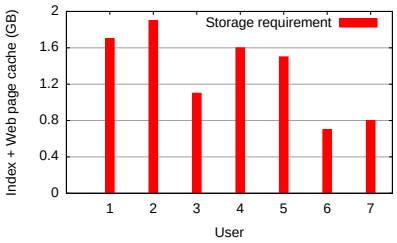

Figure 17: Cost: *FindAll*'s storage requirements, even when indexing across all of the user's devices, is considerably lower than current memory card capacities.

user's 3G limit. As carriers impose stricter 3G caps and introduce tiered payment plans [21], 3G savings using a local search engine will become more and more attractive to users.

7.3 FindAll costs

FindAll's benefit with respect to availability, latency, and 3G data usage, does not come at an energy cost. To perform the energy experiment, we run the user trace on the phone for both default search and *FindAll*, as before, and measure the power consumption using the Monsoon Power Monitor [3]. The power monitor provides accurate energy measurements by sampling the current drawn from the battery with a frequency of 5000 Hz.

FindAll reduces the energy consumption by 30% for high re-find users. We first note that our goal in *FindAll*'s design is to ensure that the energy consumption of *FindAll* is no more than that of default search. Figure 16 shows that *FindAll* in fact saves energy by retrieving web pages locally. For the 6 high re-find users, *FindAll* reduces the energy consumption by over 30%. More importantly, even for users who do not re-find often, *FindAll* does not increase overall energy consumption.

FindAll's energy benefits are primarily due to its indexing algorithm; *FindAll* takes into account the re-finding patterns of each user before making an indexing decision. We show in §7.4 that, instead, if web pages were indexed as soon as they arrive, a local search engine will consume significantly more energy compared to default search.

FindAll's storage requirements are modest. Figure 17 presents the storage requirement for *FindAll*'s indexes and cached web pages for 30 days. We present the storage requirement for the 7 users for whom we store cross-device indexes; these users have the largest storage requirement (the remaining users had a storage requirement of less than 0.3 GB per month). At its maximum, the storage requirement for *FindAll* is less than 1.9 GB; most phones have external memory cards that have a capacity of 64 GB or more. Thus, a 1-month index occupies only a small fraction of the available storage.

7.4 Comparing with alternate indexing strategies

FindAll balances availability and energy cost using its indexing algorithm. At its core, *FindAll* determines when to index a block of web pages by predicting the users re-finding behavior. Below, we compare *FindAll*'s indexing strategy with 4 alternate strategies.

- *Always-Index*: Indexes every web page as soon as it is downloaded (i.e., the block size is always 1).
- *Cloud-Index*: Indexes web pages in the cloud and periodically uploads the incremental indexes to the mobile.
- *Fixed-Block-Index*: Indexes web pages in fixed-sized blocks.
- *Never-Index*: No indexing; default search.

For clarity, we present the results of our experiments for a subset of 6 users, two users in each of the three categories: *High re-find users*, for whom 40% or more web pages are retrieved locally, *Medium re-find users*, for whom 15% to 25% of web pages are retrieved locally, and *Low re-find users*, for whom less than 10% of the web pages are retrieved locally. We also repeated the experiments for the remaining users and found the results to be qualitatively similar (not shown here).

When using the Cloud-Index strategy, we upload data from the cloud every 30 minutes; when using Fixed-Block-Index strategy, we index documents in block sizes of 5 web pages. We evaluated

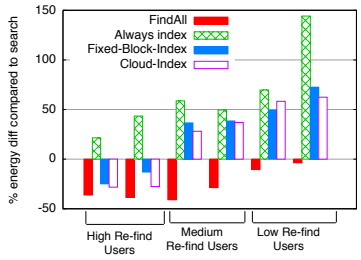

Figure 18: Comparing energy cost: The Always-Index strategy consumes considerably more energy compared to default, for all users. The Cloud-Index and Fixed-Block-Index strategies consume more energy compared to default search for low re-find users.

