Boosting the Performance of Web Search Engines: Caching and Prefetching Query Results by Exploiting Historical Usage Data

TIZIANO FAGNI, RAFFAELE PEREGO and FABRIZIO SILVESTRI
ISTI "A. Faedo" - Consiglio Nazionale delle Ricerche (CNR)

and

SALVATORE ORLANDO
Dipartimento di Informatica - Università Ca' Foscari di Venezia

This paper discusses efficiency and effectiveness issues in caching the results of queries submitted to a Web Search Engine (WSE). We propose SDC, a new caching strategy aimed to efficiently exploit the temporal and spatial locality present in the stream of processed queries. SDC stores the results of the most frequently submitted queries in a *static*, *read-only* portion of the cache, while the queries that cannot be satisfied by the static portion compete for the remaining entries of the cache according to a given replacement policy. Moreover, we improved the hit-ratio of SDC by using a speculative prefetching strategy, which anticipates future requests by introducing a limited overhead over the backend WSE. We experimentally demonstrated the superiority of SDC over purely static and dynamic policies by measuring the hit-ratio achieved on three large query logs by varying the cache parameters and the replacement policy used for managing the dynamic part of the cache. Finally, we deployed and measured the throughput achieved by a concurrent version of our caching system. Our tests showed how the SDC cache can be efficiently exploited by several threads that concurrently serve the queries of different users.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*; *Performance evaluation (efficiency and effectiveness)*

General Terms: Design, Performance, Experimentation

Additional Key Words and Phrases: Caching, Multithreading, Web Search Engines

1. INTRODUCTION

Caching is an effective technique to make scalable a service that distributes data to a multitude of clients. As suggested in [Xie and O'Hallaron 2002; Markatos 2000;

Authors' address: T. Fagni, R. Perego, F. Silvestri, Istituto ISTI A. Faedo, Consiglio Nazionale delle Ricerche (CNR), via Moruzzi 1, I-56100, Pisa, Italy. S. Orlando, Dipartimento di Informatica, Università Ca' Foscari di Venezia, via Torino 155, I-30172 Mestre (VE), Italy

Saraiva et al. 2001; Lempel and Moran 2003; Silvestri 2004], caching query results is an effective way to enhance the performance of a Web Search Engine (WSE). This is motivated by the high locality present in the stream of queries submitted by users to a WSE, and by the relatively infrequent updates of WSE indexes that allow us to think of them as mostly read-only data. WSE query results caching, as Web page caching, can occur at several places, e.g. on the client side, on a proxy, or on the server side. Independently of its placement, the presence of a cache has the effect of enhancing the WSE responsiveness, since several queries can directly be solved by the cache itself, without actually querying the WSE. Moreover, cache hits save WSE resources used to compute the page of relevant results returned to the user. Consider that, in order to answer a query in presence of a cache miss, a WSE has to perform several relatively expensive operations such as accessing the disk-resident index to retrieve the inverted lists associated with query keywords, intersecting them, and ranking the results obtained on the basis of their relevances [Witten et al. 1999]. With respect to these costs, the cost of a look-up operation on an in-core cache (cache hit) is clearly negligible.

In this paper we will discuss in detail a server-side caching system. As noted by Xie and OHallaron [Xie and O'Hallaron 2002] and confirmed by our analysis of query logs, a lot of popular queries are shared by different users. This level of shareness thus justifies the choice of designing a server-side WSE caching system. One of the issues in designing a server-side cache is the limited resources usually available on the server, in particular the RAM memory used to store the cache entries. However, the architecture of a scalable, large-scale WSE is very complex and includes several interconnected machines that take care of the various sub-tasks involved in the processing of user queries [Orlando et al. 2001; Barroso et al. 2003]. Figure 1 shows the architecture of a typical large-scale WSE placed behind an http server. It is a distributed architecture composed of a farm of identical machines running multiple *WSE Core* modules, each of which is responsible for searching a partition of the whole (inverted file) index. When each sub-index is relative to a disjoint sub-collection of documents, we have a *Local* or *Document Partitioning* index organization, while when the whole index is horizontally split, so that different partitions refer to a subset of the distinct terms contained in all the documents, we have a *Global* or *Term Partitioning* index organization. In both cases, in front of these searcher machines we have an additional machine hosting the *Broker*, which has the task of scheduling the queries to the various searchers, and collecting the results returned back. The Broker then merges and orders the results received on the basis of their relevance, and produces a ranked vector of document identifiers (DocIDs), e.g. a vector composed by 10 DocIDs. These DocIDs are then used to get from the *Urls/Snippets Server* (i.e. the server storing the original Documents downloaded by the Web spider) the associated URLs and page snippets to include in the html page returned to the user through the http server [Barroso et al. 2003]. Note that multi-threading is exploited extensively by all these modules in order to process concurrently distinct queries. Within this architecture the RAM memory is a very precious resource for the machines that host the WSE Core, which perform well only if the mostly accessed sections of their huge indexes can be buffered into the main memory. Conversely, the RAM memory is a less critical resource for the

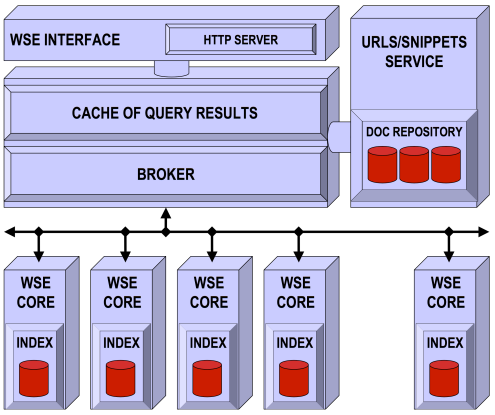


Fig. 1. Architecture of a large-scale, distributed WSE hosting a cache of query results.

machine that hosts the Broker. This machine can thus be considered as an ideal candidate to host a server-side cache. The performance improvement which may derive from the exploitation of query results caching at this level is remarkable. Queries resulting in cache-hits can be in fact promptly served thus enhancing WSE throughput, but also those resulting in cache-misses should benefit substantially, since miss penalty should be reduced due to the lower load on the WSE and the consequent lower contention for the I/O, network and computational resources.

In this paper, we are interested in studying the design and implementation of such a server-side cache of query results. Starting from the analysis of the content of three real large query logs, we propose a novel strategy to cache query results of a parallel and distributed WSE (called SDC- Static and Dynamic Cache). According to SDC, the results of the most frequently accessed queries are maintained in a fixed-size set of statically locked cache entries. This *Static Set* is rebuilt at fixed time intervals using statistical data coming from WSE usage data. When a query cannot be satisfied by the Static Set, it competes for the use of a *Dynamic Set* of cache entries. The management of the *Dynamic Set* adopts a fully-associative mapping of queries to cache entries, and can exploit, in principle, any replacement policy. We experimentally evaluated SDC by measuring the hit-ratio and the throughput achieved on real query logs by varying the size of the cache, the percentage of cache entries of the Static Set, and the replacement policy used for managing the Dynamic Set. In all the tests performed, SDC outperformed either purely static or dynamic caching policies.

Moreover, we showed that WSE query logs exhibit not only temporal locality, but also a limited spatial locality, due to requests for subsequent pages of results. To take advantage of spatial locality, our caching system adopts a *Speculative Prefetching Strategy* that, differently from other proposals [Lempel and Moran 2003], tries to maintain a low overhead on the underlying WSE Core. In fact, while server-side caching surely reduces the load over the core query service of a WSE and improves

its throughput, prefetching aims to increase the cache hit-ratio and thus the responsiveness (from the point of view of each single user) of the WSE, but may cause overhead on the same core query service.

Finally, the presence of a static portion of the cache simplifies the design of a high-throughput concurrent SDC caching system, since the read-only static entries can be accessed without synchronization by several threads serving the user queries. Differently from previous works,we accurately assessed the throughput of our concurrent caching system (i.e. the number of queries resolved per time unit) with respect to the number of concurrent threads exploited.

