Locality in Search Engine Queries and Its Implications for Caching

Yinglian Xie and David O'Hallaron
Department of Computer Science, Carnegie Mellon University
Email:{ylxie, droh}@cs.cmu.edu

Abstract—**Caching is a popular technique for reducing both server load and user response time in distributed systems. In this paper, we consider the question of whether caching might be effective for search engines as well. We study two real search engine traces by examining query locality and its implications for caching. Our trace analysis results show that: (1) Queries have significant locality, with query frequency following a Zipf distribution. Very popular queries are shared among different users and can be cached at servers or proxies, while 16% to 22% of the queries are from the same users and should be cached at the user side. Multiple-word queries are shared less and should be cached mainly at the user side. (2) If caching is to be done at the user side, short-term caching for hours will be enough to cover query temporal locality, while server/proxy caching should use longer periods, such as days. (3) Most users have small lexicons when submitting queries. Frequent users who submit many search requests tend to reuse a small subset of words to form queries. Thus, with proxy or user side caching, prefetching based on user lexicon looks promising.**

I. Introduction

CACHING is an important technique to reduce server workload and user response time. For example, clients can send requests to proxies, which then respond using locally cached data. By caching frequently accessed objects in the proxy cache, the transmission delays of these objects are minimized because they are served from nearby caches instead of remote servers. In addition, by absorbing a portion of the workload, proxy caches can increase the capacity of both servers and networks, thereby enabling them to service a potentially larger clientele.

We are interested in the question of whether caching might be effective for search engines as well. Because serving a search request requires a significant amount of computation as well as I/O and network bandwidth, caching search results could improve performance in three ways. First, repeated query results are fetched without redundant processing to minimize the access latency. Second, because of the reduction in server workload, scarce computing cycles in the server are saved, allowing these cycles to be applied to more advanced algorithms and potentially better results. Finally, by disseminating user requests among proxy caches, we can distribute part of the computational tasks and customize search results based on user contextual information.

Although Web caching has been widely studied, few researchers have tackled the problem of caching search engine results. While it is already known that search engine queries have significant locality, several important questions are still open:

- Where should we cache search engine results? Should we cache them at the server's machine, at the user's machine, or in intermediate proxies? To determine which type of caching would result in the best hit rates, we need to look at the degree of query popularity at each level and whether queries will be shared among different users.
- How long should we keep a query in cache before it becomes stale?
- What might other benefits accrue from caching? Since both proxy and client side caching are distributed ways of serving search requests, can we prefetch or re-rank query results based on individual user requirements?

With respect to the above questions, we study two real search engine traces. We investigate their implications for caching search engine results. Our analysis yielded the following key results:

- Queries have significant locality. About 30% to 40% of queries are repeated queries that have been submitted before. Query repetition frequency follows a Zipf distribution. The popular queries with high repetition frequencies are shared among different users and can be cached at servers or proxies. Queries are also frequently repeated by the same users. About 16% to 22% of all queries are repeated queries from the same users, which should be cached at the user side. Multiple-word queries are less likely to be shared by different users. Thus they can also be cached mainly at the user side.
- The majority of the repeated queries are referenced again within short time intervals. But there remains a significant portion of queries that are repeated within relatively longer time intervals. They are largely shared by different users. So if caching is to be done at the user side, short-term caching for hours will be enough to cover query temporal locality, while server/proxy caching should be based on longer periods, on the order of days.
- Most users have small lexicons when submitting queries. Frequent users who submit many search requests tend to reuse a small subset of words to form queries. Thus, with proxy or user side caching, prefetching based on user lexicons is promising. Proxy or user side caching also provide us with opportunities to improve query results based on individual user preferences, which is an important future research direction.

In the rest of the paper, we first discuss related works in Section I-A. We then describe the traces we analyzed and summarize the general statistics of the data in Section II. In Section III, we focus on repeated queries and discuss query locality in both traces. Section IV presents our findings about user lexicon analysis and its implications. Finally, we review analysis results and discuss possible future research directions.

A. Related Work

Due to the exponential growth of the Web, there has been much research on the impact of Web caching and how to maximize its performance benefits. Most Web browsers support caching documents in the client's memory or local disk to re-

duce the response time of the client. Deploying proxies between clients and servers yields a number of performance benefits. It reduces server load, network bandwidth usage as well as user access latency [1], [2], [3], [4]. Prefetching documents to proxies or clients has been studied for further performance improvement by utilizing user access patterns [5], [6].

There are previous studies on search engine traces. Jasen *et al* analyzed the Excite search engine trace to determine how users search the Web and what they search for [7]. Silverstein *et al* analyzed the Altavista search engine trace [8], studying the interaction of terms within queries and presenting results of a correlation analysis of the log entries. Although these studies have not focused on caching search engine results, all of them suggest queries have significant locality, which particularly motivates our work.

Query result caching has already been investigated as a way to reduce the cost of query execution in distributed database systems by caching the results of "similar" queries [9], [10]. Recently, Markatos has studied the query locality based on the Excite trace and shown that $20\% \sim 30\%$ of the queries are repeated ones [11], [12]. He suggests a server-side query result cache and has mainly focused on leveraging different cache replacement algorithms. Our work builds on this by systematically studying query locality and deriving the implications for caching search engine results.

II. The Search Engine Query Traces

The two traces we analyzed are from the Vivisimo search engine [13] and the Excite search engine [14]. In this section, we briefly take a look at the two search engines and review their trace data.

