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

TIZIANO FAGNI, RAFFAELE PEREGO, and FABRIZIO SILVESTRI
Consiglio Nazionale delle Ricerche (CNR)
and
SALVATORE ORLANDO
Università Ca' Foscari di Venezia

This article discusses efficiency and effectiveness issues in caching the results of queries submitted to a Web search engine (WSE). We propose SDC (Static Dynamic Cache), a new caching strategy aimed to efficiently exploit the temporal and spatial locality present in the stream of processed queries. SDC extracts from historical usage data the results of the most frequently submitted queries and stores them in a *static*, *read-only* portion of the cache. The remaining entries of the cache are dynamically managed according to a given replacement policy and are used for those queries that cannot be satisfied by the static portion. Moreover, we improve the hit ratio of SDC by using an adaptive prefetching strategy, which anticipates future requests by introducing a limited overhead over the back-end WSE. We experimentally demonstrate 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 deploy and measure the throughput achieved by a concurrent version of our caching system. Our tests show how the SDC cache can be efficiently exploited by many 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

We acknowledge the financial support of the project Enhanced Content Delivery, funded by the Ministero Italiano dell'Università e della Ricerca.

Authors' addresses: T. Fagni, R. Perego, and F. Silvestri, Istituto ISTI A. Faedo, Consiglio Nazionale delle Ricerche (CNR), via Moruzzi 1, I-56100, Pisa, Italy; email: {tiziano.fagni,raffaele.perego, fabrizio.silvestri}@isti.cnr.it; S. Orlando, Dipartimento di Informatica, Università Ca' Foscari di Venezia, via Torino 155, I-30172 Mestre (VE), Italy; email: orlando@dsi.unive.it.

## 1. INTRODUCTION

Caching is an effective technique to make scalable a service that distributes data to a multitude of clients. As suggested by Xie and O'Hallaron [2002], Markatos [2000], Saraiva et al. [2001], Lempel and Moran [2003], and 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. WSE query results caching, as Web page caching, can occur at several places, for example, 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 WSE responsiveness, since many 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 many relatively expensive operations such as accessing the disk-resident index to retrieve the inverted lists associated with query keywords, intersecting them, ranking the results obtained on the basis of their relevance [Witten et al. 1999], retrieving URLs and snippets for each returned result, and, finally, building the HTML page returned to the user [Barroso et al. 2003]. With respect to these costs, the few microseconds spent for a lookup operation on an in-core cache (cache hit) can be considered negligible.

In this article, we are interested in studying the design and implementation of a server-side cache of query results to be deployed on the front-end of a high-performance parallel (and distributed) WSE. Starting from the analysis of the content of three real query logs, we propose a novel caching strategy called *SDC* (Static and Dynamic Cache.) SDC is a caching policy driven by statistical usage. The results of the most frequently accessed queries are then stored within a fixed-size set of statically locked cache entries. This is rebuilt at fixed time intervals using statistical data coming from WSE query logs. Each query is matched first against the *static set* of cache entries. If the request 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 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. Since the hit ratio achieved by the static set may suffer from the aging of the statistical information gathered from usage data, we analyzed how such information gets old as time progresses.

Moreover, we show 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 an *adaptive 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 many threads serving the user queries. We measured average hit and miss time, and tentatively assessed the throughput of our concurrent caching system (i.e., the number of queries resolved per time unit) with respect to the number of threads concurrently serving distinct user queries.

The rest of the article is organized as follows. Related work is 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 discusses future work.

## 2. RELATED WORK

Caching is routinely used in the Web since it allows the bandwidth consumption to be reduced, and the user-perceived quality of service to be improved. It is exploited at the client side, where the browser application temporarily stores accessed Web objects. It is used at the proxy level, where Web documents requested by one client may be cached for later retrieval by another client. Finally, caches can be placed at the server-side to reduce the number of requests that the server must actually handle [Podlipnig and Boszormenyi 2003]. In this article, we propose a novel server-side caching system which stores the results of the queries submitted by a multitude of users to a WSE. As noted by Xie and O'Hallaron [2002] and confirmed by our analysis of query logs, many popular queries are shared by different users. This level of sharing justifies the choice of exploiting a server-side WSE caching system. One of the issues in designing a server-side cache is the scarcity of resources usually available on the server, in particular the random-access memory needed to store 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 subtasks 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 subindex is relative to a disjoint subcollection 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

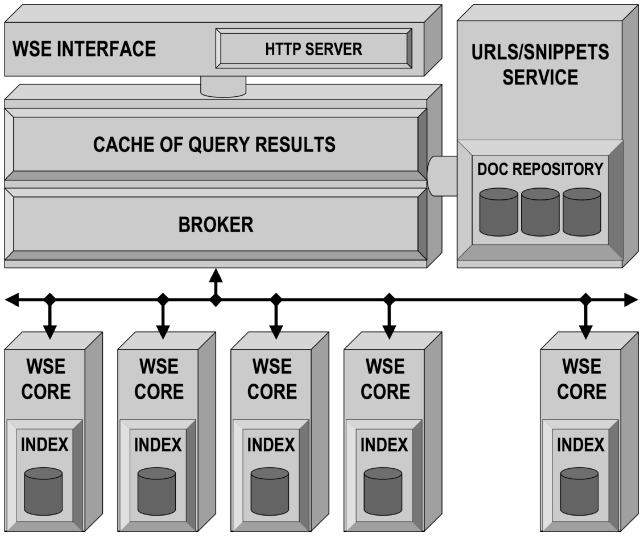



Fig. 1. Architecture of a large-scale, distributed WSE hosting a cache of query results.

and orders the results received on the basis of their relevance, and produces a ranked vector of document identifiers (DocIDs), for example, a vector composed of 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) 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 multithreading is exploited extensively by all these modules in order to process concurrently distinct queries. Within this architecture the random-access 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 random-access memory is a less critical resource for the 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 the 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 exploited by the WSE back-end.

Depending on the memory available on the broker, we could devise a cache whose blocks stores the complete HTML pages returned to users (*HTML cache*), or a cache that simply stores the DocID vectors (*DocID cache*). Note that, given a fixed memory size, the DocID cache could include a larger amount of blocks than an HTML cache, since in that case each block is a small vector of integer identifiers. Conversely, a hit in a DocID cache only returns a vector of DocIDs, while the HTML page to be delivered to the user has still to be prepared. Independently of the kind of the cache (DocID or HTML), in this article we will call *page of results* the content of a cache block.