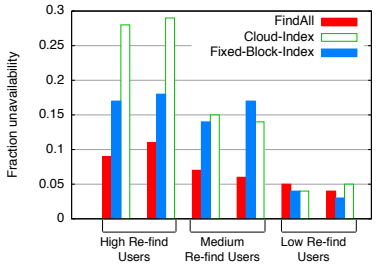

Figure 19: Comparing unavailability: The Cloud-Index strategy is up to 67% more unavailable, and Fixed-Block-Index strategy is up to 39% more unavailable compared to *FindAll* for high re-find users. High unavailability will result in lower latency and 3G saving benefits.

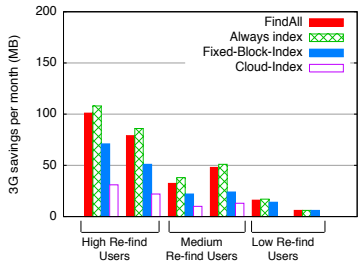

Figure 20: Comparing 3G saving benefits: The 3G savings of the Always-Index strategy is only 5% better than *FindAll*, even though it has 0 unavailability. *FindAll* provides more 3G savings than Cloud-Index and Fixed-Block-Index strategies by improving availability.

other upload periods and block sizes, and chose these values as they provided the best energy and availability trade-off.

All alternate indexing strategies consume significantly more energy compared to default search.. Figure 18 compares the increase/decrease in the percentage energy consumption compared to default search. The Always-Index strategy has the worst energy characteristics, and at its worst, incurs a 140% increase in energy consumption compared to default search. For low re-find users, all three alternate indexing strategies perform worse than default search with respect to energy. This is because, the alternate strategies do not adapt to the user's re-finding patterns, and instead index web pages even if the user is unlikely to re-find. For low re-find users, indexing web pages do not provide any energy benefit, resulting in a net energy wastage. In contrast, *FindAll* does not index often for low re-find users, reducing its energy consumption.

At its peak, the absolute energy difference between the Always-Index strategy and default search is 140 Joules (not shown in figure). Going beyond web search, this difference will become even more significant as local indexes are used to retrieve content for other apps. For example, we repeated the above experiment using the desktop search history from our logs; users download considerably more web pages on their desktops than on their mobile phones. The absolute energy difference between Always-Index and default search when using desktop logs increased to 960 Joules.

FindAll improves local availability compared to Cloud-Index and Fixed-Block-Index strategies. Figure 19 compares the unavailability across the different indexing strategies. We define *unavailability* as the fraction of time user issues a re-find query, but the corresponding web page is not available in the index. For high re-find users, the indexing frequency of Cloud-Index and Fixed-Block-Index strategies are too low; as a result, several re-find queries are not satisfied locally. The unavailability of the Always-Index strategy is 0, since the web pages are indexed immediately. Although the Always-Index strategy has lowest unavailability, we see next that its benefits are only slightly better than that of *FindAll*.

FindAll saves significantly more 3G data compared to both Fixed-Block-Index and Cloud-Index strategy. Recall that one of *FindAll*'s benefits is that it reduces 3G data usage compared to default web search by retrieving web pages locally. Figure 20 shows that by reducing unavailability, *FindAll* substantially improves 3G savings compared to the Cloud-Index and Fixed-Block-Index strategies. The Cloud-Index strategy performs especially poorly with respect to 3G data usage because it uploads indexes periodically through the cloud. For low re-find users, we omit the 3G savings using the Cloud-Index strategy, because the Cloud-Index strategy uses more 3G data than default web search. More importantly, even though the Always-Index strategy has 0 unavailability, it only saves 3G data usage by an additional 5% over *FindAll*.

7.5 FindAll vs Keyword matching vs Database lookup

Next, we compare *FindAll* with two alternate search techniques: (i) Keyword matching: Returns all web pages that contain the query terms in no particular order, and (ii) Database lookup: Stores already seen web pages in a database, as <query, web page> pairs. Web pages are retrieved by using the query as the key.