A. The Vivisimo and the Excite Search Engines

Vivisimo is a clustering meta-search engine that organizes the combined outputs of multiple search engines. Upon reception of each user query, Vivisimo combines the results from other search engines and organizes these documents into meaningful groups. The groupings are generated dynamically based on extracts from the documents, such as titles, URLs, and short descriptions. By default, Vivisimo refers to one or multiple major search engines, including (ca. Feb. 2001) Yahoo, Altavista, Lycos, Excite, and returns 200 combined results using logic operation 'ALL'. Vivisimo also supports advanced search options where users can specify which search engines to query, the number of results to be returned, and which logic operation to be performed on the query, including ANY, PHRASE and BOOLEAN.

Excite is a basic search engine that automatically produces search results by listing relevant web sites and information upon reception of each user query. Capitalization of the query is disregarded. The default logic operation to be performed is 'ALL'. It also supports other logic operations like 'AND', 'OR', 'AND NOT'. More advanced searching features of Excite include wide card matching, 'PHRASE' searching and relevance feedbacks.

B. The Query Trace Descriptions

The Vivisimo query trace was collected from January 14, 2001 to February 17, 2001, soon after the Vivisimo launched in early January, 2001. The trace captures the behavior of early adopters who may not be representative of a steady state user group. The Excite trace was collected on December 20, 1999. Although the Vivisimo trace and the Excite trace were collected independently at different times, over different temporal periods, and with different user populations, their statistical results are similar. Thus they are both representative.

In both traces, each entry contains the following fields of interest:

- **an anonymous ID** identifying the user IP address. For privacy reasons, we do not have actual user IP addresses. Each IP address in the original trace is replaced by a unique anonymous ID.
- **a timestamp** specifying when the user request is received. The timestamp is recorded as the wall clock time with a 1 second resolution.
- **a query string** submitted by the user. If any advanced query operations are selected, they will also be specified in this string.
- **a number** indicating whether the request is for next page results or a new user query.

C. Statistical Summaries of the Traces

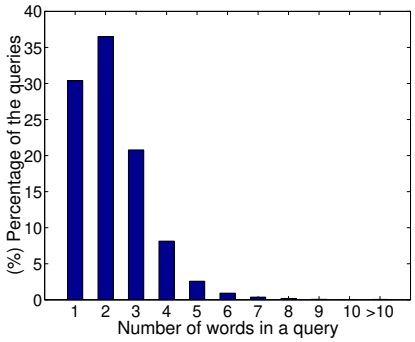
After extracting a query string from each trace entry, we transform the string to a uniform format for easy processing. We remove stopwords from the query because most search engines discard them anyway. We convert all query terms to lower case. Thus the query is case insensitive, which is also typical for search engines. However, the removal of the stopwords and the upper-to-lower case conversion actually has little impact on our analysis results. It affects our statistics by about 1% and the effect could be ignored. In the rest of the paper, we use "query" to denote all the words as a whole entered by the user in a query submission, and "words" or "terms" to denote the individual words contained in a user query. Because we cannot distinguish users who used multiple IP addresses or users who shared IP addresses in the trace, we uniformly use "user" to denote the IP address where the query came from.

Fig. 1 summarizes the statistics about the traces. The Excite trace lasts for 8 hours in a single day. The Vivisimo trace was collected more recently over a period of 35 days. Thus, the two traces provide us with both long-term and short-term views to user queries since they stand for different time scales. Several facts are obvious from this summary for both traces.

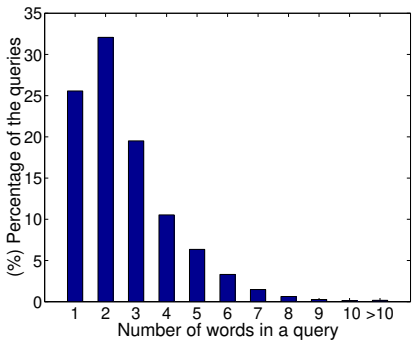
1. Users do not issue many next-page requests. Fewer than two pages on average are examined for each query.
2. Users do repeat queries a lot. In the Vivisimo trace, over **32%** of the queries are repeated ones that have been submitted before by either the same user or a different user. In the Excite trace, more than **42%** of the queries are repeated queries.
3. The majority of users do not use advanced query options: **97%** of the queries from the Vivisimo trace and **93%** of the queries from the Excite trace use the default logic operations offered by the corresponding search engines.
4. Users on average do not submit many queries. The average numbers of queries submitted by a user are **5.48** and **3.69**, respectively.
5. About **70%** of the queries consist of more than one word, although the average query length is fewer than three terms, which

Trace	Vivisimo trace	Excite trace
Start-time	14/Jan/2001:04:02	20/Dec/1999:09:00
Stop-time	17/Feb/2001:00:00	20/Dec/1999:16:59
Number of bytes	657,623,865	118,318,788
Number of HTTP requests	2,588,827	not known
Number of user queries (including next page requests)	205,342	2,477,283
Number of user queries (excluding next page requests)	110,881	1,920,997
Number of distinct user queries	75,343	1,099,682
Number of multiple word queries	77,181	1,429,618
Number of queries using default logic operation(ALL)	107,880	1,792,174
Number of users	20,220	520,883
Average queries submitted per user	5.48	3.69
Average number of terms in a query	2.22	2.63

Fig. 1. Trace statistical summary. The number of HTTP requests cannot be inferred from the Excite trace since the trace did not contain information about HTTP requests from users.