The rest of the paper is organized as follows. Related works are presented and discussed in Section 2. Section 3 analyzes the query logs used for the tests, while Section 4 discusses our novel caching policy. Section 5 shows the results of various simulations performed on the different query logs. Section 6 presents the architecture and the test results of our concurrent SDC caching system. Finally, Section 7 draws some conclusions and future works.

2. RELATED WORK

Several works studied the behavior of WSE users by analyzing usage data [Markatos 2001; Lempel and Moran 2003; Fagni et al. 2003; Saraiva et al. 2001; Xie and O'Hallaron 2002; Spink et al. 2002; Jansen et al. 1998; Spink et al. 1998; Hoelscher 1998; Silverstein et al. 1999; Lu and McKinley 1999; Feder et al. 2002; Spink et al. 2001; Chidlovskii et al. 1999; Dar et al. 1996; Jansen et al. 2000; Jansen and Pooch 2001; Lempel and Moran 2002].

Hoelscher [Hoelscher 1998] analyzed the query log of a the Fireball German search engine. The log contains about 16 millions of queries submitted on July 1996. The paper focuses on the analysis of features of the queries. The most interesting result is that a large part of the users (about 59%) just look at the first page of results, while the 79% of them look at no more than three pages of results.

Silverstein *et al.* [Silverstein et al. 1999] analyzed a large query log of the Altavista search engine containing about a billion queries submitted in a period of 42 days. The exhaustive analysis presented by the authors pointed out several interesting results. Tests conducted include the analysis of the query sessions for each user, and of the correlations among the terms of the queries. Similarly to previous works, their results show that the majority of the users (in this case about 85%) visit the first page of results only. They also showed that the 77% of the users' sessions end up just after the first query.

Markatos [Markatos 2000] analyzed a million queries contained in a log of the Excite WSE. The experiments showed the existence of temporal locality in the submission of the queries to the WSE. In particular, nearly a third of the submitted queries are successively resubmitted by either the same or other users.

Finally, Lempel and Moran [Lempel and Moran 2003] presented an analysis of a query log containing around 7.2 millions queries submitted to the Altavista search engine in the summer of 2001. They discovered that about the 63.5% of the views regard the first page of results only. On the other hand, the views accounted for the second page of results are about the 11.7% of the total. Another important statistical fact discovered by Lempel and Moran regards the topic popularity, i.e.

the number of times a particular query is requested. They showed that this popularity follows a *Zipf's Law* distribution. Practically speaking, most queries are requested only once, while a few of them are requested multiple times. Our log analyses evidenced the same Zipfian behavior.

Only a few of the works evaluating the features of WSE query logs also propose effective techniques to exploit the locality present in the stream of queries [Markatos 2000; Saraiva et al. 2001; Lempel and Moran 2003; Fagni et al. 2003].

Markatos was the first who proposed caching as a mean to reduce the response time of a WSE [Markatos 2000]. He implemented different state-of-the art caching policies and compared the (quite good) hit-ratio obtained on a log composed of queries submitted to Excite. He did not consider the possibility of exploiting a prefetching strategy in order to prepare the cache to answer possible requests for following pages.

The second paper describing a WSE caching system is the one by Saraiva *et al.* [Saraiva et al. 2001]. In their work the authors proposed a two-level caching system that aims to enhance the responsiveness of a hierarchically-structured search engine (very similar to the one sketched in Figure 1). The first-level cache stores query results and exploits an *LRU* policy. The second-level cache stores the posting list of the terms contained into the query string. For example, for the query "caching system" the second-level cache will separately store the posting lists of the terms "caching", and "system". The most interesting point is that the authors experimented their cache by measuring the overall throughput when either a two-level cache was, or was not adopted. Even though the second-level cache did not produce any increment in the throughput for low request rates, when this request rate increases the second-level can effectively help the disk in serving the requests, thus increasing the overall throughput. Also this work did not consider prefetching.

Lempel and Moran [Lempel and Moran 2003], besides analyzing query log features, also presented *PDC* (Probabilistic Driven Caching), a new WSE query results caching policy. The idea behind *PDC* is to associate a probability distribution with all the possible queries that can be submitted to a WSE. The distribution is built over statistics computed on the previously submitted queries. For all the queries that have not previously seen, the distribution function evaluates to zero. This probability distribution is used to compute a priority value that is exploited to order the entries of the cache: high probable queries are thus highly ranked, and have a low probability to be evicted from the cache. Indeed, a replacement policy based on this probability distribution is only used for queries regarding pages subsequent the first one. For queries regarding the first page of results, a simple *SLRU* policy is used. *PDC* also adopts prefetching to anticipate user requests. Moreover, *PDC* also exploits a model of user behavior. A user sessions starts with a query for the first page of results, and can proceed with one or more *follow-up* queries (i.e., queries requesting successive page of results). When no follow-up queries are received within τ seconds, the session is considered finished. This model is exploited in *PDC* by demoting the priorities of the entries of the cache referring to queries submitted more than τ seconds ago.

The results obtained on a query log of Altavista were very promising. With a cache of $256,000$ elements and prefetching 10 (i.e., 9 further pages of results are

Table I. Main characteristics of the query logs used.

Query log	queries	distinct queries	date
Excite	2,475,684	1,598,908	September 26^{th} 1997
Tiscali	3,278,211	1,538,934	April 2002
Altavista	6,175,648	2,657,410	Summer of 2001

always requested along with the first one), the authors measured a hit-ratio of about 53.5%. Unfortunately the *PDC* policy is very expensive from the computational point of view.

3. ANALYSIS OF THE QUERY LOGS

In order to evaluate the behavior of different caching strategies, we used various query logs. In particular we used *Tiscali*, a trace of queries submitted to the Tiscali WSE engine (www.janas.it) on April 2002, *Excite*, a publicly available trace of the queries submitted to the Excite WSE (www.excite.com) on September 26^{th} 1997, and *Altavista*, a query log containing queries submitted to Altavista (www.altavista.com) on the Summer of 2001. The *Excite* log is the same as the one used in [Markatos 2000], while the *Altavista* one was also used in [Lempel and Moran 2003]. Each record of a query log refers to a single query submitted to the WSE for requesting a *page* of results, where each page contains a fixed amount of URLs ordered according to a given rank. All query logs were preliminarily cleaned by removing the useless fields. At the end of this preprocessing phase, each entry of a query log had the form *(keywords, page_no)*, where *keywords* corresponds to the list of words searched for, and *page_no* determines which page of results is requested. We further normalized the query log entries by removing those referring to requests of more than 10 results-per-page.

Table I reports the main characteristics of the query logs used. While about the 46% of the total number of queries appearing in the relatively recent Tiscali and Altavista logs are distinct, in the Excite log this percentage increases up to 64%. Therefore, only looking at the numbers of distinct queries appearing in the three logs, we could deduce that locality in the Excite log, i.e. the oldest one, is lower than those present in the other two logs, since only the 36% (about 54% in the Tiscali and Altavista logs) of all its queries corresponds to re-submissions of previously submitted queries.

3.1 Temporal locality

The plots reported in Figure 2 assess the temporal locality present in the query logs using a log-log scale. In particular Figure 2.(a) plots the number of occurrences of the most popular queries within each log, where query identifiers have been assigned in decreasing order of frequency. Note that, in all the three logs, more then $10,000$ different queries are repeated more than 10 times. Since the number of occurrences of a given query is a measure that might depend on the total number of records contained in the logs, we also analyzed the distance between successive submissions of the same query. The rationale of this analysis is that if queries are repeatedly submitted within small time intervals, we can expect to be able to retrieve their results even from a cache of small size. Figure 2.(b) reports the results of this

analysis. For each query log we plotted the cumulative number of resubmissions of identical queries as a function of the *distance* between them. The distance is measured in number of queries. In particular, for each distance d, we plotted the cumulative number of repeated queries occurring at a distance less than or equal to d. The results were encouraging: for more than $350,000$ times the distance between successive submissions of the same query is less than 100 in the *Tiscali* log; this distance is slightly smaller in the *Altavista* log, but it is still less than 100 for more than $150,000$ times. In the *Excite* log we encountered a lower temporal locality. However, also in this log for more than $10,000$ times some queries are repeated at a distance smaller than 100. We think that the lower locality present in the *Excite* log is mainly due to its older age. It contains several long queries expressed in natural language like *"Where can I find information on Michael Jordan"*. In the query above the first six terms are meaningless, while the only informative keywords are the last two. Such kind of queries can be considered a symptom of the poor capacity of the users to interact with a WSE more than seven years ago. Although we tried to clean the *Excite* queries by removing *stopwords*, i.e. by eliminating meaningless terms like articles, adverbs and conjunctions, the query above still resulted syntactically different from the simpler query *"Michael Jordan"*, even if the same results may be considered relevant for both the queries.