Many works have studied the behavior of WSE users by analyzing usage data. One of the first articles analyzing the possible ways of exploiting user feedback was authored by Raghavan and Sever [1995]. Their article proposed the use of a query base, built on the basis of a set of persistent optimal queries submitted in the past. This query base is exploited to improve the retrieval effectiveness for the queries which appear to be similar to the ones in the query base according to given similarity measures. Even though the main focus of this work was not on caching, it represented one of the first attempts of reusing past knowledge about usage in order to enhance the retrieval process.

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

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

Spink et al. [2001] deeply analyzed the *Excite* query log, the one used also in our tests. They showed many different characteristics of the queries contained within the log, and also of the users search session. Similarly to us, they discovered that Web queries are, on average, short and quite simple in structure. Few users uses advanced search features, and when they do half of them are mistakes. Finally, Web users, usually, view just the first page of results. Even though with different goals in mind, the main findings of their work do not differ too much from the results of our log analysis. Indeed, the differences regard topics that are not useful for the justification of our main results about Web query caching policies.

Beitzel et al. [2004] analyzed a very large Web query log of America On Line containing the queries submitted during a whole week. The queries came from a population of tens of millions of users searching the Web for a wide variety of topics. Differently from the analysis presented in Silverstein et al. [1999], they partitioned the query log into groups of queries submitted in different hours of the day. The analysis, then, tried to highlight the changes in popularity and uniqueness of topically categorized queries within the different groups. Query repetition rate was observed to be nearly constant throughout the day, although each frequent query did not appear often during an hour. Another interesting finding was that the queries submitted during peak hours were more similar to each other than their nonpeak hour counterparts. An interesting supposition of the authors was that it is possible to devise predictive functions able to estimate the likelihood of a query being repeated. This last point, in particular, is very important for caching. The capability of estimating in advance queries that are

likely to be or not to be repeated in the future may be profitably exploited by the cache replacement policy.

Markatos [2000, 2001] 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 were successively resubmitted by either the same or other users.

Finally, Lempel and Moran [2003] presented an analysis of a query log containing around 7.2 millions queries submitted to the Alta Vista search engine in the summer of 2001. They discovered that about 63.5% of the views regarded the first page of results only. On the other hand, the views accounted for the second page of results were about 11.7% of the total. Another important statistical fact discovered by Lempel and Moran regarded topic popularity, that is, the number of times a particular query was requested. They showed that this popularity followed an inverse power-law distribution. Practically speaking, most queries were requested only once, while only a few of them were requested multiple times. We confirmed that our query logs also followed the same distribution.

Only a few of the works evaluating the features of WSE query logs also proposed effective techniques to exploit the locality present in the stream of queries. Although it is virtually certain that some commercial Web search companies adopted query results caching from the beginning of their existence, the first scientific article that proposed caching as a means to reduce the response time of a WSE appeared in 2000 [Markatos 2000, 2001]. In this article, Markatos described different state-of-the-art caching policies and compared the hit ratio obtained on a log composed of queries submitted to Excite. Nevertheless, he did not propose any policy tailored to specific statistical properties of the Excite log, and 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 article describing a WSE caching system is the one by 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 in 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 interesting point is that the authors experimented with 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 the request rate increased, the second-level effectively helped the disk in serving the requests, thus increasing the overall throughput.

According to the distributed architecture of Figure 1, other levels of caching can be added without interfering with the effectiveness of a query-result caching system like the one proposed in this article. The responsiveness and the overall throughput of the WSE can improve if a postings cache on the machines which store the index and resolve queries, or a Url/Document cache on the *Urls/Snippets Server*, are added which effectively reduce the latency of operations performed remotely from the point of view of the broker.

The work presented by Long and Suel [2005] goes in this direction and presents a caching system structured according to three different levels. The intermediate level contains frequently occurring pairs of terms and stores intersections of projections of the corresponding inverted lists. The authors proposed and studied both off-line and on-line caching algorithms. The experimental evaluation, based on a large Web crawl and a real search engine query log, showed significant performance improvements for the intermediate caching level, both in isolation and in combination with the other caching levels.

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: highly probable queries are 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 to the first one. For queries regarding the first page of results, an SLRU policy is used. SLRU [Karedla et al. 1994] maintains two LRU segments, a protected segment and a probationary segment. Pages are first placed in the probationary segment; if requested again, they are moved to the protected segment. In regard to evictions, pages evicted from the protected segment are moved to the probationary segment, while pages evicted from the probationary segment are removed from the cache.

PDC also adopts prefetching to anticipate user requests. To this purpose, PDC exploits a model of user behavior. A user session starts with a query for the first page of results, and can proceed with one or more *followup* queries (i.e., queries requesting successive page of results). When no followup queries are received within $\tau$ 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 $\tau$ seconds ago. To keep track of query priorities, a priority queue is used. PDC results measured on a query log of Alta Vista were very promising. With a cache of 256, 000 elements and prefetching 10 (i.e., nine further pages of results are always requested along with the first one), the authors measured a hit ratio of about 53.5%. Unfortunately the PDC policy is expensive from the computational point of view: its amortized complexity is logarithmic in the size of the priority queue.

## 3. ANALYSIS OF THE QUERY LOGS

In order to evaluate the behavior of different caching strategies, we used three real query logs. In particular we used *Tiscali*, a trace of queries submitted to the Tiscali WSE (www.janas.it) in April 2002, *Excite*, a publicly available trace of the queries submitted to the Excite WSE (www.excite.com) on September 16th 1997, and *Alta Vista*, a query log containing queries submitted to Alta Vista (www.altavista.com) in the summer of 2001. The *Excite* log is the same as the

Table I. Main Characteristics of the Query Logs Used

| Query Log | Queries | Distinct Queries | Date |
|---|---|---|---|
| *Excite* | 2,475,684 | 1,598,908 | September 16th, 1997 |
| *Tiscali* | 3,278,211 | 1,538,934 | April 2002 |
| *Alta Vista* | 7,175,648 | 2,657,410 | Summer of 2001 |