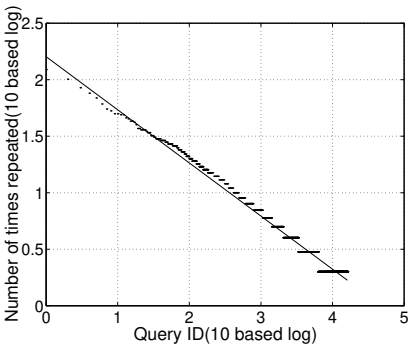


(a) The Vivisimo Trace

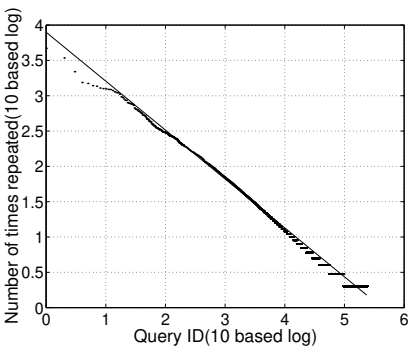


(b) The Excite Trace

Fig. 2. User query distribution according to the number of words in each query



(a) The Vivisimo Trace



(b) The Excite Trace

Fig. 3. Distribution of the query repetition frequency (logarithmic scales on both axes). The query IDs are sorted by the number of times being repeated.

is short. Fig. 2 shows the query length distributions of the two traces. We can observe that most of the queries are fewer than five terms long.

Overall, these results are consistent with those reported in [7] and [8] and thus are not surprising.

III. Query Locality and its Implications

As mentioned in Section II-C, 32% to 42% of the queries in the trace are repeated queries, which suggests caching as a way to reduce server workload and network traffic. In this section, we focus on the study of repeated queries, and discuss how the locality in these queries motivates different kinds of caching.

A. Query Repetition Distribution

Among the **35,538** queries that are repeated ones in the Vivisimo trace, there are **16,162** distinct queries. This means on average, each repeated query was submitted **3.20** times. Similarly, each repeated query in the Excite trace was submitted **4.49** times, with **235,607** distinct queries among **821,315** repeated ones. Interestingly, we found that the query repetition frequencies can be characterized by Zipf distributions. Fig. 3 plots the distributions of the repeated query frequencies for the

traces. Each data point represents one repeated query, with the X axis showing queries sorted by the repetition frequency: the first query is the most popular one, the second query is the second-most repeated query, and so on until we reach the query number 16,162 and 235,607 which were only repeated a single time.

We are interested in whether the repeated queries were from the same users or shared by different users. Out of the **32.05%** of the repeated queries in the Vivisimo trace, **70.58%** are from the same users. In the Excite trace, we have **42.75%** of the repeated queries, of which **37.35%** are repeated by the same users. Therefore, about **22%** and **16%** of all queries are repeated queries from the same users in the Vivisimo trace and the Excite trace, respectively.

We then counted the number of distinct queries that were repeated by more than one user. There are **5,675** such queries from the Vivisimo trace and **135,020** such queries from the Excite trace. The distributions of these queries, however, are non-uniform according to the frequency of the repetitions. We define the "shareness" of a query as the number of distinct users who submitted the query over some time period. Thus, shareness is a measure of a query's popularity across multiple users. Fig. 4 shows the shareness of the queries based on their repetition frequencies. We find that queries repeated at least 10 times are more likely to be shared by multiple users than queries repeated fewer than 10 times. In the Vivisimo trace, among the **404** queries that were repeated at least 10 times, **78.96%** were shared by different users. For the queries that were repeated fewer than 10 times, only **33.99%** were shared. The same trend holds for the Excite trace. Among the **12,071** queries repeated at least 10 times, as many as **99.08%** are shared, while only **55.05%** are shared over the queries repeated fewer than 10 times.

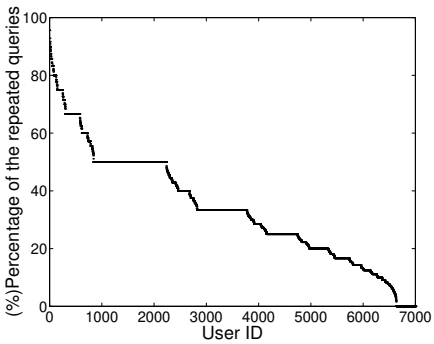
Given the Zipf distribution followed by the query repetition frequency, there are a small number of queries that were repeated very frequently. They were both shared by different users and repeated by the same users. There also exist a large number of queries that were repeated only a few times, which were mostly from the same users.

These results suggest that we should cache query results in different ways. For the small number of queries that were very popular, we could cache them at search engine servers to exploit the high degree of shareness among different users. We could also cache them at the user side, which would reduce the server processing overheads due to their high repetition frequencies. For the other type of queries that are only repeated by the same users, caching them at servers is not very effective, considering the limited server resources and the diverse requirements from the large number of users. Instead, we could consider caching these queries at the user side according to the unique requirement of each individual user.

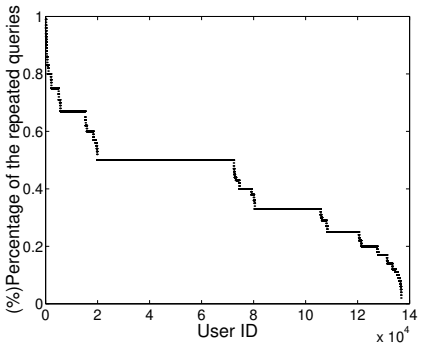
B. Query Locality Based on Individual User

The query shareness distribution indicates that query locality exists with respect to the same users as well as among different users. More specifically, most of the queries at the long tail were repeated by the same users. Therefore, we further explore query locality based on each individual user.

There are **20,220** users recorded in the Vivisimo trace, of whom **6,628** repeated queries at least once. Each user on average repeated **5.36** queries. In the Excite trace, there are **520,883** users who submitted queries. Among them, **136,626** users repeated queries, with each user repeating **6.01** queries on average. Fig. 5 plots the percentage of the repeated queries over the total number of queries submitted by each user. For example, if a user submitted 10 queries in total, and out of the 10 queries, 5 were repeated ones that had been submitted before, then the user has 50% of the repeated queries. From both traces, we observe that **80%** to **90%** of the users who repeated queries had at least **20%** of the repeated queries. Around half of these users had at least **50%** of the repeated queries.