3.2 Spatial locality

Although different in several assumptions and in some conclusions, the works surveyed in Section 2 show that WSE users in most cases submit short queries and visit only a few pages of results. Estimations reported in these works differ, in some cases, remarkably. Depending on the query log analyzed, percentages ranging from 28% to 85% of user searches only require the first page of results, so that we can expect that from 15% up to 72% of user searches retrieve two or more pages of results for the same query. This behavior causes, however, a limited spatial locality in the stream of queries processed by a WSE. Given a query requesting a single page of relevant results, with relatively high probability the WSE will receive a request for one of the following pages within a small time interval. To validate this consideration we measured the amount of spatial locality present in our query logs. Figure 3 reports the results of this analysis. In particular, Figure 3.(a) plots the percentage of queries in each log file as a function of the index of the requested page. As expected, most queries require the first page of results only. On the other hand, while there is a huge difference between the number of queries requesting the first and the second page, this difference becomes less significative when we consider the second and the third page, the third and the fourth, and so on. To highlight this behavior, Figure 3.(b) plots the probability of the occurrence of a request for the i-th page, given a previous request for the $(i-1)$-th page for the same topic. As it can be seen, this probability is low for $i = 2$, whereas it increases remarkably for higher values of i. This may reflect different usages of the WSE. When one submits a focused search, the relevant result is usually found in the first page. Otherwise, when a generic query is submitted, it is necessary to browse several pages in order to find the relevant information.

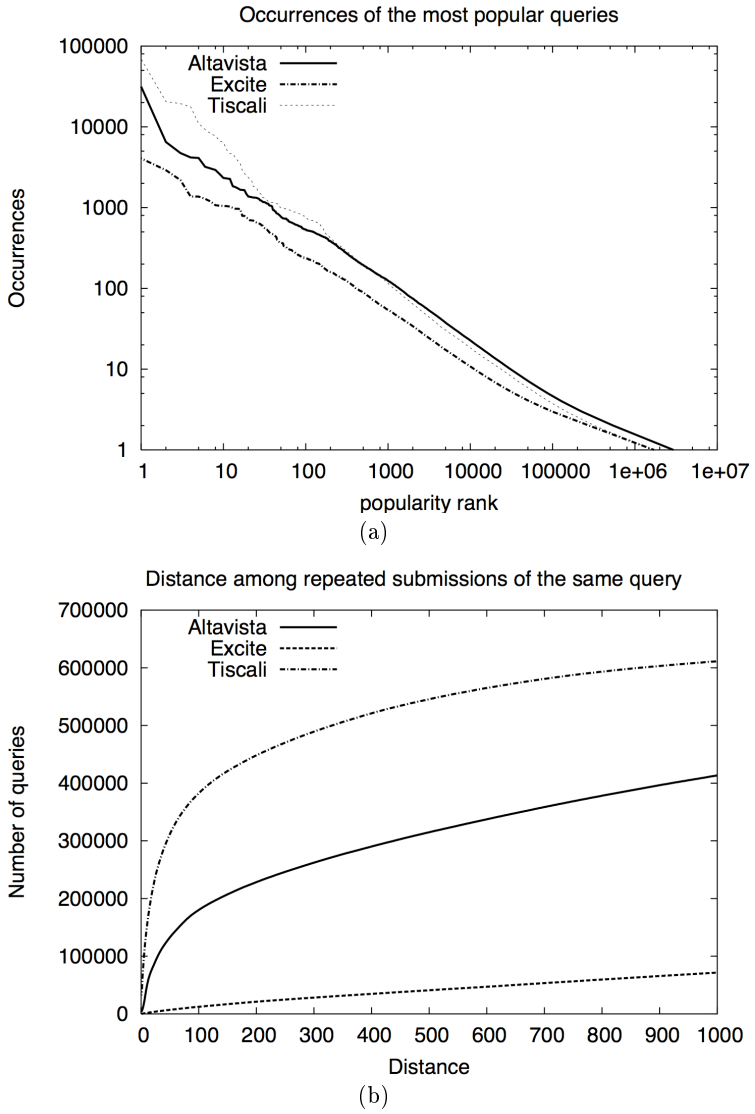
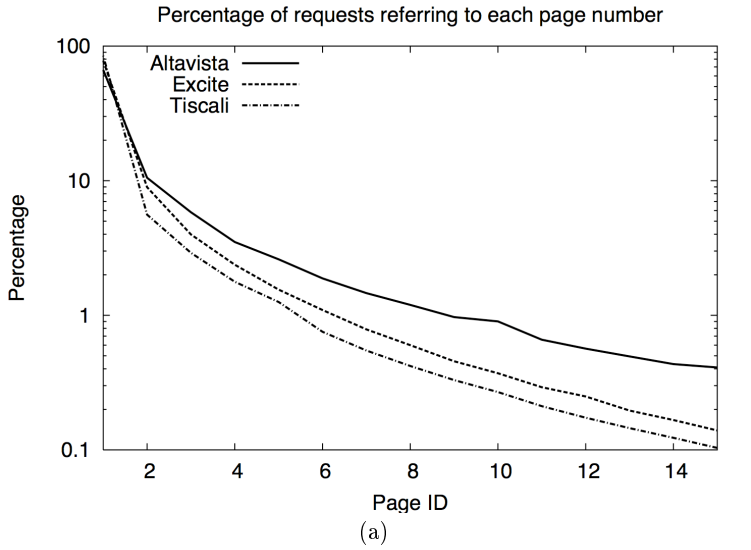


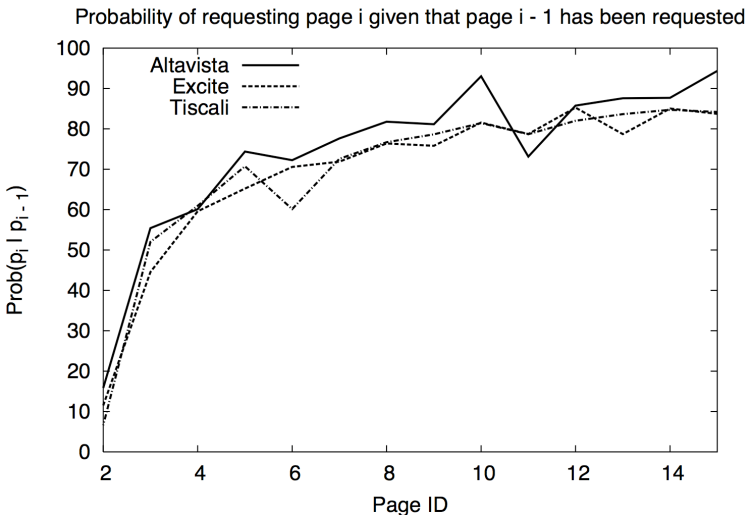
Fig. 2. (a) Number of submissions of the most popular queries contained in the three query logs (log-log scale). (b) Distances (in number of queries) between subsequent submissions of the same query.