7.5.1 *Search effectiveness*

The effectiveness of a search engine depends on whether the search results contain the relevant web page that the user is searching for. This is especially true in mobile phones where only 3-4 search results can be presented on the screen. If the relevant web page is returned as, say, the 20th search result, the user has to scroll through multiple screens before finding the relevant page.

In Figure 21 we ran the three alternate strategies, and counted the number of times the relevant web page was within the top 3 search results. The relevant web page is the page the user eventually clicks on, and we obtain this information from our logs. *FindAll returns the relevant web page 96% of the time within the top 3 search results, compared to 64% when using keyword matching, and 71% when using database lookup.*

The database lookup fails to return the relevant web page if the original query and the re-find query are different. Recall (§3) that the re-find query and the original query are different for 24% of the queries in our log. For a small percentage of cases (8%), the database lookup failed to return the relevant web page because the original web page was not retrieved through a search query. This can happen, for example, if the user visits a web page originally via social media but wants to re-find the web page later through search. Keyword matching is not effective in searching for a specific web page, since it returns every web page that contains the query terms.

7.5.2 *Performance under no connectivity*

To understand the importance of search effectiveness, we conduct experiments in a simulated *no connectivity* environment. One of the

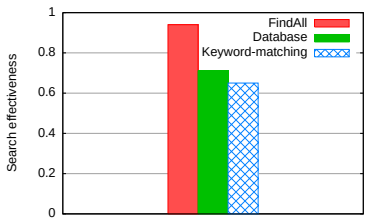

Figure 21: *FindAll* returns the relevant web page 96% of the time within the top 3 search results, compared to 64% when using keyword matching and 71% when using database lookups.

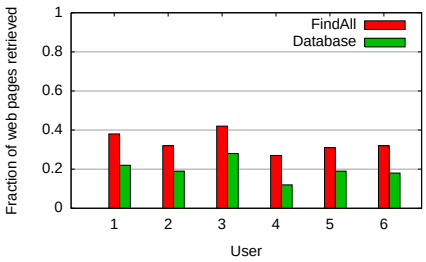

Figure 22: *FindAll* doubles the number of web pages that are retrieved even in the absence of connectivity, compared to using a database lookup.

main advantages of a local search engine is that web pages can be retrieved even when there is no connectivity.

Figure 22 shows the percentage of web pages retrieved if we assume that there is no connectivity for 50% of the trace for each user. We randomly choose the periods of no connectivity and repeat the experiment 5 times with different seed values. We present the average. Note that this scenario is not far fetched; 4 of the 23 users in our user study did not have a 3G data plan. Such users may not be connected to the Internet for large parts of their day.

We present the results for the 6 high re-find users. *FindAll* retrieves the relevant web page for 35–40% of the search queries even when there is no connectivity. In contrast, search using a database lookup only returns the relevant web page for 20% of the search queries. This is a direct consequence of the poorer search effectiveness of a database lookup.

8. RELATED WORK

Our work is inspired by several related research efforts. Below we contrast our work from existing work.

8.1 Local database

The Pocket Cloudlets [15] work explores storage architectures that allow mobile phones to trade-off connectivity for local storage. The goals of *FindAll* allign well with the goals of Pocket Cloudlets. The core idea in Pocket Cloudlets is to store the *search results page* in a local database and use the query as the key to the lookup database. The search results are retrieved in response to a query using the database lookup; however, the web page itself is not stored locally. The web page needs to be re-downloaded from the Internet. The goal of PocketCloudlets is to only eliminate the latency in retrieving the search results page, and not the web page itself.

The database lookup cannot retrieve the web page if the re-find query is different from the original query. *FindAll* uses a full blown search engine; as a result, *FindAll* works even if the re-find query terms are different from the original query terms (§7.5). In addition, *FindAll* stores the web pages in the phone, so that they can be retrieved locally, unlike PocketCloudlets. This results in additional benefits with respect to latency and 3G data savings.

PocketWeb [18], a system built on top of the PocketCloudlets system, dynamically updates web pages stored in the mobile cache, by learning the browsing behavior of users. The goals of PocketWeb is complimentary to the goals of *FindAll*. In the current implementation of *FindAll*, the web pages stored in the cache are not refreshed. Periodically refreshing the local web pages using a PocketWeb-like system can further increase the utility of *FindAll*.