(a) The Vivisimo Trace



(b) The Excite Trace

Fig. 5. The percentage of the repeated queries over the number of queries submitted by each user (Only those users who repeated queries are plotted here).

From these results, we can see that not only a lot of users repeated queries, but each user also repeated queries a lot. Since 16% to 22% of all queries were repeated by the same users and that search engine servers can cache only limited data, caching these queries and query results based on individual user requirements in a more distributed way is important. It reduces user query submission overheads and access latencies as well as server load. By caching queries at the user side, we also have the opportunity to improve query results based on individual user context, which cannot be achieved by caching queries at a centralized server.

C. Temporal Query Locality

In this section, we quantify the notion of temporal query locality, that is, the tendency of users to repeat queries within a short time interval. Fig. 6 shows the number of queries repeated within different time intervals. Overall, queries were repeated

Type of queries	Queries repeated $\geq$ 10 times		Queries repeated $<$ 10 times	
	shared	not shared	shared	not shared
Number of appearances	319	85	5356	10402
Percentage	78.96%	21.04%	33.99%	66.01%

Fig. 4 (a) Shareness of the queries from the Vivisimo trace

Type of queries	Queries repeated $\geq$ 10 times		Queries repeated $<$ 10 times	
	shared	not shared	shared	not shared
Number of appearances	11,960	111	123,060	100,476
Percentage	99.08%	0.92%	55.05%	45.95%

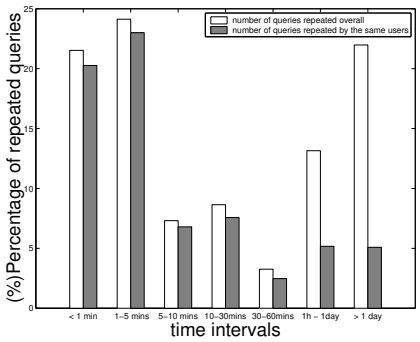
Fig. 4 (b) Shareness of the queries from the Excite trace

Fig. 4. Shareness of the queries based on their repetition frequencies. A query was shared if it was submitted by more than one user in the exact same form. Otherwise, a query was only submitted by a single user and was not shared

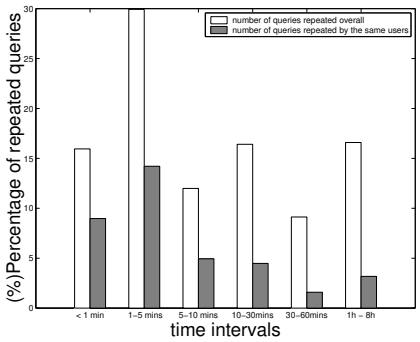
within short periods of time. In the Vivisimo trace, about **65%** of the queries were repeated within **an hour**. Since the Excite trace lasts for only 8 hours, as many as **83%** of the queries were repeated within **an hour**.

In the Vivisimo trace, these queries were mostly from the same users. More specifically, **45.5%** of the queries were repeated by the same users within **5 minutes**, which is very short. There are also many queries that were repeated over relatively longer time intervals; they are largely shared by different users. Out of the **21.98%** of the queries that were repeated over a day, only **5.09%** came from the same users. The Excite trace generally has a smaller percentage of the queries repeated by the same users. But we still observe the same pattern: the shorter the time interval, the more likely a query will be repeated by the same user.

These statistics suggest that if caching query results is to be done at servers or proxies, then we should consider longer periods of caching in order to exploit the maximum shareness among different users. Usually, caching query results for a long time is more likely to result in stale data. Since this is to be performed by the server, the stale data can be removed whenever the server updates the results. If caching is to be performed at the user side, then a short period of caching would be enough to cover most of the temporal query locality, which is also less likely to result in stale data.



(a) The Vivisimo Trace



(b) The Excite Trace

Fig. 6. Repeated query distribution within different time intervals

D. Multiple-Word Query Locality

Multiple-word queries are important because they account for about **70%** of the queries. Multiple-word queries also have significant locality, which will be discussed in this section.

Fig. 7 summarizes the statistics about multiple-word queries. The Excite trace has a larger portion of multiple-word queries than the Vivisimo trace, but both traces have as many as **62%** of the multiple-word queries over the repeated ones. We observe that multiple-word queries are repeated less frequently compared with single-word queries. In the Vivisimo trace, each repeated multiple-word query was submitted **3.00** times on average, compared with **3.63** times for single-word cases. In the Excite trace, each repeated multiple-word query was submitted **3.77** times, compared with **7.12** times for single-word cases.

We also observe that multiple-word query locality mostly exists among the same users, that is, multiple-word queries have less degree of shareness. Fig. 8 shows the comparison between multiple-word queries and single-word queries with respect to their degrees of shareness. From both traces, we can see that multiple-word queries are less likely to be shared by different users. Each shared multiple-word query also tends to be shared by fewer users. This is easily explained because the chances for different users to submit the same multiple-word queries are much smaller than those for single-word queries.

From the above results, we can see that multiple-word query locality is significant. Caching multiple-word queries is more promising because it takes more time to compute multiple-word query results. Since multiple-word queries have less degree of shareness, we could cache them mainly at the user side.

