On Caching Search Engine Query Results

Evangelos P. Markatos
Institute of Computer Science (ICS)
Foundation for Research & Technology – Hellas (FORTH)
P.O.Box 1385
Heraklio, Crete, GR-711-10 GREECE
tel: +30 81 391 655, fax: +30 81 391 661
markatos@csi.forth.gr
In Proceedings of the 5th International Web Caching and Content Delivery Workshop, May 2000

Abstract

In this paper we explore the problem of *Caching of Search Engine Query Results* in order to reduce the computing and I/O requirements needed to support the functionality of a search engine of the World-Wide Web.

We study query traces from the EXCITE search engine and show that they have a significant amount of temporal locality: that is, a significant percentage of the queries have been submitted more than once by the same or a different user. Using trace-driven simulation we demonstrate that medium-size caches can hold the results of most of the frequently-submitted queries. Finally, we compare the effectiveness of static and dynamic caching and conclude that although dynamic caching can use large caches more effectively, static caching can perform better for (very) small caches.

1 Introduction

Search Engines have become a popular way of finding information on the World Wide Web. A large number of users reach web sites via a Search Engine. As the web becomes larger, individual sites become more difficult to find, and thus the percentage of users that reach web sites after querying a Search Engine will probably increase. Besides people, computer programs (usually called agents, robots, or spiders) frequently access Search Engines searching for information [20].

To reduce the load on busy web servers (including busy search engines) web server accelerators are frequently used [7]. These accelerators are high-performance computers placed in front of a workstation cluster which hosts the web servers. The accelerator presents a single IP address to web clients, effectively hiding the number of workstations that host the web server. If the requested page is found in the accelerator's cache, it is returned to the client. Otherwise, the request is forwarded to one of the workstations in the cluster [26].

The make sure that accelerators are not a performance bottleneck, they are usually hosted on specialized computers with highly optimized main memory caching and networking software that can sustain several thousand requests per second. Although accelerators have been shown to result in large cache hit rates for static data [14], little is known about their effectiveness on caching dynamic data. Given that the generation dynamic data is a processor-intensive operation on most web servers, caching of dynamic data at the accelerator may turn out to have significantly higher benefits than caching static data [5]. In this paper we explore the effectiveness of dynamic data caching in accelerators used in front of Search Engines.

Although it improves performance, caching query results for a long time may affect the "freshness" of the data returned. However, caching query results even for several days may not noticeably contribute to staleness of data: the data (i.e. URLs) returned by Search Engines are already several weeks old; caching them for a few more hours (or even days) does not affect their freshness. Even in the rare cases where freshness of data is of crucial importance, accelerators provide mechanisms that remove data from their cache [14], making sure that they never deliver stale data to interested clients.

In this paper we show that it is possible to cache Search Engine results using moderate amounts of memory and small computational overhead. The contributions of the paper are:

- We study the traces of a popular Search Engine (Excite) and show that there is a significant locality in the queries asked, that is, 20%-30% of the queries have been previously submitted by the same or a different user.
- Using trace-driven simulation we show that medium-sized accelerator caches are enough to hold the results of most of the repeatedly submitted queries.
- We compare caching of the most popular queries (static caching) with caching of the most recently accessed queries (dynamic caching) and show that static caching of query results is promising approach for small caches, while dynamic caching has significantly better performance for large caches.

The rest of the paper is organized as follows: Section 2 places our work in context and summarizes related work. Section 3 analyzes the query trace and shows that there exists a significant amount of locality in the queries requested. Based on trace-driven simulations section 3 presents the performance benefits of of query result caching. Finally, section 4 summarizes and concludes the paper.

2 Previous Work

Caching documents to reduce access latency is extensively used on the web. Most web browsers cache documents in the client's main memory and disk [1]. To improve cache hit rates by aggregating the requests of several users, caching proxies are widely used. Proxies employ large caches which they use to serve a stream of requests coming from a large number of clients. Since even large caches may eventually fill up, cache replacement policies have been the subject of intensive recent research. One of the first cache replacement policies considered was LRU (Least Recently Used) and its variations. LRU is based on the heuristic that the documents not used recently will probably not be requested in the near future. After studying the access patterns of several web clients, it was realized that caching small documents may result in better performance (especially in a memory-limited environment). Thus, LRU was extended with heuristics that favor caching of small documents, either by precluding caching of large documents [1, 18], or by eagerly removing large documents [1, 27, 35]. Given that clients experience different latency to different servers, cache replacement policies may also take network latency into account in order to avoid replacing documents that may take a lot of time to download [30, 37]. Some policies aggregate all the above factors (access recency, size, latency) into a "weight" or "value" and try to keep in the cache the documents with the largest values [4, 15]. To improve hit rates even further, cache proxies are frequently organized into a hierarchy that can achieve better performance [6, 17]. Other approaches identify popular documents which are then cached for a predefined period of time (usually a day or so) [29]. Finally, some caches employ intelligent *prefetching* methods to improve the hit rate even further [3, 8, 9, 13, 19, 25, 34].

Spink *et al.* have analyzed the transaction logs of queries posed by users of *EXCITE*, a major Internet Search Engine [11, 32]. They have focused on *how* do users search the web, and on *what* do they search on the web. They present a large number of findings. Among the most interesting ones are that users usually ask short queries and view no more than 2-3 pages of the query's results.

C. Silverstein *et al.* have studied a very large log of AltaVista Queries [31]. Among other results, they report that the average frequency of the queries in their trace was 3.97. That is, each query was submitted four times (on the average), which implies that caching Search Engine results may achieve a hit rate up to 75% (for large caches). This is particularly encouraging for our approach since it demonstrates that there is an overwhelming amount of locality in Search Engine queries.