3.3 Theoretical upper bounds on the cache hit-ratio

From the analysis of the three query logs we can devise the theoretical upper bounds on the maximum achievable hit-ratios when prefetching is not used. Thus we are supposing the availability of a cache of infinite size, so that only *compulsory* misses will be taken into account, i.e. cache misses corresponding to the first reference to each distinct query. Therefore the total number of compulsory misses is



(a)



(b)

Fig. 3. Spatial locality in the three query logs analyzed: (a) percentage of queries as a function of the index of the page requested; (b) probability of the occurrence of a request for the i-th page given a previous request for page $(i-1)$.

$$m = \frac{D}{Q} \tag{1}$$

where D is the number of distinct queries, whereas Q is the total number of queries. The value m is thus the minimum miss-ratio, while the maximum hit-ratio, H, is given by

Table II. Hit-ratio upper bounds for each query log as a function of the prefetching factor.

Prefetching factor	Query log			Prefetching factor	Query log		
	Excite	*Tiscali*	*Altavista*		*Excite*	*Tiscali*	*Altavista*
2	45.4283	56.652	60.8912	11	47.5763	57.0003	62.7765
3	47.2301	56.8608	61.7167	12	47.5768	57.0006	62.8283
4	47.4871	56.9751	62.078	13	47.5769	57.0007	62.8758
5	47.5442	56.9881	62.2799	14	47.5769	57.0008	62.9182
6	47.5634	56.9936	62.4008	15	47.5771	57.0009	62.9588
7	47.5685	56.9963	62.4886	16	47.578	57.0009	62.9968
8	47.5695	56.9979	62.5664	17	47.578	57.001	63.0329
9	47.5701	56.999	62.6374	18	47.5782	57.001	63.0725
10	47.5705	56.9997	62.716	19	47.5783	57.001	63.1124
				20	47.5783	57.001	63.1657

$$H = 1 - m = 1 - \frac{D}{Q} \tag{2}$$

If we apply Equation 2 to the data reported in Table I we obtain the following upper bounds:

—*Excite*: $0,3541$ (35.41%)
—*Tiscali*: $0,5305$ (53.05%)
—*Altavista*: $0,5697$ (56.97%)

The analysis becomes a little more complicated when we consider prefetching [Lempel and Moran 2003]. It is worth recalling that a general user query has the form *(keywords, page_no)*, and that for this analysis we are interested in considering queries characterized by identical *keywords* and distinct *page_no*. In particular, if the prefetching factor is k, we have to distinguish among queries requesting pages in the first *block* of k pages ($1 \leq page\_no \leq k$), in the second *block* of k pages ($k+1 \leq page\_no \leq 2 \cdot k$), and so on. Therefore, for all the user queries characterized by identical *keywords* and asking for pages belonging to the $i-th$ block of pages, we can thus count a single compulsory miss, since this miss will cause all the k pages of the block to be uploaded in the cache due to the prefetching strategy. Note that this reduces the fraction of compulsory misses, i.e. the minimum miss-ratio m, while the maximum hit-ratio H is increased accordingly. Table II shows the results computed on the three query logs.

As it can be seen from the values reported in the table, large prefetching factors do not result in sensible increases to hit-ratio bounds. In particular, for prefetching factors greater than 4 the variations are in the second decimal digit. We can also note that the highest differences in the hit-ratios attainable with and without prefetching are measured on the *Excite* log (prefetching increases of more than 10% the hit-ratio bound). Once more, this fact seems related to the peculiarities of the *Excite* log which contains poorly expressed queries that require users to browse some pages of results in order to satisfy their needs.

4. THE SDC POLICY

SDC is a two-level caching system which makes use of two different sets of cache

entries. The first level contains the *Static Set*, a set of statically locked cache entries filled with the most frequent queries appeared in the past. The Static Set is periodically refreshed on the basis of the WSE usage data. The second level contains the *Dynamic Set*, a set of cache entries managed by a classical replacement policy. The behavior of SDC in the presence of a query q is very simple. First it looks for q in the Static Set, and then in the Dynamic Set. If q is present (cache hit), it returns the associated page of results back to the user, otherwise (cache miss) SDC asks the WSE for the requested page of results, which possibly replaces an entry of the Dynamic Set according to the replacement policy adopted.

The rationale of introducing a Static Set, where the contents of the cache entries are statically decided, relies on the observation that query popularity follows a *Zipf's law* [Lempel and Moran 2003], and that the most popular queries submitted to WSEs do not change very frequently. Note that the idea of using a statically locked cache is present also in [Markatos 2000] where a purely static caching policy for WSE results is proposed and compared with purely dynamic ones. On the other hand, several queries are popular only within relatively short time intervals, or may become suddenly popular due to, for example, un-forecasted events (e.g. the 11^{th} September 2001 attack). Since these queries cannot clearly profit from a static cache, we introduced the Dynamic Set. The advantages deriving from this novel two-level caching strategy are two-fold: the results of the most popular queries can be retrieved from the Static Set even if some of these queries might be not requested for relatively long time intervals, and, on the other hand, the Dynamic Set of the cache can adequately cover sudden interests of users. Furthermore, the presence of the Static Set has a positive impact on the throughput of the cache system and thus of the WSE, since the static entries of the cache can be safely accessed without synchronization by all the concurrent threads which serve the queries of the users.

First level - Static Set. The static cache has to be initialized off-line, i.e., with the results of the most frequent queries computed on the basis of a previously collected query log. These queries and corresponding results are statically mapped to cache entries of the Static Set using a completely associative mapping functions. The implementation of the first level of our caching system is thus very simple. It basically consists of a lookup data structure that allows to efficiently access a set of $f_{static} \cdot N$ entries, where N is the total number of entries of the whole cache, and f_{static} the factor of locked entries over the total. In our implementation, f_{static} is a parameter, given at start time, whose admissible values ranges between 0 (a fully dynamic cache) and 1 (a fully static cache).

Each time a query is received, SDC first tries to retrieve the corresponding results from the Static Set. On a cache hit, the requested page of results is promptly returned. On a cache miss, we also look for the query results in the Dynamic Set.

Second level - Dynamic Set. Also for the Dynamic Set we adopt a completely associative mapping method. Differently from the Static Set, it has to rely on a replacement policy for choosing which pages of query results should be evicted as a consequence of a cache miss when the Dynamic Set is completely full. Literature on caching proposes several replacement policies which, in order to maximize the hit-ratio, try to take the largest advantage from information about recency and

frequency of references. SDC surely simplifies the choice of the replacement policy to adopt. The presence of a static read-only cache, which permanently stores the most frequently referred pages, makes in fact recency the most important parameter to consider. Currently, our caching system supports the following replacement policies: *LRU*, *LRU/2* [O'Neil et al. 1993] which applies a *LRU* policy to the penultimate reference, *FBR* [Robinson and Devarakonda 1990], *SLRU* [Karedla et al. 1994], *2Q* [Johnson and Shasha 1994], and *PDC* [Lempel and Moran 2003]. All these policies consider both recency and frequency of accesses to cache blocks.

The choice of the replacement policy to be used is performed at start-up time, and clearly affects only the management of the $(1 - f_{static}) \cdot N$ dynamic entries of our caching system.

Hereinafter we will use the notation SDC-r$_s$ to indicate a configuration of SDC with replacement policy r, and $f_{static} = s$. For example, SDC-$LRU_{0.4}$ means that we are referring to SDC using *LRU* as the replacement policy for the Dynamic set of the cache, and $f_{static} = 0.4$.

5. EXPERIMENTS

All the experiments were conducted on a Linux PC equipped with a 2GHz Pentium Xeon processor and 1GB of RAM. Since SDC requires the blocks of the static section of the cache to be preventively filled, we partitioned each query log into two parts: a *training set*, which contains 2/3 of the queries of the log, and a *test set*, which contains the remaining queries used in the experiments. The N most frequent queries of the training set were then used to fill the cache blocks: the first $f_{static} \cdot N$ most frequent queries (and corresponding results) were used to fill the static portion of the cache, while the following $(1 - f_{static}) \cdot N$ queries to fill the dynamic one. Note that, according to the scheme above, before starting the tests not only the static blocks but also the dynamic ones are filled, and this holds even when a pure dynamic cache ($f_{static} = 0$) is adopted. In this way we always start the experiments from the same initial warm cache, and we can fairly compare different configurations of SDC.