	Number of appearance		Percentage	
	Vivisimo	Excite	Vivisimo	Excite
Multiple word queries	77,181	1,429,618	69.61% (over the number of the queries)	74.42%
Unique multiple word queries	55,193	924,546	73.26% (over the number of the unique queries)	84.07%
Multiple word queries that were repeated	21,995	505,249	61.89% (over the number of the repeated queries)	61.52%
Unique multiple word queries that were repeated	11,020	182,335	68.18% (over the number of unique repeated queries)	77.39%
Unique multiple word queries that were submitted by more than one user	3,181	101,098	5.76% (over the number of unique multiple word queries)	11%

Fig. 7. Multiple-word query summary

Type of queries	Shared multiple-word queries		Shared single-word queries	
	Vivisimo	Excite	Vivisimo	Excite
Percentage	5.76%	10.93%	12.28%	19.37%
Number of users	2.53	3.95	3.24	7.38

Fig. 8. The comparison between multiple-word queries and single-word queries with respect to the degrees of the shareness in both traces. 'Percentage' means the portion of shared multiple-word or single-word queries over the total number of multiple-word or single-word queries. 'Number of users' means the average number of users sharing each such query.

E. Users with Shared IP Addresses

Some users use dial up services such as AOL to access the Internet. For those users, their IP addresses are dynamically allocated by DHCP servers. Unfortunately, there is no common way to identify these kinds of users. This impacts our analysis in two ways. First, because different users can share the same IP address at different times, their queries seem like they come from the same user, leading to an overestimate of the query locality from the same users. Second, because the same users can use different IP addresses at different times, it is also possible for us to underestimate the query locality from the same users. Since AOL clients will have keyword "AOL" in their user-agent fields, which were recorded in the Vivisimo trace, we are able to identify AOL users who might have shared IP addresses in the Vivisimo trace. We found that among the 110,881 queries received, there are only **2,949** queries submitted by **749** AOL clients. And only three of them are frequent users who submitted more than 70 queries. Therefore, we believe our results about the Vivisimo trace are not biased by these users.

IV. User lexicon analysis and its implications

In this section, we analyze the user query lexicons. We also propose possible ways to prefetch query results for each individual user, by recognizing their most frequently used terms.

A. Distribution of the User Lexicon Size

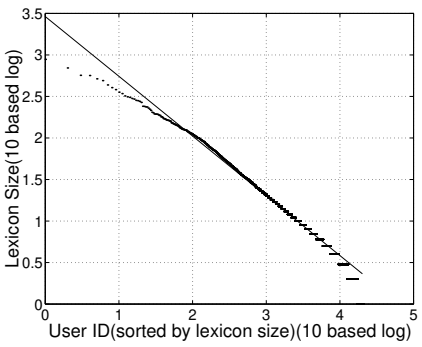
We noticed that the word frequency in the trace cannot be characterized by a Zipf distribution, which was also noticed in [7]. The graph falls steeply at the beginning and has an unusually long tail.

Instead of looking at the overall lexicons used in the trace, we group all the words used by each user individually, and examine the user lexicon size distribution. Among the **249,541** words in the queries from the Vivisimo trace, there are **51,895** distinct words. In the Excite trace, there are **350,879** distinct words among the **5,095,189** words from the queries. We notice that some users in the Excite trace submitted a large number of queries. For example, one user submitted 130,220 queries during the 8 hours. These users are likely to be meta-search engines instead of normal users. Thus we ignore the 60 users who submitted more than 100 queries in the Excite trace and examine the remaining users on their lexicon sizes. Compared with the overall lexicon size, the user lexicon sizes are much smaller. The largest user lexicons in the two traces have only **885** words and **202** words, respectively. We also find that the user lexicon size does not follow a Zipf distribution. Fig. 9 plots the distributions of the user lexicon size. The graphs have heavy tails, meaning the majority of the users have small lexicons.

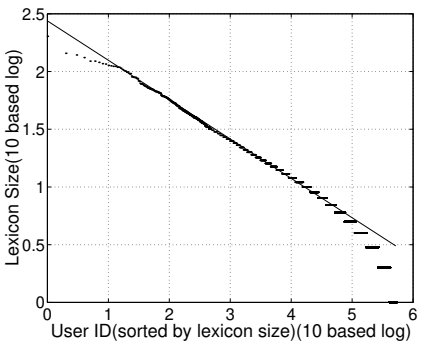
We also looked at the relationship between the number of queries submitted by each user and the corresponding lexicon size. For both traces, the more queries submitted by a user, the larger the user's lexicon. Fig. 10 shows the relationship between the number of queries submitted by a user and the corresponding user lexicon size based on the Vivisimo trace. The user lexicon sizes are in proportion to the number of queries submitted by the users. But there are a few exceptions, where the users submitted a lot of queries out of small lexicons. The Excite trace shows the same pattern and is thus not plotted here.

B. Analysis of Frequent Users and Their Lexicons

Though some users have large lexicons, they do not use all of the words uniformly. We are interested in how many words are most frequently used in the queries for each user. For the users in the Vivisimo trace who submitted only a few queries over the 35 days, calculating their frequently used words is not meaningful. Similarly, we do not look at the users in the Excite trace either because the trace lasts for too short a period. So we



(a) The Vivisimo Trace



(b) The Excite Trace

Fig. 9. Distribution of the user lexicon sizes with logarithmic scales on both axes. The X axis denotes the user IDs sorted by the lexicon sizes. When plotting the Excite trace, we remove those users who submitted more than 100 queries over the 8-hour period. Those users are very likely to be meta-search engines or robots instead of normal users.