Query result caching has been previously studied in order to evaluate new queries based on the results of previous queries with similar constraints, or in cases when the source database is not readily available [2, 33]. The focus of previous work in query result caching was to reduce the cost of query execution (communication and computation overhead) in a distributed database system by caching the results of "similar queries", while the focus of our work is to reduce server load by employing a server-side query result cache.

Although dynamic data used to comprise a small portion of all web accesses, recent results suggests that as many as 40% of all web accesses are to dynamic data [36]. Challenger et al. report that it is possible to cache dynamic data, esp. in very popular athletic sites, like those of the Olympic games [5, 10]. Carefully caching and invalidating data at the 1998 Olympic games web server allowed it to reach hit rates close to 100%. By transferring only the differences between the new version and the old version of a web document, Mogul et al. manage to reduce the number of data need to be transferred from servers to proxies even for dynamically generated data [22].

Cao et al. suggest a novel architecture where proxies will be allowed to cache dynamically generated documents, provided that each time a proxy accesses its local copy of the dynamic document it executes some code provided by the original server of the dynamic document [24]. This code may verify the validity of data, add some advertisements to data, or process the data in some appropriate way. In their subsequent work, Lao et al. propose to use active caching in order to allow proxy servers to cache a search engine's query results [16]. W Meira Jr. *et al.* propose the use of "cache plugins" on special cache servers that reconstruct dynamic objects based on static components [21]. These plugins are provided by the original servers of dynamically generated documents. We view our work as complementary to [16, 21], since we focus on caching at the web server (search engine) level, while [16, 21] focus on caching at the proxy level.

Summarizing, although there has been significant work on web caching, there has been no previous systematic study on the existence of locality in Search Engine queries, and of evaluating the performance of caching those queries' results. In this paper we show Search Engine's queries have a significant amount of locality and evaluate several caching methods that exploit the existing locality.

3 Experiments

3.1 The Experimental Environment

3.1.1 The Traces

We use query traces from the EXCITE search engine (`www.excite.com`) gathered on Sep 16th 1997. The traces contain about one million searches. Of them, 927,010 are keyword-based queries and the rest are requests for "similar" documents [1]. Each entry in the trace file is actually a request by a specific user for a particular page of the results of a keyword-based search (query). Thus, when a user requests the first page of results of a given keyword-based search this represents one trace record in the log file. When the same user requests the second page of results of the same keyword-based search, this represents another trace record in the log file, etc. Thus, from now on, when we say query, we actually mean "a particular page of the results of a given keyword-based search". Since the trace file did not contain the number of results returned by each query, queries that generate less than a page of results are treated as queries that generate one page of results (which is roughly 4 Kbytes large). In this sense our results are somewhat pessimistic: we overestimate the amount of cache needed to hold query results.

[1] EXCITE gives users the ability to find documents that are "similar" to a URL returned.

3.1.2 The Experimental Environment

To evaluate the effectiveness of caching, the query traces are fed into a trace-driven web cache simulator. [2] During its execution, the simulator gathers a variety of statistics, the most important being the cache hit ratio: the percentage of queries that found their results in the cache.

3.1.3 Replacement Algorithms

Accelerator caches have limited size and will need to replace some of their data when they fill up. We will study several cache replacement algorithms in order to evaluate which is the most appropriate for such web caches. The algorithms we study include:

- **LRU**: the algorithm replacement the least recently used document (query results) from the cache.
- **FBR**: the algorithm that takes into account both the recency and frequency of access to a page [28]. It can be summarized as follows: "(1) the blocks in the cache are totally ordered in the same fashion as a cache using LRU replacement; (2) the cache is divided into three parts, a *new* section containing the most-recently used blocks, an *old* section containing the least-recently used blocks, and a *middle* section between the new and the old sections; (3) reference counts are only incremented for blocks that are not in the new section; and (4) on a miss, the block with the smallest reference count in the old section is replaced (with the least-recently-used such block selected if there is more than one).
- **LRU/2**: the algorithm replaces the page whose penultimate (second-to-last) access is least recent among all penultimate accesses [23]. [3]
- **SLRU**: the algorithm combines both the recency and frequency of access when making a replacement decision. "An SLRU cache is divided into two segments, a probationary segment and a protected segment. Lines in each segment are ordered from the most to the least recently accessed. Data from misses is added to the cache at the most recently accessed end of the probationary segment. Hits are removed from wherever they currently reside and added to the most recently accessed end of the protected segment. Lines in the protected segment have thus been accessed at least twice. The protected segment is finite, so migration of a line from the probationary segment to the protected segment may force the migration of the LRU line in the protected segment to the most recently used (MRU) end of the probationary segment, giving this line another chance to be accessed before being replaced. The size limit on the protected segment is an SLRU parameter that varies according to the I/O workload patterns. Whenever data must be discarded from the cache, lines are obtained from the LRU end of the probationary segment" [12].

3.2 Locality of Reference

In our first experiment we set out to find if there exists any locality of reference among the queries submitted, that is, if queries are submitted more than once. Figure 1 plots the number of accesses for the 1,000 most popular queries. Our results suggest that some queries are very popular: the most

[2] The simulator used was the one developed by Pei Cao at the University of Wisconsin [4]. We used most of its existing functionality and added a few more cache replacement policies.

[3] The above description of LRU/2 is oversimplified. If LRU/2 worked as stated it would soon degenerate to MRU, and would replace the block that has just brought in because eventually all blocks in the cache would have a positive penultimate access time, while newly accessed blocks would have a $-\infty$ penultimate access time, leading to their replacement. To avoid such degenerate behavior, O'Neil *et al.* suggest that blocks once brought into the memory buffer should stay there for a number of seconds. In our implementation of LRU/2, we keep all blocks in the cache according to their LRU access time and replace according to LRU/2 only from the last two thirds of the buffer. Thus, new blocks are given the chance to grow old and have a positive penultimate access time.