5.1 SDC without prefetching

The plots on the top sides of Figures 4-6 report the cache hit-ratios obtained by SDC on the *Tiscali*, *Excite*, and *Altavista* query logs by varying the ratio f_{static} between the sizes of the static and dynamic sets. Each curve corresponds to tests conducted by adopting a different replacement policy for the dynamic portion of the cache. The value of f_{static} was varied between 0 (a fully dynamic cache) and 1 (a fully static cache), while the replacement policies exploited were *LRU*, *FBR* [Robinson and Devarakonda 1990], *SLRU* [Karedla et al. 1994], *2Q* [Johnson and Shasha 1994], and *PDC* [Lempel and Moran 2003]. The total size of the cache was fixed to $256,000$ blocks. Several considerations can be done looking at these plots. Firstly, we can note that the hit-ratios achieved are in some cases impressive, although the curves corresponding to different query logs have different peak values and shapes, thus indicating different amounts and kinds of locality in the query logs analyzed. Secondly, and more importantly, we see that SDC remarkably outperformed in all the tests either purely static ($f_{static} = 1$), or purely dynamic caching ($f_{static} = 0$) policies. The best value for f_{static} depends on the query log considered, but values

between 0.3 and 0.8 work quite well in all the cases. Finally, we can see that the different replacement policies for the Dynamic Set behave similarly with respect to the SDC hit-ratio. For example, on the *Altavista* log, all the curves but the *FBR* one resulted almost completely overlapped. This behavior seems to suggest that the most popular queries are satisfied by the Static Set, while the majority of the queries appearing a few times only do not benefit from the cache at all. Thus, the Dynamic Set is only exploited by those queries which we can define *Burst Queries*, i.e., which appear relatively frequently but just for a brief period of time. For these queries, the small variations in the hit-ratio figures seems to not justify the adoption of a complex replacement policy rather than a simple LRU one.

To measure the sensitivity of SDC with respect to the size of the cache, the plots on the bottom sides of Figures 4 - 6 show the hit-ratios achieved on our query logs as a function of the number of blocks of the cache for different values of f_{static}. As expected, when the size of the cache is increased, hit-ratios increase correspondingly. In the case of the *Altavista* log, the highest hit-ratio achieved with a cache of $50,000$ blocks is about 29%, and about 36% when the size of the cache is $256,000$. Note that actual memory requirements are however limited: a cache of $50,000$ blocks storing the results as complete and not compressed *Html* pages requires about 200MB of RAM.

5.2 SDC and prefecthing

The spatial locality present in a stream of queries submitted to a WSE can be exploited by anticipating the requests for the following pages of results. In other words, when the cache receives a query of the form *(keywords, page_no)*, and the corresponding page of results is not found in the cache, an expanded query requesting k consecutive pages starting from page *page_no*, where $k \geq 1$ is the *prefetching factor*, might be forward to the core query service WSE. To this end, the queries passed to the core query service of the WSE have generally expressed in the form *(keywords, page_no, k)*. Note that, according to this notation, $k = 1$ means that prefetching is not activated.

When the results of the expanded queries are returned to the caching system, the retrieved pages are stored into k distinct blocks of the cache, while only the first page is returned to the requesting user. In this way, if a query for a following page is received within a small time interval, its results can surely be found into the cache.

Figure 7 and 8 show the results of the tests conducted with SDC and prefetching. In particular, we tested SDC with either LRU or PDC as replacement policy for the Dynamic Set, with no prefetching (subscript of the curve label = 1), and with prefetching factor $k = 5, 10$.

As expected, prefetching allows the hit-ratios of the various caching policies to be considerably increased. For example, on the *Altavista* query log with a SDC-*LRU* cache of $256,000$ entries, we obtained a hit-ratio of 54.20 when 10 pages of results are prefetched ($k = 10$), versus a hit-ratio of 35.76 when no pages are prefetched. In this case the adoption of prefetching increased the hit-ratio of about 50%. Unfortunately the prefetching factor k cannot indefinitely grow. In fact, all the pages of results prefetched have to be inserted in the cache by likely evicting from the cache an equal number of entries according to the replacement policy

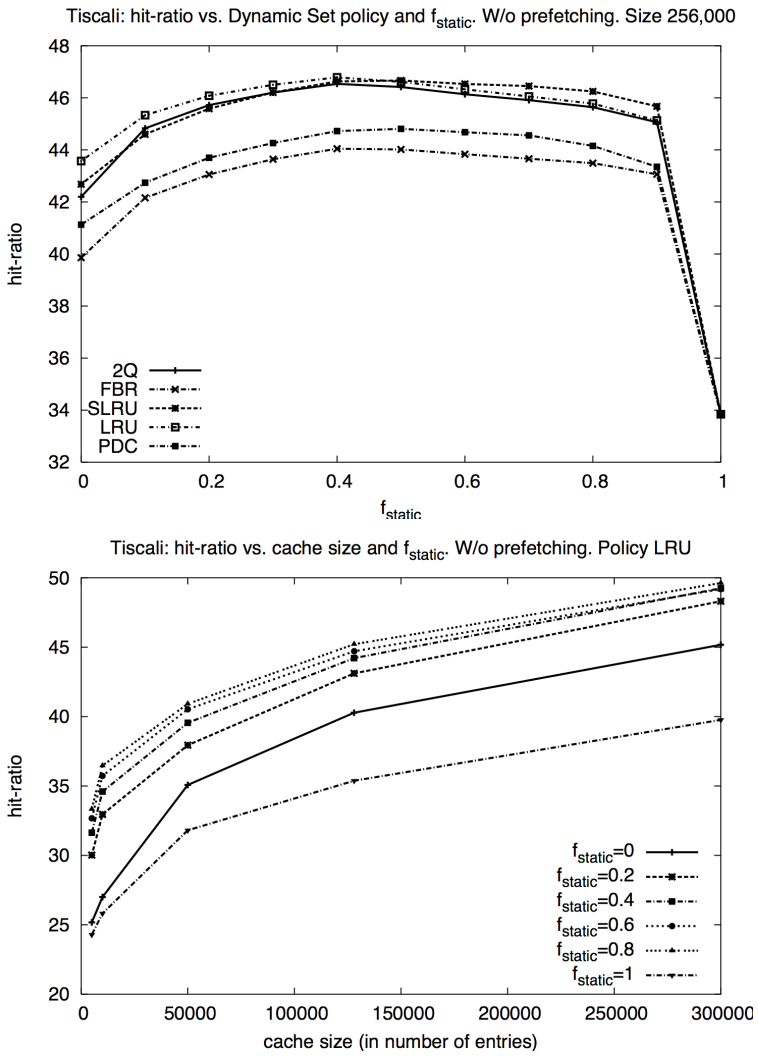


Fig. 4. Hit-ratios achieved on the *Tiscali* query log for different replacement policies and varying values of f_{static}, and for different sizes of the cache.

adopted. Obviously, the hit ratio increases only if the probability of accessing the prefetched pages is greater than the evicted ones. Particularly for caches of small size, prefetching might negatively affect the effectiveness of the cache replacement policy adopted. In this regard, Figure 9 plots the results of the test conducted with a SDC-*LRU* cache of 32,000 entries on the Excite query log. As it can be seen, with this small cache the hit-ratio begin to decrease when a prefetching factor greater than 3 is used.