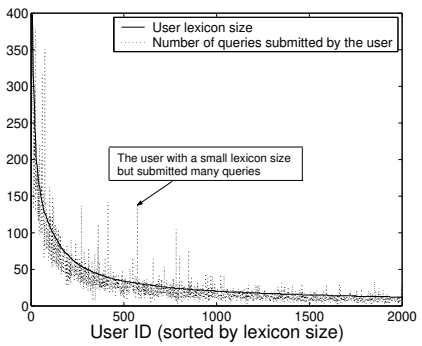


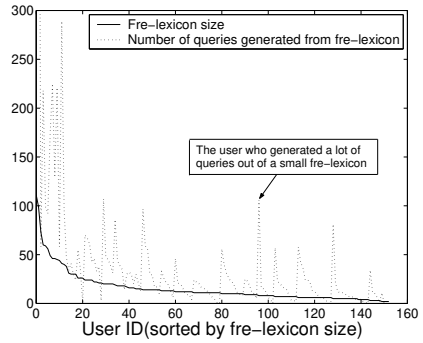
Fig. 10. User lexicon sizes and the number of the queries they submitted in the Vivisimo trace. The solid line plots the sorted user lexicon sizes. The dashed line plots the numbers of the queries submitted by the corresponding users. For scale reasons, this figure is a zoomed in part from a complete figure, but the pattern shown here is general for the complete figure as well.

only focus on those frequent users in the Vivisimo trace. We call a user a *fre-user* if the user submitted at least 70 queries. The fre-users thus submitted at least 2 queries each day on average. We also define a *fre-lexicon* for each fre-user. The fre-lexicons consist of the words that were used at least five times by the corresponding users.

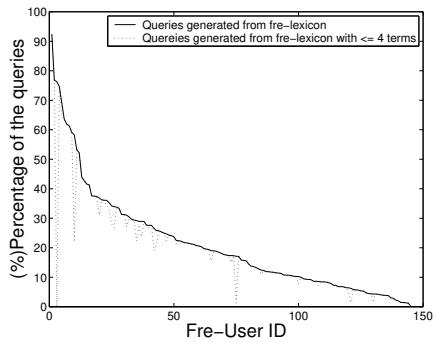
Fig. 11 shows the statistical summary about the fre-users in the Vivisimo trace. There are **157** fre-users, among whom, **153** fre-users have non-empty fre-lexicons. Although fre-users ac-

Number of fre-users	157
Number of queries submitted by fre-users	25,722
Number of fre-users with fre-lexicon size ≥ 1	153
Number of fre-users with fre-lexicon size ≤ 20	128

Fig. 11. Fre-users statistics in the Vivisimo trace. Fre-users are defined as those users who submitted at least 70 queries over the 35 days. Those users visited Vivisimo frequently and submitted at least 2 queries each day on average. Fre-lexicon is defined as the set of terms that were used by the corresponding user for at least five times.



(a) Queries and Fre-lexicons



(b) Percentage of the queries generated from fre-lexicons

Fig. 12. (a) The number of queries generated from fre-lexicons and the corresponding fre-lexicon sizes. The fre-user IDs are sorted by the lexicon sizes. (b) The percentage of the queries generated by the fre-lexicon for each fre-user. The fre-user IDs are sorted by the percentages.

count for less than **1%** of the users, they submitted **25,722** queries in total, which account for **23.20%**, a significant portion over the total number of queries. Since these users are non-trivial, caching and improving query results based on their individual requirements looks promising.

We also observe from the figure that most of the fre-users had small fre-lexicons. Thus we are interested in how many queries were generated purely by the words from fre-lexicons. If the users tend to re-use a small number of words very often to form queries, then we can predict queries and prefetch query results by simply enumerating all the word combinations. Fig. 12 (a) plots the number of queries generated from fre-lexicons. There are a small number of fre-users with relatively larger fre-lexicons, from which they submitted a lot of queries. So prefetching based on fre-lexicons for these users will help reduce the number of queries to be submitted dramatically. But we need a large cache size to store all the possible word com-

binations due to the relatively large fre-lexicon size. There are also quite a few users who generated a lot of queries from small fre-lexicons. For example, the user specified in the figure generated **106** queries from an **8** word fre-lexicon. For these users, prefetching according to fre-lexicons would be most effective.

Since the majority of queries have fewer than five terms, it is interesting to see how many queries with fewer than five terms were generated from fre-lexicons. Fig. 12 (b) shows both the percentage of the queries from fre-lexicons and the percentage of the queries from fre-lexicons with fewer than five terms. The addition of the extra constraints imposed on the number of terms does not affect the results for most of the fre-users. Therefore, we could just enumerate the word combinations using no more than four terms, which would greatly reduce the number of queries to be prefetched.

V. Research Implications

In this section, we review the statistical results derived from both traces and discuss their implications on future research directions. We focus on three aspects: caching search engine results, prefetching search engine results, and improving query result rankings.

A. Caching Search Engine Results

With $30\% \sim 40\%$ of repeated queries, caching search engine results is a non-trivial problem. Cache placement is one of the key aspects of the effectiveness of caching. Query results can be cached on the servers, the proxies, and the clients. For optimal performance, we should make decisions based on the following aspects:

1. *Scalability.* A cache scheme should scale well with the increasing size and the density of the Internet.
2. *Performance improvement.* This includes the amount of reduction in server workload, user access latency, and network traffic.
3. *Hit rate and shareness.* The higher the hit rate, the more efficient the caching. We would also like cached query results to be shared among different users of common interests.
4. *Overhead.* The overheads include the system resources devoted for caching and the extra network traffic induced.
5. *Opportunity for other benefits.* By disseminating user requests among caches, what other benefits can we achieve with the existence of caching?