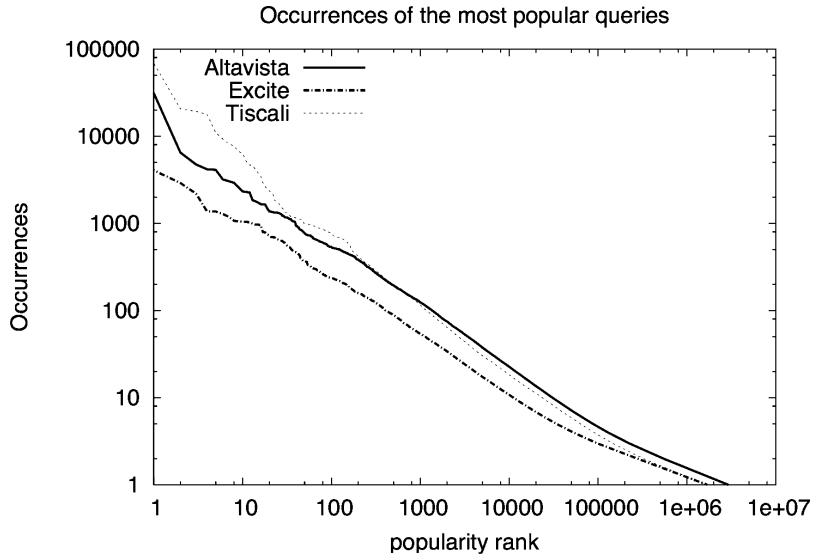
one used by Markatos [2000] and by Spink et al. [2001], while the *Alta Vista* log was also used by Lempel and Moran [2003]. Each record of a query log refers to a single query submitted to the WSE for requesting a *page* of results, where each page contains a fixed amount of URLs ordered according to a given rank. All query logs were preliminarily cleaned by converting query words to lowercase. In order to unify the different formats of the logs, we also removed useless fields such as, for example, the timestamps and client identifiers in the *Excite* log. We further normalized the query log entries by removing those referring to requests of more than 10 results per page.[1] Finally, it is worth noting that we didn't remove stopwords (except in the Excite case discussed below), and we didn't reorder query words. Thus, similarly to real-world search engines like Google, we considered the queries "new york" and "NEW YORK" the same query, which is different from both the query "york new" and "the new york." At the end of this preprocessing phase, each entry in each one of our query logs has 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.

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

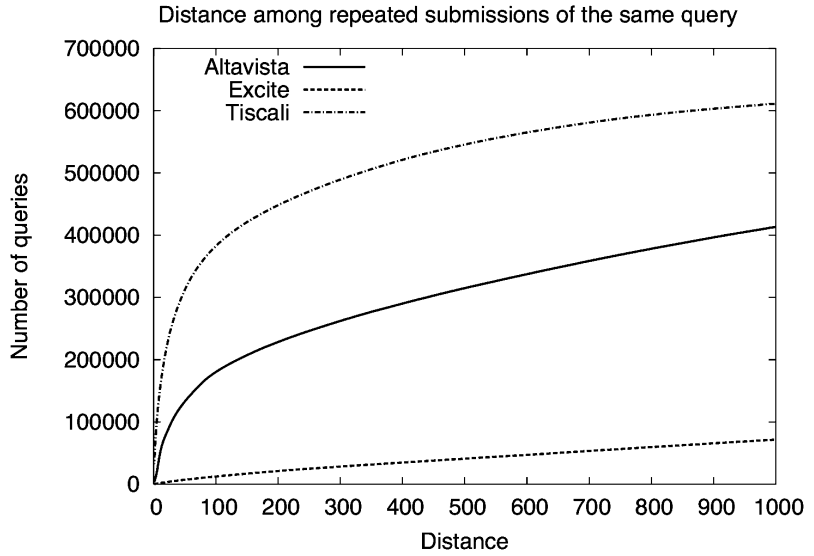
### 3.1 Temporal Locality

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 query popularity follows an inverse power-law distribution in all the three logs. 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*

[1]Since our cache stores fixed-size blocks, in the tests conducted we always added the number of pruned queries requesting more than 10 results to the number of cache misses.




(a)




(b)

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.

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: in more than 350,000 cases the distance between successive submissions of the same query was less than 100 in the *Tiscali* log; this distance is slightly smaller in

Table II. Percentage of Queries in the Logs as a Function of the Index of the Page Requested

| Query Log | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| *Excite* | 77.59 | 8.92 | 3.98 | 2.37 | 1.54 | 1.09 | 0.78 | 0.60 | 0.45 | 0.37 |
| *Tiscali* | 83.20 | 5.50 | 2.86 | 1.74 | 1.23 | 0.74 | 0.54 | 0.41 | 0.32 | 0.26 |
| *Alta Vista* | 64.76 | 10.25 | 5.68 | 3.41 | 2.54 | 1.83 | 1.42 | 1.16 | 0.94 | 0.88 |

the *Alta Vista* log, but it is still less than 100 in more than 150,000 cases. In the *Excite* log, we encountered a lower temporal locality. However, also in this log in more than 10,000 cases some queries were repeated at a distance smaller than 100. We think that the lower locality present in the *Excite* log was mainly due to its older age. It contains many long queries expressed in natural language like "Where can I find information on Michael Jordan." In the above query, the first terms are very general and thus of poor significance, while only the last two words really discriminate results. Such kinds of queries can be considered a symptom of the poor capacity of the users to interact with a WSE more than 8 years ago. In order to address the locality issues deriving from the age of the Excite log, we cleaned its queries by removing a small set of stopwords, that is, by eliminating meaningless terms like articles, adverbs, and conjunctions.