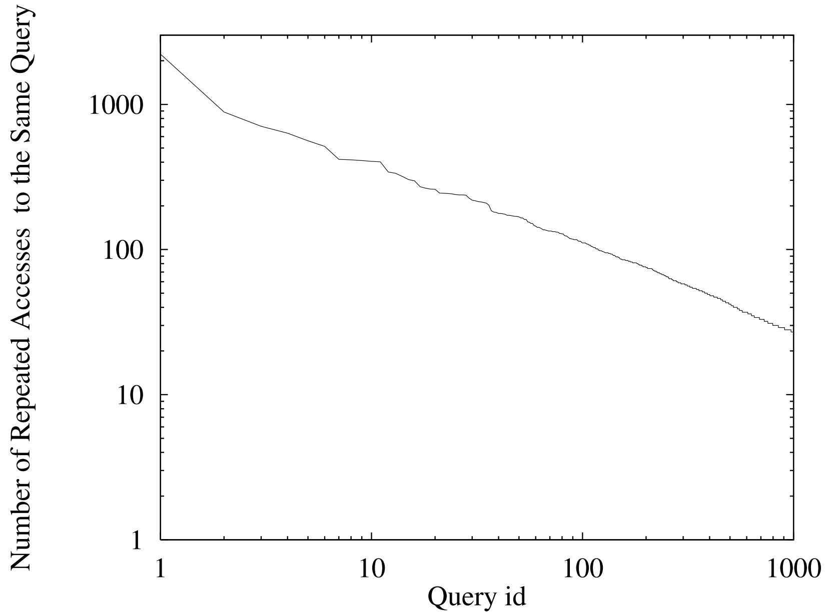

Figure 1: **Number of accesses for the 1,000 most popular queries.** The graph displays the number of requests for the 1,000 most popular queries of the trace studied sorted in decreasing number of accesses. We see that the most popular query is submitted 2,219 times, while the 1,000th most popular query is submitted 27 times.

popular query is submitted 2,219 times. Moreover, there exists a large number of queries that are accessed several times and are excellent candidates for caching: even the 1,000th most popular query is submitted as many as 27 times. We measured that the 25 most popular queries (1.23% of the total) were requested 11,435 times. [4] Although the most popular query is submitted 2,219 times, the average query is submitted 1.33 times, and thus caching search engine query results may reach a hit rate up to 25%. It is important to mention that an independent study of the AltaVista trace suggests that each query was submitted 3.97 times on the average, and thus caching the AltaVista search engine results may reach a hit rate be up to 75% [31]. [5] Although quantitatively different, the AltaVista statistics are very encouraging since they suggest that caching may result in significantly higher hit rates than the ones we observe using the Excite trace.

Although Figure 1 suggests that several queries are being submitted repeatedly, it does not quantify the temporal locality of the queries submitted. That is, is the same query repeatedly submitted within a small time interval? To quantify the temporal locality of the queries submitted we measured the time between successive submissions of the same query. The time was measured in queries intervened between the two successive submissions. The results were rounded to the nearest hundred and plotted in Figure 2. We see that in 1,639 instances the time between successive submissions of the same query was less than 100. In other words, a cache size that can hold the results of only the most recent 100 queries is enough to result in 1,639 hits. We also see that in 14,075 instances the time between successive submissions of the same query was less than 1,000, and in 68,618 instances the time between successive submissions of the same query was less than 10,000 (other queries).

[4] Our results agree with those from the AltaVista study [31] that suggest that the 25 most popular queries of the AltaVista trace amounted for the 1.5% of the total number of queries requested.

[5] This difference is probably because the AltaVista trace is about 1,000 times larger than our trace.

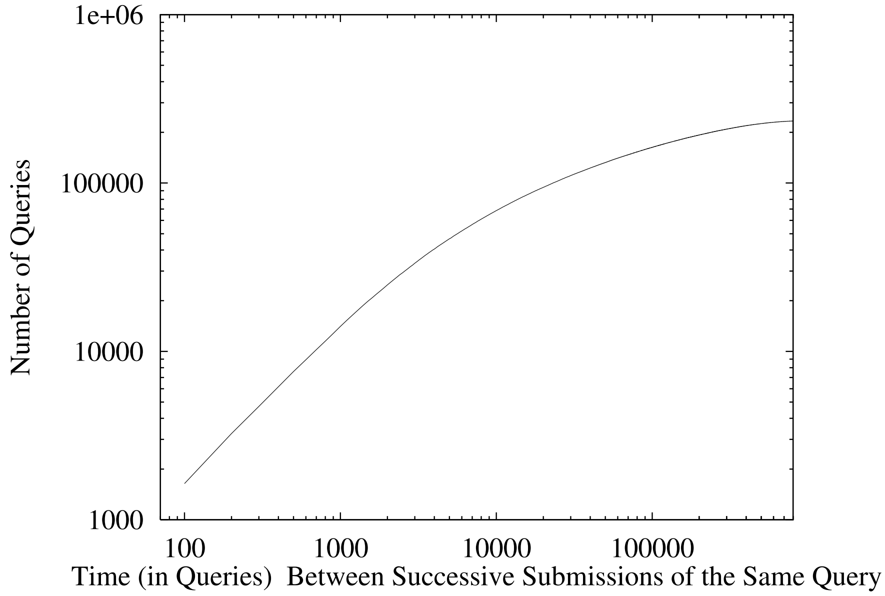

Figure 2: **Distances between submissions of the same query.** The graph displays the cummulative distribution of times between submissions of the same query by some user. We see that there is a large number of queries (1,639) whose time between two successive submissions is less than 100 (other) queries.