Fig. 13 compares the pros and cons of different cache placement schemes in the general case:

- Server caching has only limited system resource available, so it does not scale well with the increasing size of the Internet. In addition, though we can avoid redundant computation, we cannot reduce the number of requests received by servers. The reduction in user access latency is also very limited. However, server caching has small overhead. Moreover, it allows the maximum query shareness among different users and the hit rate would be high by caching popular queries.
- Proxy caching is effective to reduce both server workload and network traffic. In the case of caching query results, this assumes that the users nearby a proxy would share queries. However, one of the main disadvantages of proxy caching is the significant overhead of placing dedicated proxies among the Internet.
- User side caching has the best performance improvement in case of a cache hit because the redundant user requests will not be sent out at all. Compared with the limited system resources at servers, the sum of each individual user resource is almost infinite. So user side caching achieves the best scalability. Because the overhead of caching can be amortized to a large number of users, the overhead at each user side is small. More importantly, with user side caching, it is now possible to prefetch or improve query results based on individual user requirements. However, no shareness can be exploited with user side caching. Thus if queries are mostly shared instead of being repeated by the same users, we would have low hit rate in such case.

The above discussion indicates that the degree of shareness is important to decide where we should cache query results. In one extreme case where users never repeat their own queries, we reach the maximum degree of shareness. In such case, we should cache query results at servers or proxies because user side caching would result in zero hit rate. In another extreme case where users never share queries, caching query results at the user side would be most helpful.

Our trace analysis results show that, among all queries, repeated queries account for 32% to 42%, while repeated queries from the same users account for 16% to 22%. The significant portion of the queries repeated by the same users and the non-trivial difference between the above two percentages suggest both server/proxy caching and user side caching. For the queries repeated only by the same users, we can cache them at the user side for efficiency, while the rest of the repeated queries can be cached at either servers or proxies. Since we do not know the actual IP addresses of the users in both traces, we cannot further distinguish the queries that can be shared by nearby users and thus can be cached at proxies. But it is clear from the traces that the percentage of such queries would be between 16% and 42%. Therefore, we leave proxy caching as a future work for search engines themselves to determine whether and where to place such proxies.

Since multiple-word queries have less degree of shareness, we suggest caching multiple-word queries mainly at the user side. Therefore, by caching only popular single-word queries at servers/proxies, we can achieve larger hit rates and greatly reduce the required system space.

Another important question about caching is how long we should cache query results. Temporal query locality indicates that most of the queries are repeated within short time intervals. So in general, caching query results for short periods should work well. More detailed analysis in Section III-C shows that the shorter time interval, the more likely for a query to be repeated by the same users. Thus, for user side caching, caching query results for hours will be enough to cover the query locality existing on the same users. This also helps to remove or update stale query results in time. There also exist a non-trivial portion of queries repeated over relatively longer time intervals. These queries are mainly shared among different users. So if caching is to be done at servers or proxies, we can consider longer-term caching such as a couple of days.

Level of caching	Server caching	Proxy caching	User side caching
Scalability	worst	medium	best
Performance improvement	worst	medium	best
Hit rate and shareness	best	medium	worst
Overhead	small	large	small
Opportunity for other benefits	least	medium	most

Fig. 13. Comparisons between different cache placement schemes.

The temporal query locality and the content dynamics of query results directly impact the mechanisms for maintaining cache consistency. Usually, search engines update query results on the order of days or even weeks. Therefore, for user side caching, where caching query results for hours would cover most of the query locality on the same users, we can use the following two mechanisms to maintain cache consistency. First, we can associate each query in the cache with a *time-to-live(TTL)* field and set the TTL to a relative short interval. Second, we can use an *if-modified-since* header field in each user request, like the HTTP protocol. The query result freshness can be guaranteed with the second mechanism. But we have larger overhead at both user side and server side since the user side needs one more interaction with the server for each request.

For server side and proxy caching, we should use different mechanisms to maintain cache consistency. With server side caching, we can easily ensure cache consistency by removing or updating stale query results whenever new query results are computed. For proxy caching, since queries shared by different users tend to be repeated over longer time intervals, the simple TTL based approach would incur more overhead. The reason is that TTL is usually set to a relatively short interval to prevent caches from serving stale data, and thus data may be reloaded unnecessarily. Therefore, more advanced invalidation protocols should be used to update stale query results while keeping the overhead low. For example, we can use a proxy polling technique where proxies periodically check back with the server to determine if cached objects are still valid. We can also let servers notify proxies when query results are updated. Both mechanisms are already in use for web caching on the Internet.

B. Prefetching Search Engine Results

Our user lexicon analysis in Section IV suggests that prefetching query results based on user fre-lexicons is promising. Prefetching has always been an important method to reduce user access latency. In the case of prefetching search engine results, it can be performed at both user side or proxies.

Caching query results at the user side provides useful information about user interest. For this reason, user side prefetching is natural and worth looking at. From the user lexicon analysis, we observe that the majority of the user lexicon sizes are small. Frequent users who submit many queries often use a small subset of words more often than other words. So a straightforward way of prefetching is to enumerate all the word combinations from fre-lexicons and prefetch the corresponding query results into a user level cache. The user query latency can be greatly reduced in case of a cache hit because no interaction with remote servers will be involved at all. Since queries are usually short, we can skip queries longer than four terms to reduce the prefetching overhead while achieving approximately the same performance improvement. For example, a 10-word fre-lexicon needs to prefetch 385 queries using the above naive algorithm, which will cost less than 20 minutes network download time using a 56Kbps modem and about 8M disk for storage. These overheads are trivial considering that a normal PC today is idle about 70% of the time on average and the normal disk size is tens of gigabytes. With user interests changing gradually, the corresponding fre-lexicons should also be updated to match user interests. New queries will be formed and new query results will need to be prefetched to achieve the best cache hit rates. In most cases, when user interests stay relatively stable, the majority of the words from a fre-lexicon will remain the same. There will be fewer new query results to fetch each time the prefetching is performed. Thus the prefetching overhead at the user side is small.