### 3.2 Spatial Locality

Although different in some assumptions and in some conclusions, the works surveyed in Section 2 show that WSE users in most cases submit short queries and only visit 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. We consider this behavior as the presence of a limited spatial locality in the stream of queries processed by a WSE. At the memory cache level, spatial locality means that, if an instruction accesses a particular memory location, often nearby memory locations will be soon accessed. In our case, spatial locality means that, given a query requesting a page of relevant results, often 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. Table II and Figure 3 report the results of this analysis. In particular, Table II shows the percentage of queries in each log file as a function of the index of the requested page. As expected, most queries required the first page of results only. On the other hand, while there was a huge difference between the number of queries requesting the first and the second page, this difference becomes less remarkable when we consider the second and the third page, the third and the fourth, and so on. To highlight this behavior, Figure 3 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 can be seen from Figure 3, 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 on the first page. Otherwise, when a generic

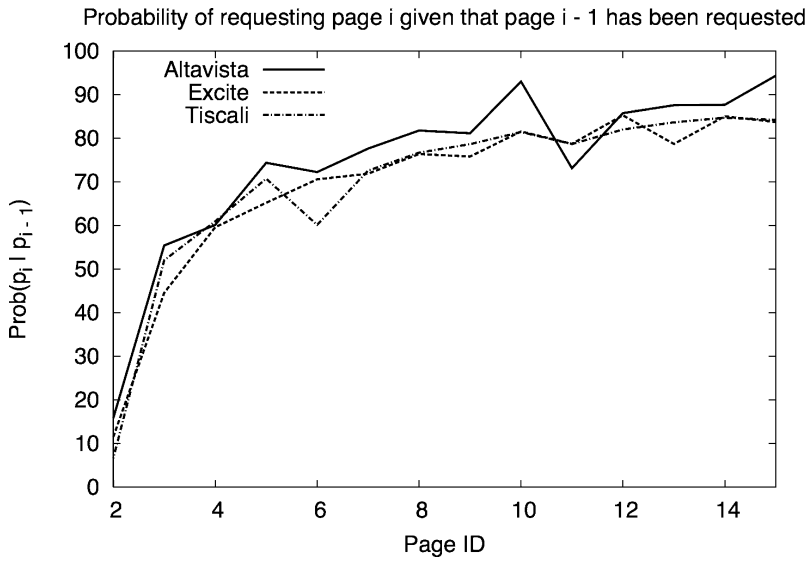



Fig. 3. Probability of the occurrence of a request for the $i$th page given a previous request for page $(i-1)$.

query is submitted, it is necessary to browse many pages in order to find the relevant information.

### 3.3 Theoretical Upper Bounds on the Cache Hit Ratio

From the data reported in Table I it is easy to devise the theoretical upper bounds on the hit ratios achievable on the three query logs when prefetching is not used. To this end, let us suppose the availability of a cache of infinite size, so that only compulsory misses have to be taken into account, that is, cache misses corresponding to the first reference to each distinct query. The rate of compulsory misses is thus

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

where $D$ is the number of distinct queries, and $Q$ is the total number of queries in the log. The value $m$ is the minimum miss ratio, while the maximum hit ratio, $H$, is obviously given by

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

The analysis becomes a little more complicated when we introduce 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 we retrieve each time $k$ successive pages of results starting from the one requested by the user (hereinafter we will call $k$ the prefetching factor), 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 ($k+1 \leq page\_no \leq 2 \cdot k$),

Table III. Hit-Ratio Upper Bounds for Each Query Log as a Function of the Number of Pages of Results Prefetched

| | Query Log | | |
|---|---|---|---|
| Pages Prefetched | *Excite* | *Tiscali* | *Alta Vista* |
| No prefetching | 35.41 | 53.05 | 56.97 |
| 2 | 45.43 | 56.65 | 60.89 |
| 3 | 47.23 | 56.86 | 61.72 |
| 4 | 47.49 | 56.97 | 62.08 |
| 5 | 47.54 | 56.99 | 62.28 |
| 6 | 47.56 | 56.99 | 62.40 |
| 7 | 47.57 | 57.00 | 62.49 |
| 8 | 47.57 | 57.00 | 62.57 |
| 9 | 47.57 | 57.00 | 62.64 |
| 10 | 47.57 | 57.00 | 62.72 |
| 15 | 47.58 | 57.00 | 62.96 |
| 20 | 47.58 | 57.00 | 63.17 |

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, that is, the minimum miss ratio $m$, while the maximum hit ratio $H$ is increased accordingly. Table III shows the results computed on the three query logs.

As can be seen from the values reported in the table, increasing the prefetching factor does not always result in a corresponding increase to hit ratio bounds. In particular, for prefetching factors greater than 4, the variations are very small. We 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 *Excite*. Its log contains poorly expressed queries and the results returned were perhaps not effective. As a result, users had to browse more than one page of results in order to satisfy their needs.

## 4. THE SDC POLICY

SDC is a hybrid 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 appearing 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 given 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 an inverse power-law distribution [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 was already proposed by Markatos [2000], where a purely static caching policy for WSE results was proposed and compared with purely dynamic ones. On the other hand, some queries are popular only within relatively short time intervals, or may become suddenly popular due to, for example, unforeseen events (e.g., the September 11, 2001, attacks). Since these queries clearly cannot profit from a static cache, we introduced the dynamic set. The advantages deriving from this novel hybrid caching strategy are twofold: 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.

### 4.1 Static Set

The static cache has to be initialized offline, that is, 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 function. 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 $f_{static}$ is 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.

### 4.2 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. The literature on caching proposes many replacement policies which, in order to maximize the hit ratio, try to take the largest advantage possible 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 pages most frequently referred in the past, 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 an 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].

The replacement policy to be used is chosen at the startup 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 $f_{static} = 1 - s$. 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 whose size is 40% of the total size of the cache.

## 5. EXPERIMENTS

All the experiments were conducted on a Linux PC equipped with a 2-GHz Pentium Xeon processor and 1 GB of random-access memory. All the tests discussed in this section were performed by running a single process that reads the queries from the log and manage the cache. Concurrency issues and the tests performed in a multithreaded environment are discussed in Section 6.