3.3 Caching Query Results

3.3.1 The Benefits of Caching Query Results

In our first caching experiment we evaluate the performance of caching query results, by comparing the effectiveness of several cache replacement algorithms: SLRU, LRU, FBR, and LRU/2.

Figure 3 plots the hit rate as a function of cache size. We see that the performance (hit rate) of all algorithms increases with cache size. For caches larger than 1 Gbyte (not shown here) all algorithms perform very close to the best [6]. For very small caches (<100 Mbytes) FBR and SLRU have the best performance followed by LRU/2 and LRU. For mid-size caches, LRU/2, SLRU and FBR have similar performance followed by LRU. For large caches (> 1 Gbyte) all algorithms have similar performance.

Figure 4 presents the hit rates achieved by FBR, SLRU, and LRU/2 alone in logscale in order to focus on their differences. We see that for very small cache sizes (<100 Mbytes) FBR performs better than LRU/2. For large caches LRU/2 performs a little better than FBR. However, in all cases, SLRU performs very close to the best of FBR and LRU/2.

Figure 4 suggests that a small 100 Mbyte cache is able to achieve close to 15% hit rate, or 60% of the maximum hit rate achievable. Such a small cache may easily fit in the main memory of a web server accelerator. However, to get the rest 40% we may need to use caches that are 1-2 Gbytes large, which should probably be kept in secondary memory. However, it may still be faster to keep a query's results in secondary memory instead of re-evaluating the query, especially for queries that are composed of several keywords, and/or queries that need several accesses to secondary memory [7].

[6] The cache size that would be required to hold the results of *all* queries submitted is 2.5 Gbytes.

[7] In such cases it is not uncommon for a program to consume over a second of CPU time in order to generate a single dynamic page containing the query's results [5].