Finally, the authors of PocketWeb [18] and PocketCloudlets [15] conduct a mobile user study using Bing search logs. In an earlier section (§3), we detail the similarities and differences between these studies and the *FindAll* study.

8.2 Cloud solutions

Several commercial solutions allow users to search through their history, or sync the browser history across devices. For example, search engines such as Bing and Google provide a history feature that allows (logged-in) users to search through their search history. The search engines store the history in the cloud. The Chrome Sync feature on the chrome browser allows users to sync all the URLs they visit, across their devices. The Chrome Sync feature augments search, but the user still needs to connect to the Internet to retrieve the web page.

The search engine and the chrome browser solutions require that the user have good Internet connectivity, whereas *FindAll* operates even when connectivity to the Internet is poor or unavailable. Further, *FindAll* allows users to store their search history under their own control, alleviating some of the privacy concerns of letting a search provider store the history.

8.3 Cache-based approaches

Browser history and bookmarks store a link to a web page previously downloaded by the user. To re-find a web page, the user has to manually browse through the history, which is tedious on mobile phones.

Similarly, Web caching solutions [9, 8, 12, 28] store a copy of the web page locally, to reduce latency and improve performance of web browsing. However, web caching can be leveraged for re-finding only if the user has a link to the URL; usually users do not remember the URL. Some browsers such as Opera and Safari cache URLs and match new URL requests character by character to cached ones, to provide hints to users. However, even these solutions require that the users remember partial URLs. Further, previous research has shown that users rarely look through their history or their cache to re-find web pages [10].

8.4 Cross-device synchronization

The Dessy system [16] is a mobile desktop search system. Dessy indexes files across multiple devices and uses the mobile phone to search through the index. The authors build the search engine indexes in the cloud and upload the index to the mobile phone. Instead, *FindAll* implements a search engine locally on the phone, and we show that the local index performs better than a cloud-based index.

Finally, *FindAll* goes beyond search–*FindAll* indexes all web pages that the user downloads; not just those found through a search engine. Today, a large proportion of browsing activity originates from social networking apps and emails. Say, Bob reads an article that he finds through facebook, and wishes to re-find this article. The search engine or a database lookup will not have stored this

article because it was not found through search. However, *FindAll* indexes all web pages, and therefore can retrieve the article for Bob. The search interface in *FindAll* can also potentially leverage the re-finding characteristics of other apps, including social networking apps, maps, and movie databases.

Similarly, *FindAll* is useful not only to re-find previously viewed web pages, but also to search within previously viewed web pages. For example, a user searching for hotels in Alaska may want to look for "Valet Parking" within the hotels that she previously viewed. By building an index, *FindAll* allows users to issue new queries to their local cache.

9. CONCLUSIONS AND FUTURE WORK

In this work, we developed *FindAll*, a local search engine that supports re-finding on mobile phones. To aid with its design and to better understand mobile re-finding, we conducted a user study with 23 users over 30 days. The study showed that users have diverse browsing and re-finding habits, and 45% of the URLs are re-found within 50 minutes. Therefore, the key design goal in *FindAll* is to design a search engine that indexes web pages locally and soon after the URL is first visited, to improve availability. The challenge is in designing a search engine for a resource-constrained mobile environment, while adapting to the user's re-finding behavior. To this end, *FindAll* indexes web pages only when the expected energy benefit of indexing outweigh the indexing cost. *FindAll* estimates the benefits of indexing by learning the re-finding patterns of each user and predicting the re-finding probability. We implemented *FindAll* over Android by adapting a publicly available search engine called Galago. Our evaluations show that *FindAll* reduces search latency by two-folds for users who re-find often. *FindAll* also reduces 3G data usage by up to 100 MB a month by serving over 40% of the web pages locally.