An important issue to deal with, is to prefetch or not whenever a Static Set hit occurs. As the Static Set is statically filled with the most frequently occurring

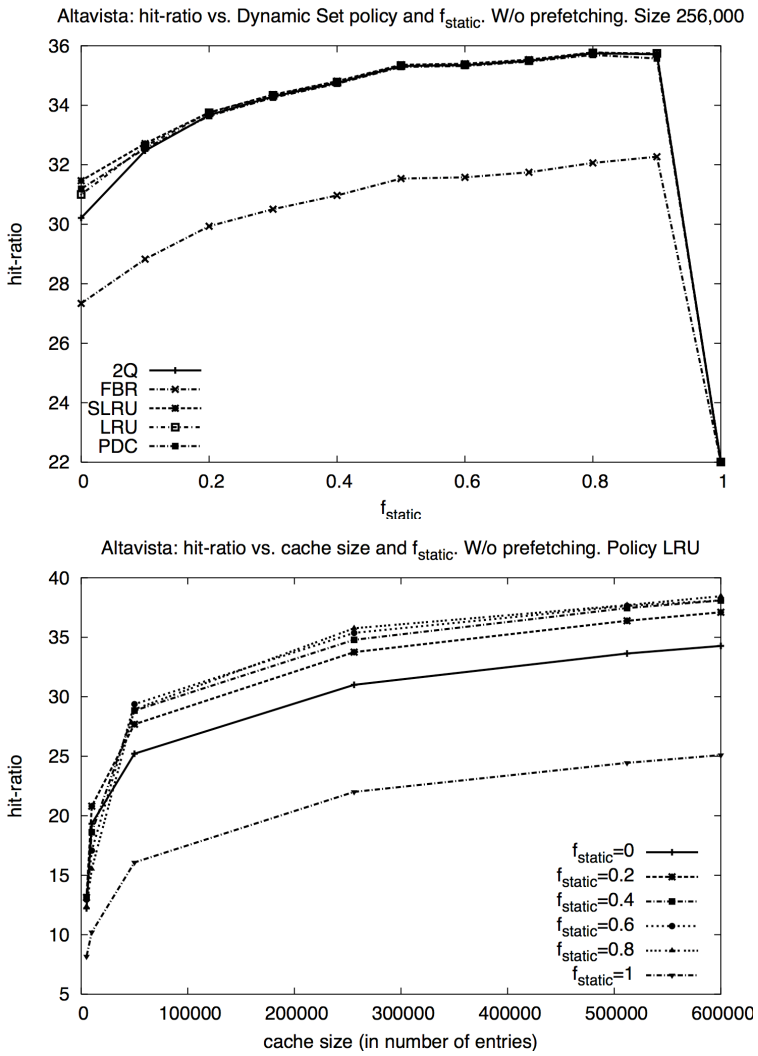


Fig. 5. Hit-ratios achieved on the *Altavista* query log for different replacement policies and varying values of f_{static}, and for different sizes of the cache.

query results it is likely that the successive results will not be present in the cache at that particular moment. Thus, to ensure the coherence of the cache we should check the Dynamic Set for the presence of successive results for a request contained within the Static Set. We choose not to perform this check since in the case of a multithreaded system this would result in a general throughput degradation due to the effects of software lockout.

Another possible drawback of using prefetching is that on the WSE side the load obviously increases. The cost of resolving a query is, in several WSE implementations, logarithmic with respect to the number of results retrieved [Witten et al.

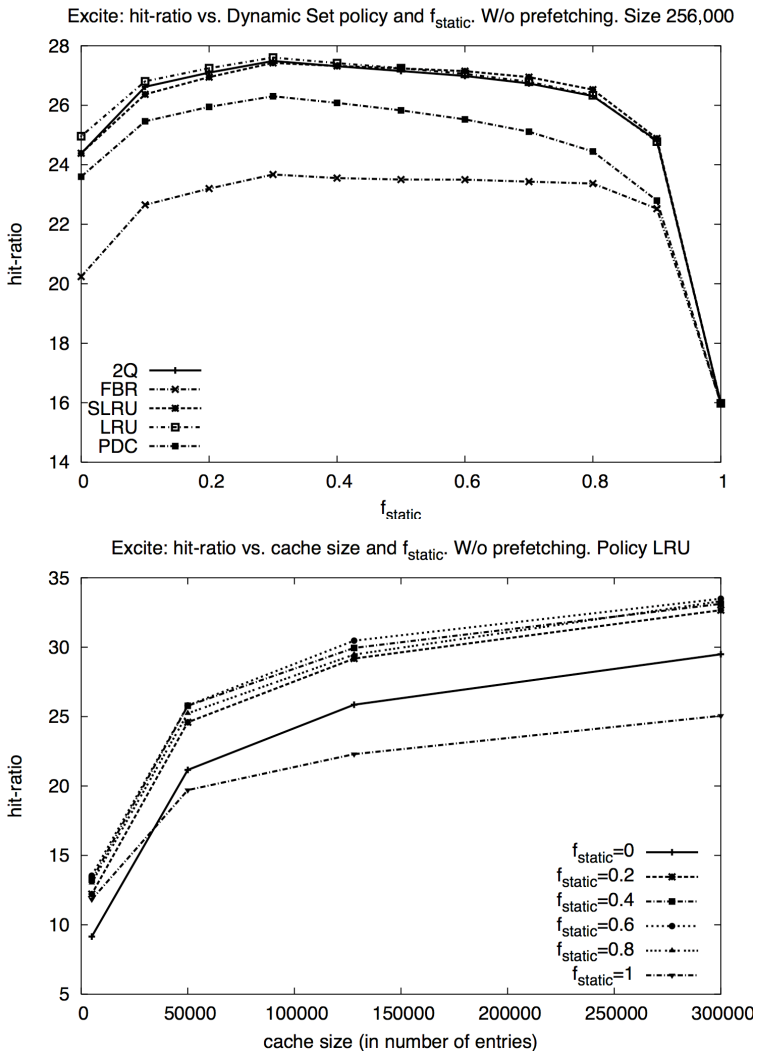


Fig. 6. Hit-ratios achieved on the *Excite* query log for different replacement policies and varying values of f_{static}, and for different sizes of the cache.

1999]. An aggressive prefetching strategy might thus degrade the throughput of the WSE core service. On the *Altavista* query log, we measured that with a SDC-$LRU_{0.9}$ cache of 256,000 entries and a prefetching factor 5, only 11.71% of the prefetched pages are actually referred by the following queries.

In order to maximize the benefits of prefetching, and, at the same time, reduce the additional load on the WSE, we can take advantage of the characterization of the spatial locality as presented in Section 3. Figure 3.(b) shows that, given a request for the i-th page of results, the probability of having a request for page $(i+1)$ in the future is about 0.1 for $i = 1$, but becomes approximately 0.5 or greater for $i > 1$.

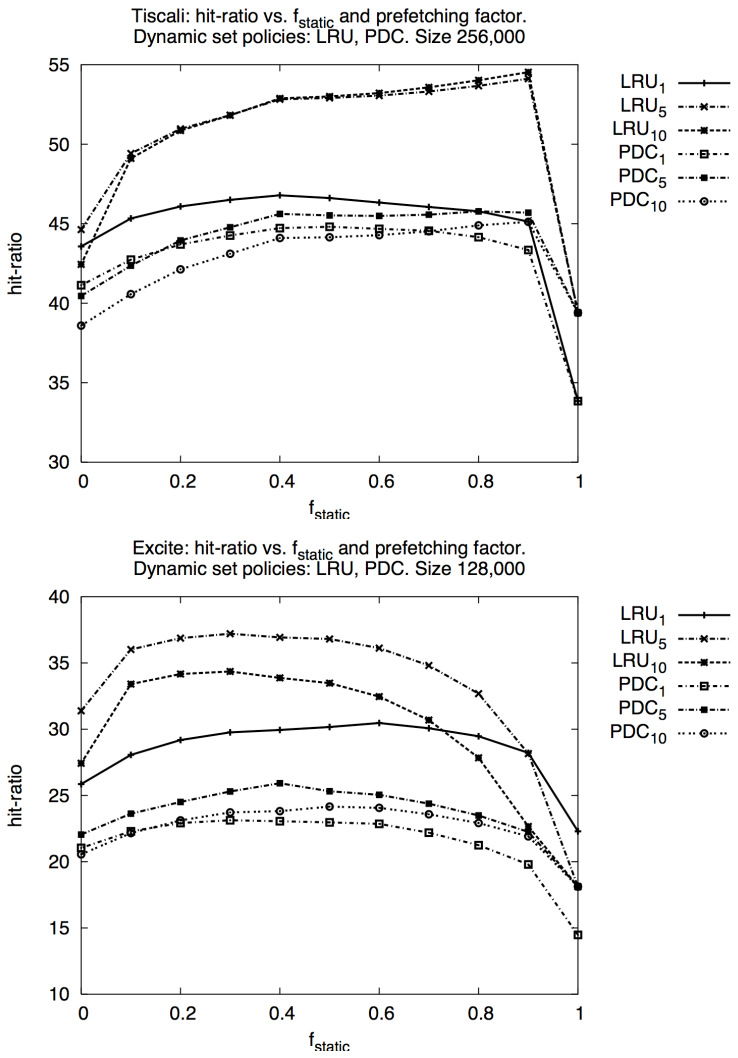


Fig. 7. Hit-ratios obtained on the *Tiscali* and *Excite* query logs with different replacement policies and different prefetching factors. The tests performed are referred to a cache of 256,000 entries for *Tiscali*, and 128,000 entries for *Excite*.