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 two-thirds 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 were used 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 started 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, 5, and 6 show the cache hit ratios obtained by SDC on the *Tiscali*, *Excite*, and *Alta Vista* 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 at 256,000 blocks. The plots reveal some interesting things. First, we 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. Second, 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

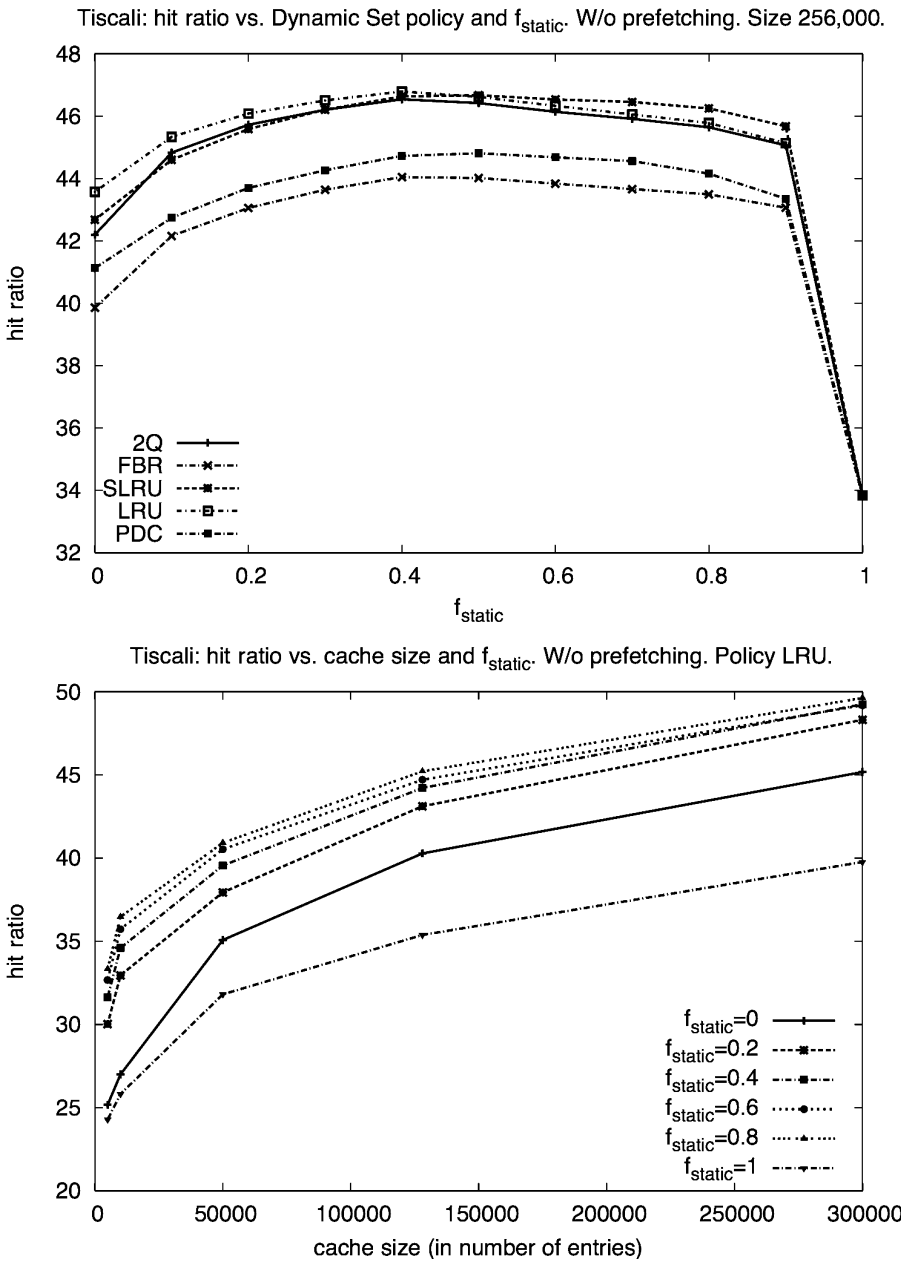



Fig. 4. Hit ratios achieved on the *Tiscali* query log for different replacement policies and varying values of $f_{static}$, and for different sizes of the cache.