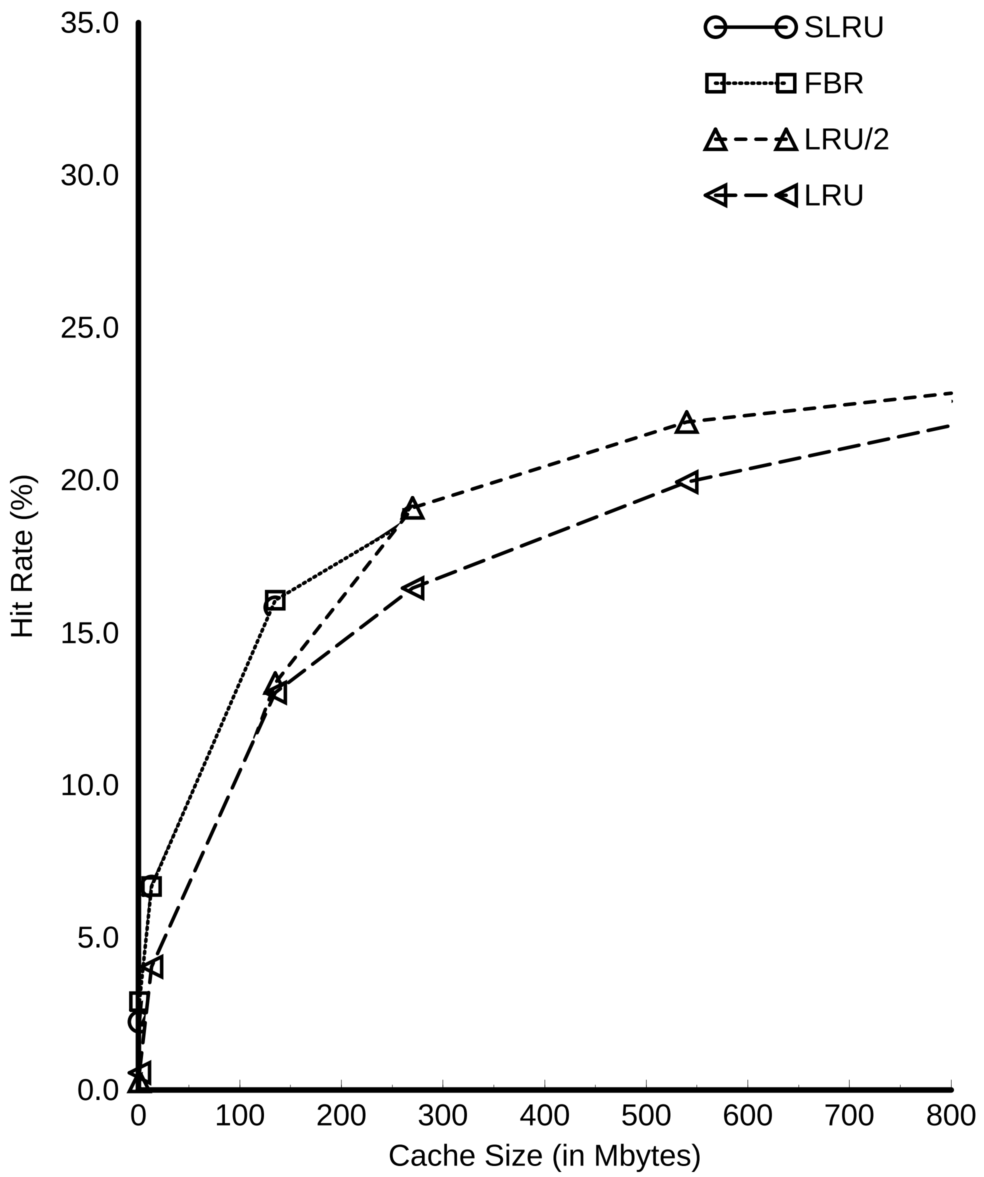

Figure 3: **Hit Rate as a function of the Cache Size**

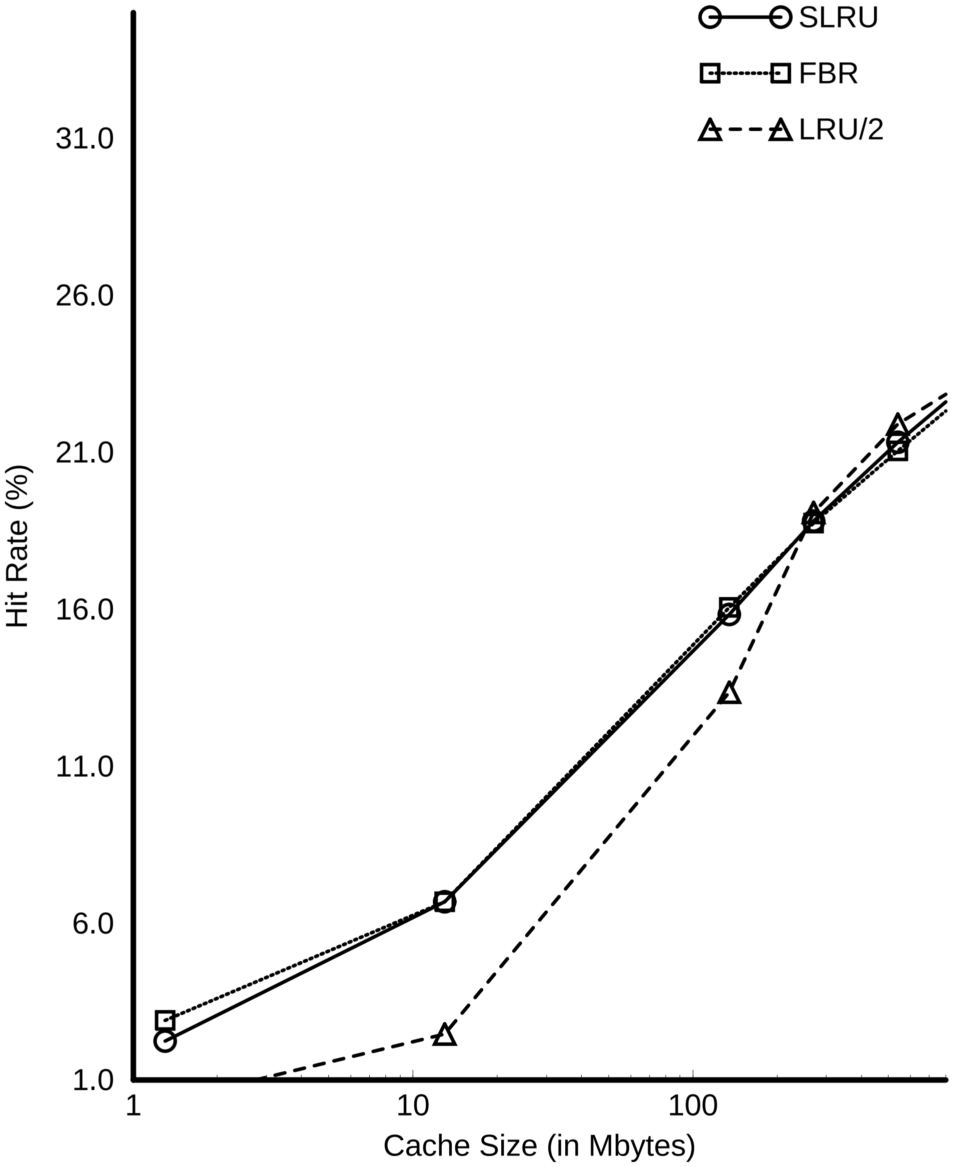

Figure 4: **Hit Rate of FBR, SLRU, and LRU/2 as a function of the Cache Size**

3.3.2 The Limits of Caching

Figure 3 suggests that although the hit rate increases with cache size, it reaches a maximum at 25%. This relatively low hit rate is, we believe, due to the short duration of the traces we had available. A study of a very large Alta-Vista trace suggests that the maximum hit rate is close to 75% [31]. We believe that the maximum achievable hit rate increases with the size of the trace. To verify our assumption, we repeated our simulations and measured the hit rate achieved for each individual interval of 50,000 queries. Figure 5 plots this hit rate as a function of time for cache size equal to 1.6 Gbytes (60% of the maximum cache size). We see that at the beginning the hit rate is low, but increases with time. Halfway through the simulation (at 450,000 submissions) it has reached a 25%. From that point on it continues to increase (albeit slowly) and stabilizes in the area of 29%. This experiment suggests that once the caches are warmed up, their effectiveness increases with the size of the trace and may reach close to 30%, or to put it simply, (roughly) one out of three queries submitted will be served from the cache.

3.4 Static Caching

Although the queries submitted to search engines cover a wide variety of interests, the most popular of the queries do not change frequently. Thus, it is theoretically possible to calculate what the popular queries are, and put their results in the cache. To keep the cache up-to-date, the popular queries may be recalculated periodically. We call this static approach of populating the cache **static caching**, to contrast with the rest of the policies that we have described which dynamically populate the cache with query results. Static caching may perform better than dynamic caching, since it populates the cache only with *popular* queries. On the other hand, its dynamic counterpart, caches *recent* queries, replacing in the process, some queries that may be less popular than the newly cached queries. However, static caching may also perform worse than dynamic caching, since, by its nature, it uses out-of-date information. That is, today's popular queries were not necessarily popular yesterday (e.g. breaking news about an earthquake will spring a significant number of new queries on the subject), and thus, they will not be cached by static caching until the popular queries are re-evaluated.