10. ACKNOWLEDGEMENTS

We thank our shepherd, Y. Charlie Hu, and all anonymous reviewers. Their reviews and comments greatly helped improve the presentation of this paper. We thank all the participants of our user study who helped us collect data. This work was supported in part by an NSF Computing Innovation Fellowship and NSF grant CNS-1217644.

11. REFERENCES

[1] Alexa top 500 websites.: `http://www.alexa.com/topsites`.

[2] Galago search engine: `http://www.galagosearch.org/`.

[3] Monsoon power monitor.: `http://www.msoon.com/`.

[4] D. Abrams, R. Baecker, and M. Chignell. Information archiving with bookmarks: personal web space construction and organization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 41–48, 1998.

[5] E. Adar, J. Teevan, and S. T. Dumais. Resonance on the web: web dynamics and revisitation patterns. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1381–1390, 2009.

[6] A. Aula, N. Jhaveri, and M. Käki. Information search and re-access strategies of experienced web users. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 583–592, New York, NY, USA, 2005. ACM.

[7] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3g using wifi. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 209–222, New York, NY, USA, 2010. ACM.

[8] G. Barish and K. Obraczka. World wide web caching: Trends and techniques. *IEEE Communications Magazine*, 38:178–184, 2000.

[9] E. Benson, A. Marcus, D. Karger, and S. Madden. Sync kit: a persistent client-side database caching toolkit for data intensive websites. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 121–130, New York, NY, USA, 2010. ACM.

[10] A. Cockburn, S. Greenberg, S. Jones, B. Mckenzie, and M. Moyle. Improving web page revisitation: analysis, design and evaluation. *IT & Society*, 1:159–183, 2003.

[11] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62, New York, NY, USA, 2010. ACM.

[12] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24:51–78, January 2006.

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[14] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 165–178, New York, NY, USA, 2010. ACM.

[15] E. Koukoumidis, D. Lymberopoulos, K. Strauss, J. Liu, and D. Burger. Pocket cloudlets. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '11, pages 171–184, New York, NY, USA, 2011. ACM.

[16] E. Lagerspetz, T. Lindholm, and S. Tarkoma. Dessy: Towards flexible mobile desktop search. In *Proceedings of the DIALM-POMC International Workshop on Foundations of Mobile Computing, Portland, Oregon, August 16, 2007, CD-ROM*. ACM, 2007.

[17] N. Lester, A. Moffat, and J. Zobel. Efficient online index construction for text databases. *ACM Trans. Database Syst.*, 33:19:1–19:33, September 2008.

[18] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas. Pocketweb: instant web browsing for mobile devices. *SIGARCH Comput. Archit. News*, 40(1):1–12, Mar. 2012.

[19] Microsoft. See your search history: `http://onlinehelp.microsoft.com/en-us/bing/ff808483.aspx`.

[20] P. Shodjai. Your slice of the web: `http://tinyurl.com/24vhmw`, 2007.

[21] Sprint Capping Unlimited 3G Data Service at 5GB. `http://gizmodo.com/391887/oh-no-sprint-capping-unlimited-3g-data-service-at-5gb`, 2008.

[22] J. Teevan, E. Adar, R. Jones, and M. A. S. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 151–158, 2007.

[23] S. K. Tyler and J. Teevan. Large scale query log analysis of re-finding. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 191–200, 2010.

[24] S. K. Tyler, J. Wang, and Y. Zhang. Utilizing re-finding for personalized information retrieval. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1469–1472, 2010.

[25] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How effective is mobile browser cache? In *Proceedings of the 3rd ACM workshop on Wireless of the students, by the students, for the students*, S3 '11, pages 17–20, New York, NY, USA, 2011. ACM.

[26] I. Witten, A. Moffat, and T. Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

[27] J. Wortham. Customers Angered as iPhones Overload 3G. `http://www.nytimes.com/2009/09/03/technology/companies/03att.html?_r=2&partner=MYWAY&ei=5065/`, 2009.

[28] Y. Xie and D. R. O'Hallaron. Locality in Search Engine Queries and Its Implications for Caching. In *Proc. IEEE Infocom*, pages 1238–1247, June 2002.