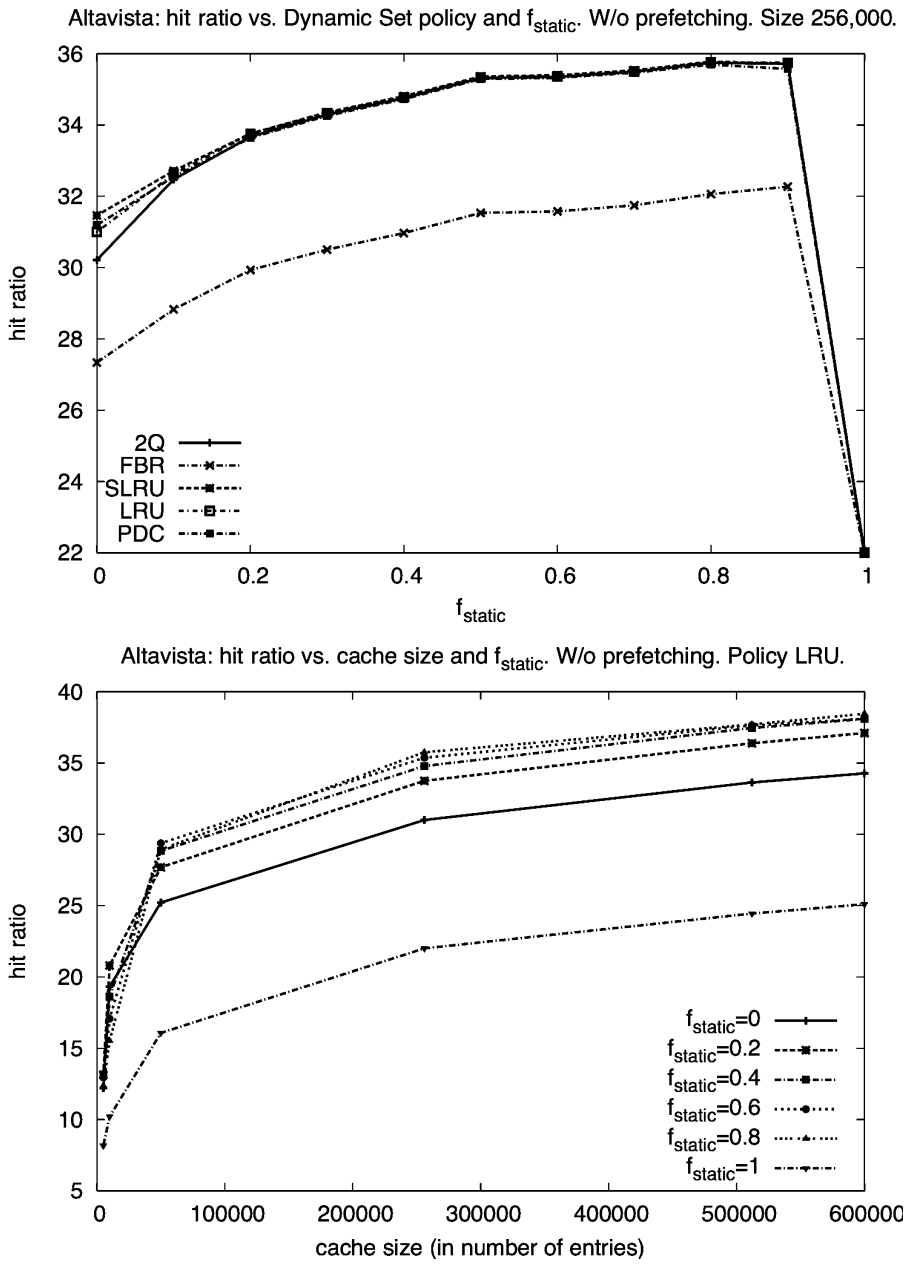



Fig. 5. Hit ratios achieved on the *Alta Vista* query log for different replacement policies and varying values of $f_{static}$, and for different sizes of the cache.

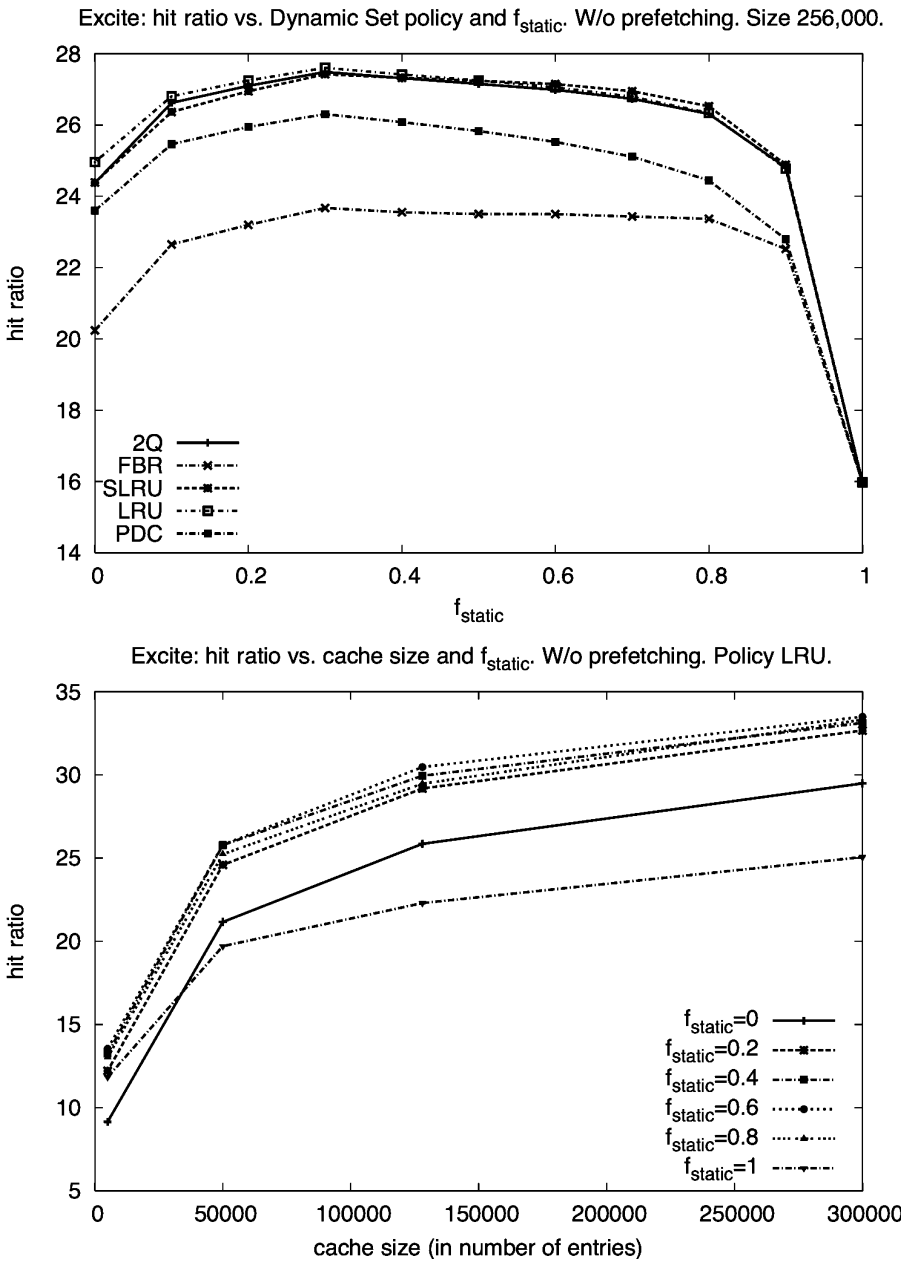



Fig. 6. Hit ratios achieved on the *Excite* query log for different replacement policies and varying values of $f_{static}$, and for different sizes of the cache.

Table IV. Average Miss and Hit Times (in Microseconds) for Different Sizes of an SDC-$LRU_{0.5}$ Cache

| Cache size | Hit time, static set | Hit time, dynamic set | Miss time |
|---|---|---|---|
| 32,000 | 10.00 | 13.79 | 61.01 |
| 64,000 | 10.25 | 14.07 | 60.54 |
| 128,000 | 10.72 | 14.15 | 61.45 |
| 256,000 | 11.29 | 14.48 | 60.92 |
| 512,000 | 12.62 | 14.71 | 61.07 |

$f_{static}$ depends on the query log considered: the maximum hit ratio was achieved for values of $f_{static}$ equal to 0.4, 0.8, and 0.3 in the case of the *Tiscali*, *Alta Vista*, and *Excite* query logs respectively. However, since differences in the hit ratio were slight when $f_{static}$ ranged between 0.3 and 0.8, setting it to 0.7 can be considered a good compromise for all the logs considered. Moreover, a higher value for $f_{static}$ resulted in a larger read-only portion of the cache. In the case of a multithread cache system, the read-only section allowed software lockout effects to be reduced and the throughput of the WSE to be remarkably enhanced.