To evaluate the performance of static caching, we partitioned our 927,010 queries into two equal-sized sets. We used the first set to find the popular queries, and put them in cache. We drive the simulator with the second set of queries and measure the hit rate. During the trace-driven simulation the contents of the (statically-populated) cache do not change. Figure 6 plots the results of static caching compared to traditional dynamic caching algorithms like LRU and SLRU. We see that for small cache sizes (less than 50 Mbytes) static caching outperforms, although not by a significant amount, all dynamic caching approaches. Static caching can successfully populate small caches with most useful documents. Under dynamic caching, popular documents do not get the chance to stay in (the small) cache long enough to be reused at a later time. Figure 6 shows that for large cache sizes dynamic caching outperforms static caching, as expected, while for small cache sizes static caching is comparable (if not preferable) to dynamic caching.

4 Conclusions

In this paper we explore the performance advantages of caching dynamic data (and in particular Search Engine Query Results) on a web accelerator used in front of a web search engine. We use query traces from the EXCITE search engine to find the locality that may exist in the queries submitted and evaluate the performance of various cache replacement algorithms. Based on our trace-driven simulations we conclude:

- *There exists a significant amount of locality in the queries submitted to popular web search engines*. Our experiments suggest that one out of three of the queries submitted has been

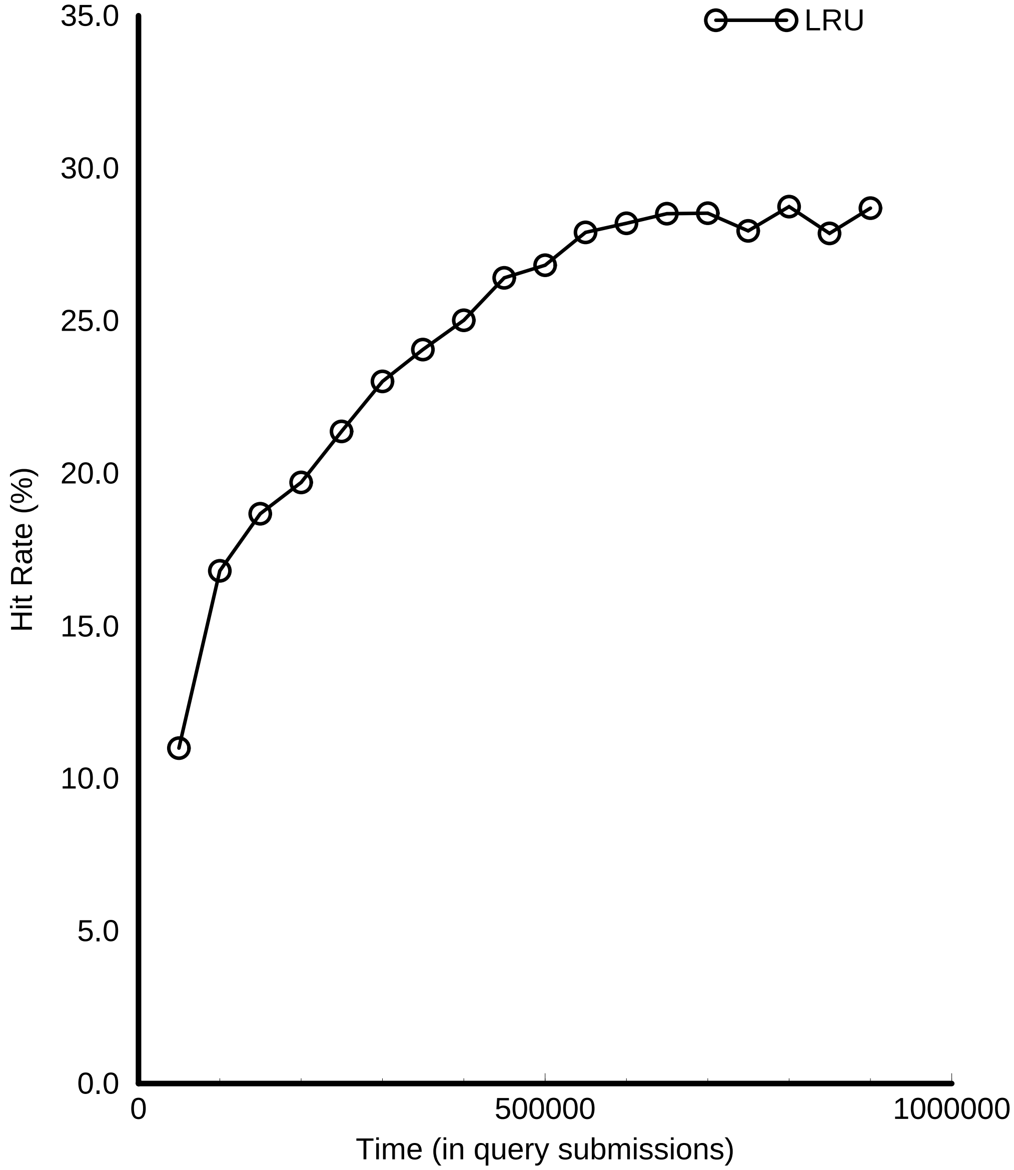

Figure 5: **Hit Rate as a function of Time**. We see that all policies start with a low hit rate. When they warm up the caches, hit rates may reach up to 30%.