Therefore, an effective heuristic to adopt is to prefetch additional pages only when the cache miss has been caused by a request for a page different from the first one. In this way, since the prefetched pages will be actually accessed with sufficiently high probability, we avoid to fill the cache with pages that are accessed only rarely and, at the same time, we reduce the additional load on the core query service of the WSE. Our caching system thus adopts this simple prefetching heuristic whenever a *cache miss* occurs. The query is sent to the WSE for resolution, by using a prefetching factor k for the expanded query *(keywords, page_no, k)* chosen as follows:

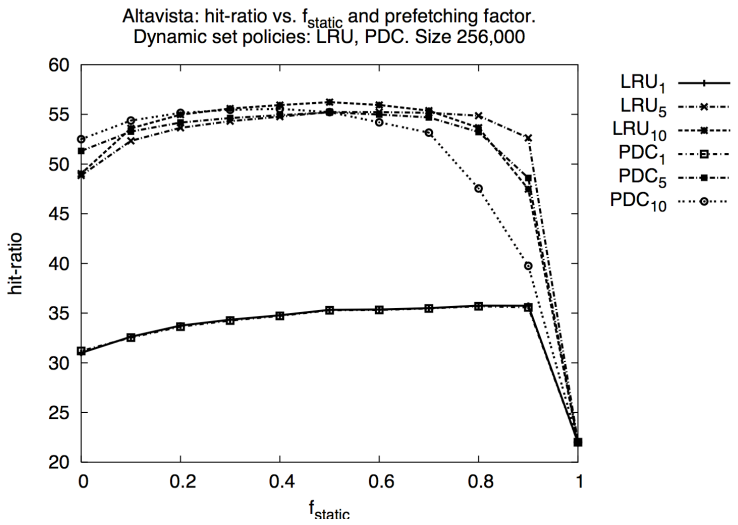


Fig. 8. Hit-ratios obtained on the *Altavista* query log with different replacement policies and different prefetching factors. The tests performed are referred to a cache of 256,000 entries.

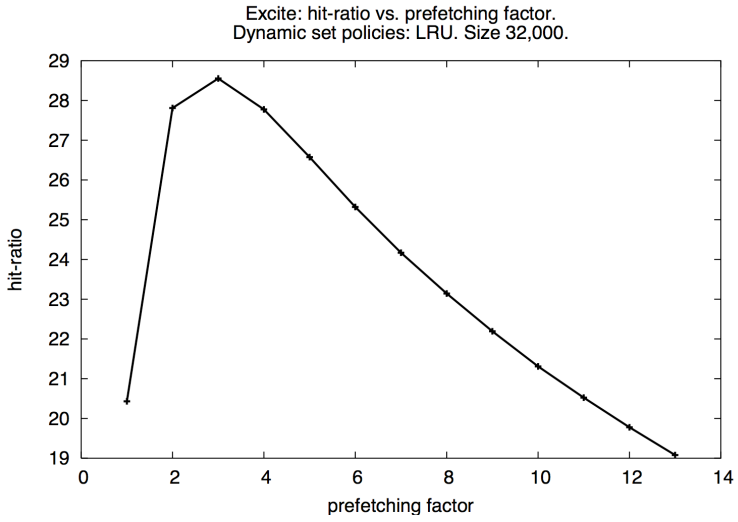


Fig. 9. Hit-ratio measured on the Excite query log with a SDC-*LRU* cache of 32,000 elements as a function of the prefetching factor.

$$k = \begin{cases} K_{MAX} & \text{if } page\_no > 1 \\ 2 & \text{otherwise} \end{cases} \quad (3)$$

In the formula above, K_{MAX} represents the maximum number of page prefetched by the cache. In practice the heuristic is very simple: whenever the first page is requested, we expand the query by requesting the first and the second pages. When the second page is requested, it is promptly returned to the user and the underlying WSE is asked for the successive K_{MAX} pages of results.

Whenever this *speculative* prefetching heuristic is adopted the percentage of prefetched pages actually referred increases remarkably. In the case above considered (the *Altavista* query log and a SDC-$LRU_{0.9}$ cache of 256,000 entries, ,with $K_{max} = 5$), the percentage of prefetched pages actually referred grows from 11.71% up to 30.3%). We can thus assert that the proposed *speculative* heuristic constitutes a good trade-off between the maximization of the benefits of prefetching, and the reduction of the additional load on the WSE.

6. CONCURRENCY ISSUES

We designed our caching system to allow several concurrent threads to efficient access its blocks. This is motivated by the fact that a WSE has to process several user queries concurrently, and this is usually achieved by making each query processed by a distinct thread. The methods exported by our caching system are thus thread-safe, and also ensure the mutual exclusion whenever it is necessary. In this regard, the advantage of SDC over a pure dynamic cache is related to the presence of the Static Set, which is a read-only data structure. Multiple threads can thus concurrently lookup the Static Set to search for the results of the submitted query. In case of a hit, the threads can also retrieve the associated page of results without synchronization. For this reason our caching system may sustain linear speed-up even in configurations containing a very large number of threads. Conversely, the Dynamic Set must be accessed in a critical section. Note, in fact, that the Dynamic Set is a read-write data structure: while a cache miss obviously causes both the associative memory and relative list of pointers to be modified, also a cache hit entails the list pointers to be modified in order to sort the cache entries according to the replacement policy adopted.

Our caching system can be easily integrated into a typical WSE operating environment. A possible placement is between the http server and the Broker, as shown in Figure 1. In the tests conducted to evaluate the throughput of our implementation, the behavior of the multi-threaded WSE http server, which forwards user queries to the cache system and waits for query results, was simulated with a farm of concurrent threads which read the queries from a log file and call the thread-safe methods exported by the cache implementation. The WSE core query service, which is invoked to resolve the queries that result in cache misses, was instead simulated by sleeping the thread which manages the query for an amount of time depending on the number of pages of results asked for. The behavior of each thread is very simple. In the case of a cache hit, the thread serving the query immediately returns the requested page of results, and gets another query. Conversely, when a query causes a cache miss, the thread sleeps for $\delta = 40$ms to simulate the

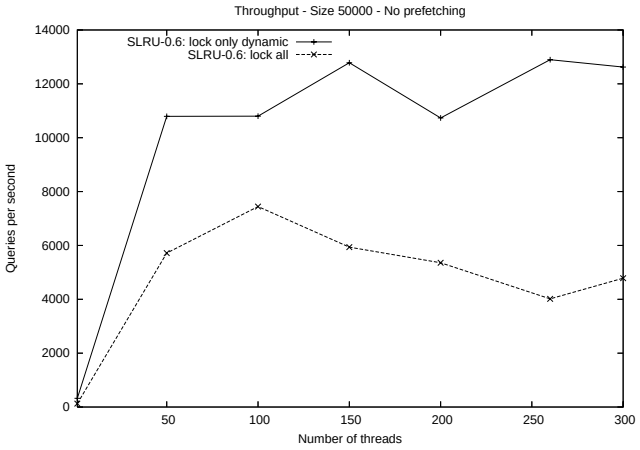


Fig. 10. Throughput of our caching system (in number of queries served per second) for $f_{static} = 0.6$ as a function of the number of concurrent threads used and different locking policies. LRU (a), PDC (b)

latency of the WSE core in resolving the query. In these tests we did not activate the prefetching policy, otherwise the value of δ should be changed accordingly.