The above prefetching algorithm can potentially benefit servers as well. Since prefetched query results might not be used by the users, redundant query requests will be sent to servers. More network bandwidth and other system resources will be consumed compared with the case without prefetching. However, if we perform the prefetching algorithm regularly when the user machines are idle and the servers are not busy, most of these requests will be processed at non-peak times by servers. So we will not have any additional burden on servers at peak times. Instead, the server peak time workload can even be reduced since many of the queries have already been satisfied by user prefetching. Thus scarce server CPU and networking resources at peak times can be saved to serve more clients or apply to more advanced algorithms.

Prefetching can also be performed at proxies. In such cases, server's global knowledge about user query patterns can be utilized to decide what to prefetch and when to prefetch. Since proxies allow query shareness among different users, exploring how to achieve the maximum hit rate in such case would also be an interesting problem for future research.

C. Improving Query Result Rankings

Although today's search engines often return tens of thousands of results for a user query, only a few results are actually reviewed by users. Section II-C shows that each user on average reviews fewer than two pages of results. Thus, improving query result rankings based on individual user requirements is more important than ever. The personal nature of "relevance" requires incorporating user context to find desired information. Because centralized search engines provide services to millions of users, it is impractical to customize results for each user. Some spe-

cialized search engines do offer search results which are different than standard for some specialized user requirements. But none of them allows users to define their own requirements at will. With user side/proxy caching, it is now possible to re-rank the returned search engine results based on the unique interest of individual user. For example, a naive algorithm would be to increase the ranks of the Web pages visited by the user among the next query results. We can also explore other algorithms and integrate them with caching to customize search engine results for individual users.

VI. Conclusions

Caching is an important technique for reducing server workload and user access latency. In this paper, we investigated the issue of whether caching might work in the case of search engines, as it does in many other areas. We studied two real search engine traces and investigated the following three questions: (1) Where should we cache search engine results? At servers, proxies, or user side? (2) How long should we cache search results? (3) What are the other benefits of caching search engine results?

Our analysis of both the Vivisimo search engine trace and the Excite search engine trace indicate that: (1) Queries have significant locality. Query repetition frequency follows a Zipf distribution. The popular queries with high repetition frequencies are shared among different users and can be cached at servers or proxies. There are also about 16% to 22% of the queries repeated by the same users, which should be cached at the user side. Multiple-word queries have less degree of shareness and should be cached mainly at the user side. (2) The majority of the repeated queries are referenced again within short time intervals. There is also a significant portion of queries that are repeated within relatively longer time intervals. They are largely shared by different users. So if caching is to be done at the user side, short-term caching for hours will be enough to cover query temporal locality, while server/proxy caching should user longer periods, such as days. (3) Most users have small lexicons when submitting queries. Frequent users who submit many search requests tend to re-use a small subset of words to form queries. Thus with proxy or user side caching, prefetching based on user lexicon is promising. Proxy or user side caching also provide us with opportunities to improve query results based on individual user requirements, which is an important direction for future research.

VII. Acknowledgements

We gratefully acknowledge the assistance of Raul Valdes-Perez and the Vivisimo search engine team for providing us with a trace. Thanks also to the Excite team for making their trace available. Without the generous sharing of the trace data by Vivisimo and Excite, this work would not be possible. We also thank Jamie Callan for his helpful comments and suggestions.

References

[1] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich, "Performance of Web Proxy Caching in Heterogenous Environments," in *Proceedings of the IEEE infocomm'99, New York, NY*, March 1999.

[2] C. Maltzahn, K. Richardson, and D. Grunwald, "Performance Issues of Enterprise Level Web Proxies," in *Proceedings of the SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, June 1997.

[3] M.S. Raunak, P. Shenoy, P. Goyal, and K. Ramamritham, "Implications of Proxy Caching for Provisioning Networks and Servers," in *Proceedings of the ACM SIGMETRICS Conference, Santa Clara, CA*, June 2000.

[4] A. Rousskov and V. Soloviev, "On Performance of Caching Proxies," in *Proceedings of the ACM SIGMETRICS Conference, Madison, WI*, June 1998.

[5] P. Cao, L. Fan, and Q. Jacobson, "Web Prefetching Between Low-Bandwidth Clients and Proxies:Potential and Performance," in *Proceedings of the ACM SIGMETRICS Conference, Atlanta, GE*, May 1999.

[6] T. Kroeger, D. Long, and J. Mogul, "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[7] B.J. Jansen, A. Spink, J. Bateman, and T. Saracevic, "Real life information retrieval: a study of user queries on the Web," in *SIGIR Forum, Vol. 32. No. 1*, 1998, pp. 5–17.

[8] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz, "Analysis of a Very Large AltaVista Query log," Tech. Rep. 1998-014, Digital System Research Center, October 1998.

[9] S. Adali, K.S. Candan, Y. Papakonstantinou, and V.S. Subrahmanian, "Query Caching and Optimization in Distributed Mediator Systems," in *H. V. Jagadish and Inderpal Singh Mumick, editors, Proc. of the 1996 ACM SIGMOD Conf. on Management of Data,*, 1996, pp. 137–148. ACM Press.

[10] M. Taylor, K. Stoffel, J. Saltz, and J. Hendler, "Using Distributed Query Result Caching to Evaluate Queries for Parallel Data Mining Algorithms," in *Proc. of the Int. Conf. on Parallel and Distributed Techniques and Applications*, 1998.

[11] Evangelos P. Markatos, "On Caching Search Engine Results," Tech. Rep. 241, Institute of Computer Science, Foundation for Research & Technology - Hellas (FORTH), 1999.

[12] Evangelos P. Markatos, "On Caching Search Engine Query Results," in *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, May 2000.

[13] "Vivisimo," http://www.vivisimo.com.

[14] "Excite," http://www.excite.com.