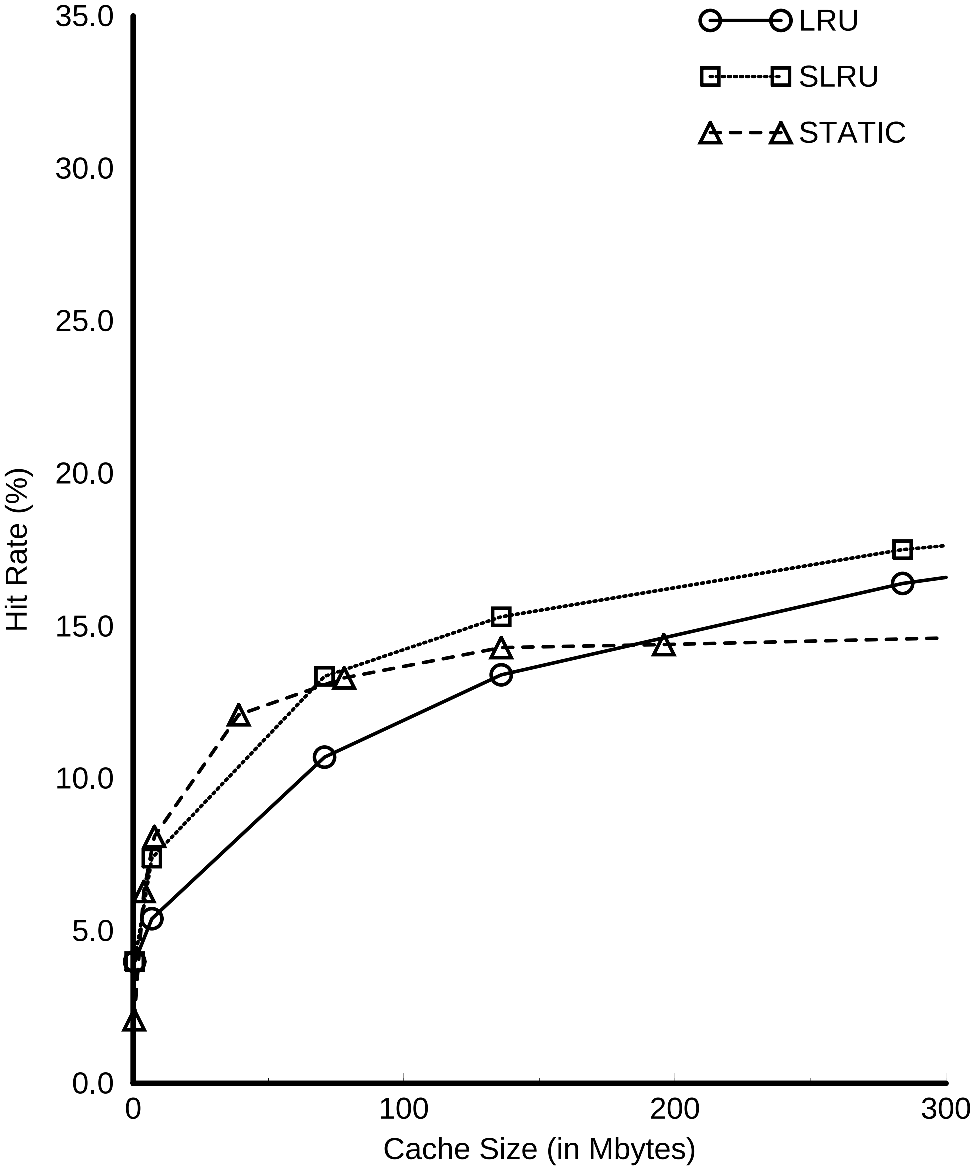

Figure 6: **Hit Rate comparison between static and dynamic caching.** Static caching is better than dynamic for small cache sizes: around 50 Mbytes.

previously submitted by the same or by another user. This amount of locality is already high and will probably increase in longer traces [31].

- *Medium-sized main memory accelerator caches absorb a significant percentage of the submitted queries*. Our experiments suggest that medium-sized caches can result in hit rates ranging around 20% (or even higher for warm caches). Effectively, the web server accelerator absorbs 20% of the incoming queries, reducing the load of the web servers by the same amount.

- *Effective Cache Replacement Policies should take into account both recency and frequency of access in their replacement decisions.* Our experimental results suggest that FBR, LRU/2, and SLRU always perform better than simple LRU which does not take frequency of access into account.

- *Static caching is effective for small cache sizes.* Static caching executes faster and has higher hit rate than dynamic caching algorithms for small cache sizes. Thus, it is a promising caching approach for very small caches, and should probably be used in conjunction with dynamic caching for larger caches.

Summarizing, we showed that although people have a wide variety of interests (from "aa alcoholics" to "zz top"), caching their interests is an effective and promising method to improve the performance of popular search engines.

Acknowledgments

This work was supported in part by PENED project 2041 2270/1-2-95, We deeply appreciate this financial support.

Some of the experiments were run at the Center for High-Performance Computing of the University of Crete (http://www.csd.uch.gr/hpcc/). Pei Cao provided the initial version of the simulator used in this paper. Amanda Spink and Jim Larsen pointed out the EXCITE traces and helped us with several questions. Dionisions Pnevmatikatos provided several useful comments. We thank all of them.

Vita

Evangelos Markatos is an assistant professor of Computer Science at the University of Crete, and Head of the Operating Systems and HPCN of the Institute of Computer Science, FORTH, Heraklion, Crete, Greece. He received his diploma in Computer Engineering from the University of Patras, Greece in 1988, and the M.S., and Ph.D degrees in Computer Science from the University of Rochester, NY in 1990, and 1993 respectively. His research interests include operating systems, computer architecture, networking, parallel/distributed systems, resource discovery, and the Web. He can be reached at markatos@ics.forth.gr.

References

[1] M. Abrams, C.R.Standridge, G. Abdulla, S. Williams, and E.A. Fox. Caching Proxies: Limitations and Potentials. In *Proceedings of the Fourth International WWW Conference*, 1995.