Figure 10 reports the results of some of the performance tests conducted. In particular, the figure plots, for $f_{static} = 0.6$ and no prefetching, the throughput of our caching system (i.e., the number of queries answered per second) as a function of the number of concurrent threads sharing the cache. The two curves show the throughput of our system when each thread accesses in a critical section either the whole cache or just the Dynamic Set. Note that locking the whole cache is exactly the mandatory behavior of threads when accessing a purely dynamic cache (i.e., $f_{static} = 0$). Throughput was measured by considering that a large bunch of $500,000$ queries (from the *Tiscali* log) arrives in a burst. The size of the cache was $50,000$ blocks, while the replacement policy considered was *SLRU*.

It is worth noting that the adoption of a Static Set of about half of the entire cache entries approximately doubles the number of queries served per second. Moreover, the caching system does not only provides high throughput but can also sustain a large number of concurrent queries. Performance starts degrading only when more than 200 queries are served concurrently.

7. CONCLUSIONS

In this paper we presented SDC, a new startegy for caching the query results of a WSE. SDC exploits the knowledge about the queries submitted to the WSE in the past to enhance cache effectiveness. In particular, we maintain the most popular queries and associated results in a read-only static section of our cache. Only the queries that cannot be satisfied by the static cache section compete for the use of a dynamic, second-level cache. The benefits of adopting SDC were experimentally shown on the basis of tests conducted with three large query logs. In all the cases our

strategy remarkably outperformed either purely static or dynamic caching policies. We evaluated the hit-ratio achieved by varying the percentage of static blocks over the total, the size of the cache, as well as the replacement policy adopted for the dynamic section of our cache. Moreover, we evaluated cost and scalability of our cache implementation when executed in a multi-threaded environment. The SDC implementation resulted very efficient due to an accurate software design that allowed to make cache hit and miss times negligible, and to the presence of the read-only static cache that reduces the synchronizations between multiple threads concurrently accessing the cache. Finally, we showed that WSE query logs also exhibit spatial locality. Users, in fact, often require subsequent pages of results for the same query. Our caching system takes advantage of this locality by exploiting a *speculative* prefetching strategy which constitutes a good trade-off between the maximization of the benefits of prefetching, and the reduction of the additional load on the WSE. Future work will regard the estimation of how frequently the Static Set should be refreshed. Caching policies which are driven by statistical properties of data, may in fact suffer from a progressive performance degradation due to problems concerning the freshness of the data from which statistics have been drawn.

Acknowledgements

We acknowledge the financial support of the Project *Enhanced Content Delivery*, funded by the *Ministero Italiano dell'Università e della Ricerca*, Ideare S.p.A. for providing us the *Tiscali* query log, Ronny Lempel for the useful discussions and for providing us the *Altavista* query log.

REFERENCES

Barroso, L. A., Dean, J., and Hölze, U. 2003. Web search for a planet: The google cluster architecture. *IEEE Micro 22,* 2 (Mar/Apr).

Chidlovskii, B., Roncancio, C., and Schneider, M. 1999. Semantic cache mechanism for heterogeneous web querying. In *Proc. of the 8th Int. World Wide Web Conf.*

Dar, S., Franklin, M., Jonsson, B., Srivastava, D., and Tan, M. 1996. Semantic data caching and replacement. In *Proc. of the 22nd Int. Conf. on Very Large Database.* 330–341.

Fagni, T., Orlando, S., Palmerini, P., Perego, R., and Silvestri, F. 2003. A hybrid strategy for caching web search engine results. In *Proceedings of the twelfth international conference on World Wide Web.*

Feder, T., Motwani, R., Panigrahy, R., Seiden, S., van Stee, R., and Zhu, A. 2002. Web caching with request reordering. In *Proceedings of the 13th Annual Symposium on Discrete Algorithms.* 104–105.

Hoelscher, C. 1998. How internet experts search for information on the web. Paper presented at the World Conference of the World Wide Web, Internet, and Intranet, Orlando, FL.

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

Jansen, B. J. and Pooch, U. W. 2001. A review of web searching studies and a framework for future research. *Journal of the American Society of Information Science 52,* 3, 235–246.

Jansen, B. J., Spink, A., and Saracevic, T. 2000. Real life, real users, and real needs: a study and analysis of user queries on the web. *Inf. Proc. and Manag. 36,* 2, 207–227.

Johnson, T. and Shasha, D. 1994. 2q: A low overhead high performance buffer management replacement algorithm. In *Proc. 1994 VLDB.* 439–450.

KAREDLA, R., LOVE, J., AND WHERRY, B. 1994. Caching strategies to improve disk system performance. *IEEE Computer 27,* 3.

LEMPEL, R. AND MORAN, S. 2002. Optimizing results prefetching in web search engines with segmented indices. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB).*

LEMPEL, R. AND MORAN, S. 2003. Predictive caching and prefetching of query results in search engines. In *Proc. of the twelfth international conference on World Wide Web.* ACM Press, 19–28.

LU, Z. AND MCKINLEY, K. S. 1999. Partial replica selection based on relevance for information retrieval. In *SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15-19, 1999, Berkeley, CA, USA.* ACM, 97–104.

MARKATOS, E. P. 2000. On caching search engine results. In *Proc. of the 5th Int. Web Caching and Content Delivery Workshop.*

MARKATOS, E. P. 2001. On Caching Search Engine Query Results. *Computer Communications 24,* 2, 137–143.

O'NEIL, E. J., O'NEIL, P. E., AND WEIKUM, G. 1993. The lru-k page replacement algorithm for database disk buffer. In *Proc. of the 1993 ACM SIGMOD International Conference On Management Of Data.* 297–306.

ORLANDO, S., PEREGO, R., AND SILVESTRI, F. 2001. Design of a Parallel and Distributed WEB Search Engine. In *Proceedings of Parallel Computing (ParCo) 2001 conference.* Imperial College Press.

ROBINSON, J. T. AND DEVARAKONDA, M. V. 1990. Data cache management using frequency-based replacement. In *Proc. of the 1990 ACM SIGMETRICS Conference.* 134–142.

SARAIVA, P. C., DE MOURA, E. S., ZIVIANI, N., MEIRA, W., FONSECA, R., AND RIBEIRO-NETO, B. 2001. Rank–Preserving Two–Level Caching for Scalable Search Engines. In *Proceedings of the SIGIR2001 conference*, ACM, Ed. SIGIR, New Orleans, LA.

SILVERSTEIN, C., HENZINGER, M., MARAIS, H., AND MORICZ, M. 1999. Analysis of a very large web search engine query log. In *ACM SIGIR Forum.* 6–12.

SILVESTRI, F. 2004. High performance issues in web search engines: Algorithms and techniques. Ph.D. thesis, Università degli Studi di Pisa - Facoltà di Informatica.

SPINK, A., JANSEN, B., AND BATEMAN, J. 1998. Users' searching behavior on the EXCITE web search engine. In *Proceedings of WebNet 98.*

SPINK, A., JANSEN, B., WOLFRAM, D., AND SARACEVIC, T. 2001. Searching the web: the public and their queries. *J. Am. Soc. Inf. Sc. & Tech. 53,* 2, 226–234.

SPINK, A., JANSEN, B., WOLFRAM, D., AND SARACEVIC, T. 2002. From e-sex to e-commerce: Web search changes. *Computer 35,* 3 (March), 107–109.

WITTEN, I. H., MOFFAT, A., AND BELL, T. C. 1999. *Managing Gigabytes – Compressing and Indexing Documents and Images*, second edition ed. Morgan Kaufmann Publishing, San Francisco.

XIE, Y. AND O'HALLARON, D. 2002. Locality in search engine queries and its implications for caching. In *Proceedings of IEEE INFOCOM 2002, The 21st Annual Joint Conference of the IEEE Computer and Communications Societies.*