Finally, we can see that the different replacement policies for the dynamic set behaved similarly with respect to the SDC hit ratio. For example, on the *Alta Vista* log, all the curves but that for FBR almost completely overlapped. This behavior seems to suggest that the most popular queries were satisfied by the static set. Thus, the dynamic set was only exploited by those queries which we can define as *burst queries*, that is, those which appeared frequently just for a brief period of time. For these queries, the small variations in the hit-ratio figures seem not to 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 in the bottom graphs of Figures 4, 5, and 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 was increased, the hit ratios increased correspondingly. In the case of the *Alta Vista* log, the highest hit ratio achieved with a cache of 50,000 blocks was about 29%, and with a cache of 256,000 was about 36%. For all the sizes tested, our strategy remarkably outperformed either purely static ($f_{static} = 1$) or dynamic ($f_{static} = 0$) caching policies.

Since our logs are not huge, we did not test caches of larger sizes. To make the comparison with related works easier, similarly to Markatos [2000], Saraiva et al. [2001], and Lempel and Moran [2003], we used a cache of 256,000 pages in most of the tests. Note that the actual memory requirements were limited: a cache of 256,000 blocks storing uncompressed HTML pages of 4 kB each requires about 1 GB of random-access memory.

In regard to cache management costs, Table IV shows the average times spent by our implementation to manage hits and misses as a function of the size of the cache. The tests were conducted on the *Alta Vista* query log with a single thread accessing an SDC-$LRU_{0.5}$ cache. Miss times reported in the table include only the times spent for managing the cache in the case of a miss—that is, the time needed for the choice of the page to be evicted, the replacement

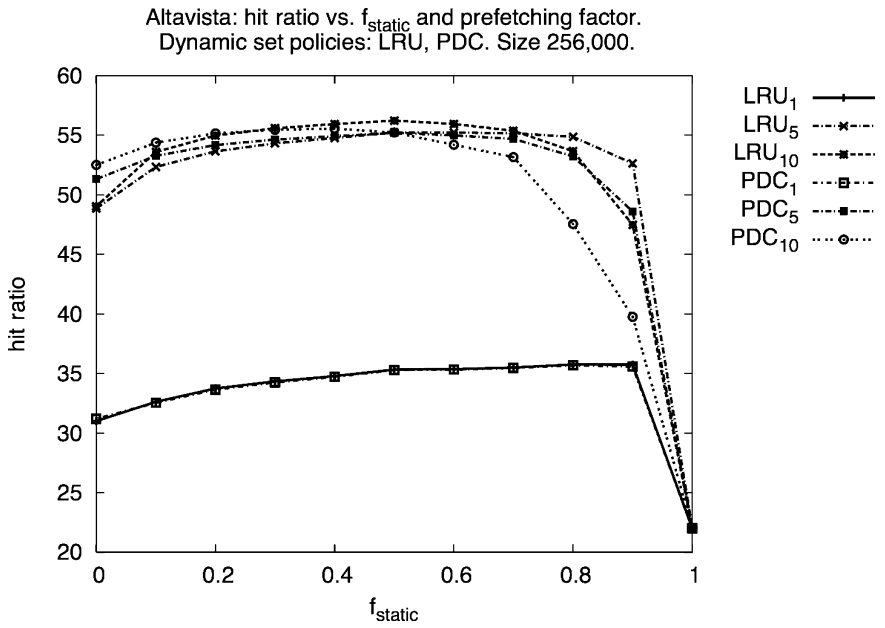



Fig. 7. Hit ratios obtained on the *Alta Vista* query log with different replacement policies and different prefetching factors. The tests performed were referred to a cache of 256,000 entries.

of the cache block, and the update of the priority queue—and do not take into account the times required by the WSE back-end to retrieve the requested page of results. As it can be seen, cache management times are very low, and almost independent of the size of the cache when an O(1) replacement policy like LRU is used.

## 5.2 SDC and Prefetching

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 *(keywords, page_no, k)* requesting $k$ consecutive pages starting from page *page_no*, where $k \geq 1$ is the *prefetching factor*, might be forwarded to the core WSE query service. 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 which are not already in cache are stored into $k$ distinct blocks of the dynamic part 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 in the cache.

Figure 7 shows the results of the tests conducted on the *Alta Vista* query log with SDC and prefetching. In particular, we tested SDC with either LRU or PDC as the replacement policy for the dynamic set, with no prefetching (subscript of the curve label $= 1$), and with a prefetching factor $k = 5, 10$. The tests on

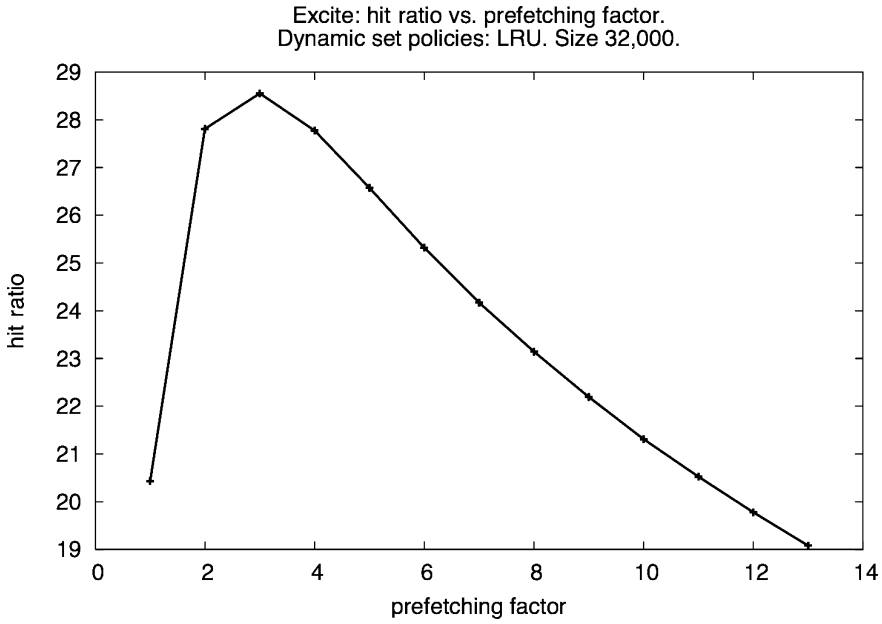



Fig. 8. Hit ratio measured on the *Excite* query log with an *LRU* cache of 32,000 elements as a function of the prefetching factor.

the other logs gave similar results and the corresponding plots are thus not reported.