[2] 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*, pages 137–148. ACM Press, 1996.

[3] Azer Bestavros. Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time for Distributed Information Systems. In *Proceedings of ICDE'96: The 1996 International Conference on Data Engineering*, March 1996.

[4] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proc. of the first USENIX Symposium on Internet Technologies and Systems*, pages 193–206, 1997.

[5] J. Challenger, A. Iyengar, and P. Dantzig. A Scalable System for Consistently Caching Dynamic Data. In *INFOCOM 99*, 1999.

[6] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A Hierarchical Internet Object Cache. In *Proc. of the 1996 Usenix Technical Conference*, 1996.

[7] D. Dias, W. Kish, R. Muhkerjee, and R. Tewari. A scalable and highly available web server. In *COMPCON 96*, 1996.

[8] D. Duchamp. Prefetching Hyperlinks. In *Proc. of the second USENIX Symposium on Internet Technologies and Systems*, October 1999.

[9] J. Gwertzman and M. Seltzer. The Case for Geographical Push Caching. In *Proceedings of the 1995 Workshop on Hot Operating Systems*, 1995.

[10] A. Iyengar and J. Challenger. Improving Web Server Performance by Caching Dynamic Data. In *Proc. of the first USENIX Symposium on Internet Technologies and Systems*, 1997.

[11] B.J. Jansen, A. Spink, J. Bateman, and T. Saracevic. Searchers, the subjects they search, and sufficiency: A study of a large sample of EXCITE searches. In *Proceedings of Webnet 98*, 1998.

[12] R. Karedla, J. Love, and B. Wherry. Caching Strategies to Improve Disk System Performance. *Computer*, 1994.

[13] T.M. Kroeger, D.D.E. Long, and J.C. Mogul. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In *Proc. of the first USENIX Symposium on Internet Technologies and Systems*, pages 13–22, 1997.

[14] E. Levy-Abegnoli, A. Iyengar, and J. Song. Dias: Design and performance of a Web server accelerator. In *INFOCOM 99*, 1999.

[15] P. Lorenzetti, L. Rizzo, and L. Vicisano. Replacement Policies for a Proxy Cache, 1998. http://www.iet.unipi.it/˜ luigi/research.html.

[16] Q. Luo, R. Krishnamurthy, Y. Li, P. Cao, and J. F. Naughton. Active Query Caching for Database Web Servers, 1999.

[17] Radhika Malpani, Jacob Lorch, and David Berge. Making World Wide Web Caching Servers cooperate. In *Proceedings of the Fourth International WWW Conference*, 1995.

[18] E.P. Markatos. Main Memory Caching of Web Documents. *Computer Networks and ISDN Systems*, 28(7-11):893–906, 1996.

[19] E.P. Markatos and C. Chronaki. A Top-10 Approach to Prefetching on the Web. In *Proceedings of the INET 98 Conference*, 1998.

[20] Evangelos P. Markatos, Christina Tziviskou, and Athanasios Papathanasiou. Effective Resource Discovery on the World Wide Web. In *Proceedings of Webnet 98*, pages 611–616, 1998.

[21] W. Meira Jr, M. Cesario, R. Fonseca, and N. Ziv. Integrating WWW Caches and Search Engines. In *Proceedings of the IEEE 1999 Global Telecommunications Internet Mini-Conference*, 1999.

[22] J.C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *Proc. of the ACM SIGCOMM 97*, 1997.

[23] E.J. O'Neil, P.E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering. In *Proc. of the 1993 ACM SIGMOD Conf. on Management of Data*, pages 297–306, 1993.

[24] J. Zhang P. Cao and K. Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pages 373–388, 1998.

[25] V.N. Padmanabhan and J. Mogul. Using Predictive Prefetching to Improve World Wide Web Latency. *SIGCOMM Computer Communication Review*, 26:22–36, 1996.

[26] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proc. of the 8-th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1998.

[27] J.E. Pitkow and M. Recker. A Simple, Yet Robust Caching Algorithm Based on Dynamic Access Patterns. In *Proceedings of the Second International WWW Conference*, 1994.

[28] John T. Robinson and Murthy V. Devarakonda. Data Cache Management Using Frequency-Based Replacement. In *Proc. of the 1990 ACM SIGMETRICS Conference*, pages 134–142, 1990.

[29] A. Rousskov, V. Soloviev, and I. Tatarinov. Static Caching for Proxy Servers. In *Proceedings of the NLANR Web Cache Workshop*, 1997.

[30] P. Scheuearmann, J. Shim, and R. Vingralek. A Case for Delay-Conscious Caching of Web Documents. In *6th International World Wide Web Conference*, 1997.

[31] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a Very Large AltaVista Query Log. Technical Report SRC Technical Note 1998-014, 1998.

[32] A. Spink, B.J. Jansen, and J. Bateman. Users' searching behavior on the EXCITE web search engine. In *Proceedings of Webnet 98*, 1998.

[33] 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.

[34] Joe Touch. Defining High Speed Protocols : Five Challenges and an Example That Survives the Challenges. *IEEE JSAC*, 13(5):828–835, June 1995.

[35] S. Williams, M. Abrams, C.R. Standbridge, G. Abdulla, and E.A. Fox. Removal Policies in Network Caches for World-Wide Web Documents. In *Proc. of the ACM SIGCOMM 96*, 1996.

[36] A. Wolman, G. Voelker, N Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy. Organization-Based Analysis of Web-Object Sharing and Caching. In *Proc. of the second USENIX Symposium on Internet Technologies and Systems*, 1999.

[37] Roland P. Wooster and Marc Abrams. Proxy Caching that Estimates Page Load Delays. In *6th International World Wide Web Conference*, 1997.