As expected, prefetching increased the hit ratio achieved. For example, on the *Alta Vista* query log with an SDC-*LRU* cache of 256,000 entries, we obtained a maximum hit ratio of 54.20 when 10 pages of results were prefetched ($k = 10$), versus a hit ratio of 35.76 when no pages were prefetched. In this case, the adoption of prefetching increased the hit ratio by about 50%. Unfortunately the prefetching factor $k$ cannot grow indefinitely. In fact, all the prefetched pages of results have to be inserted in the cache by likely evicting from it an equal number of entries according to the replacement policy adopted. Obviously, the hit ratio increases only if the probability of accessing the prefetched pages is greater than the evicted ones. On the *Alta Vista* query log, we measured that with an SDC-$LRU_{0.1}$ cache of 256,000 entries and a prefetching factor of 5, only 11.71% of the prefetched pages were actually referred to by the following queries.

Particularly for caches of small size, prefetching might negatively affect the effectiveness of the cache replacement policy adopted. In this regard, Figure 8 plots the results of the test conducted with an LRU cache of 32,000 entries on the *Excite* query log. As can be seen, with this small cache the hit ratio started decreasing when a prefetching factor greater than 3 was used.

Moreover, prefetching increases the load on the WSE back-end and may degrade its throughput. Although the cost of resolving a query is not linearly proportional to the number of results retrieved [Witten et al. 1999; Moffat and Zobel 2004], some additional costs must be paid.

Table V. Behavior of SDC with Adaptive Prefetching

| $n$ = requested page_no | Hit/Miss | Cache Action |
|---|---|---|
| $n = 1$ | Hit | Return page 1 |
| | Miss | Submit query *(keywords, 1, 2)* |
| $n = 2$ | Hit | Return page 2 and submit query *(keywords, 3, K)* |
| | Miss | Submit query *(keywords, 2, K)* |
| $n > 2$ | Hit | Return page $n$ |
| | Miss | Submit query *(keywords, n, K)* |

In order to choose a strategy that maximizes the benefits of prefetching, that is, that maximizes the responsiveness of the system, and, at the same time, limits the additional load on the WSE back-end, 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 $i > 1$.

Therefore, it seems profitable to prefetch a limited number of 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 filling 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. Given a query *(keywords, page_n)*, our cache system behaves in the way specified in Table V, where $K$ represents the maximum number of pages prefetched by the cache.

In practice, the heuristic is very simple: whenever the first page is requested and it is not in the cache, we expand the query by asking the WSE for the first and the second pages, which are both inserted into the cache. When the second page is requested, it is promptly returned to the user, but the underlying WSE is asked for the successive $K$ pages of results. In this way, we grant responsiveness and hit ratios roughly equal to those obtained by adopting a static prefetching factor $K + 1$, but we pay most of the cost of prefetching only when the probability of having references to the prefetched pages in the near future is about 0.5 or greater. Whenever this adaptive prefetching heuristic is adopted, the percentage of prefetched pages actually referred increases remarkably. In the case considered above (the *Alta Vista* query log and an SDC-$LRU_{0.1}$ cache of 256,000 entries, with $K = 5$), the percentage of prefetched pages actually referred grew from 11.71% to 30.3%. We can thus assert that the proposed heuristic constitutes a good tradeoff between the maximization of the benefits of prefetching and the reduction of the additional load on the WSE.

### 5.3 Freshness of the Static Set

Caching policies like SDC and PDC, which are driven by statistical data, may suffer from performance degradation due to problems concerning the freshness of the data from which the statistics have been drawn. Some interesting

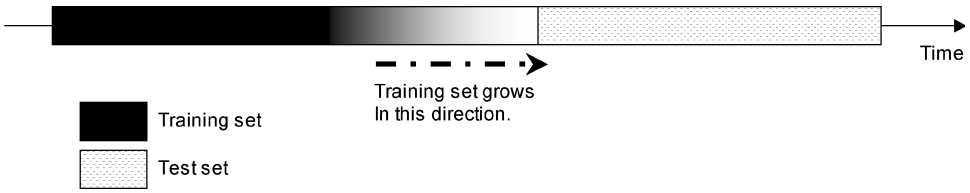



Fig. 9. Schema of the training process for the static set: the training set grows on the right, thus reducing the size of the test set.

questions arise in this regard:

—How frequently should the static set of the cache be refreshed? Does the hit ratio achieved by the static set degrade constantly as time goes by?
—Do time-of-day patterns exist in the query stream?

To answer these questions, we analyzed the *Alta Vista* log to measure how the frequent queries, used to fill the static set, were referred to in terms of time.

The log contained 7,175,648 queries spanning a period of about 4.7 days. We partitioned the log into two distinct sets: the *training set*, and the *test set* (see Figure 9). From the training set we extracted the $S$ most frequent queries, where $S$ is the size of the static set. For these experiments, we fixed $S$ to 128,000 elements.

By varying the time on which the static set is trained (as shown in Figure 9), we measured the static set hit ratio for the remaining part of the log (i.e., for the test set), and we plotted it at regular time intervals. The results of the experiment are shown in Figure 10. Each curve represents the trend of the hit-ratio value as a function of the time for a different training window.

5.3.1 *Static Set Refresh Rate.* When trained with the queries submitted to Alta Vista during 1 h only, the hit ratio on the static set was initially very high, but degraded rapidly to very low values (below 8%). As we increased the training period, the curves became higher and flatter since the hit ratio degraded less remarkably as time progressed. For example, when the static set was trained with the queries of 1 day, the hit ratio achieved on the remaining test set of about 3 days ranged from 19% to 17%, with a slow progressive degradation.

Finally, when trained for 2 days, the performance improved further. In this case, however, the remaining portion of the log used as test set was too small to allow us to draw strong conclusions.

Some observations can be made on the basis of the above results. First, the freshness of the static set does not appear to be a big issue. When statistics are drawn over a relatively long period of time, the degradation rate of the hit ratio in the static set is quite slow. Thus, a static set refreshed daily should grant good and quite stable performances for a relatively extended period. Moreover, from the high initial value and fast degradation rate of the curve relative to the shortest training time in the experiment, we can deduce that some queries appear relatively frequently but just for a brief period of time. Thus, their particular kind of locality is difficult to exploit with a static cache. SDC successfully

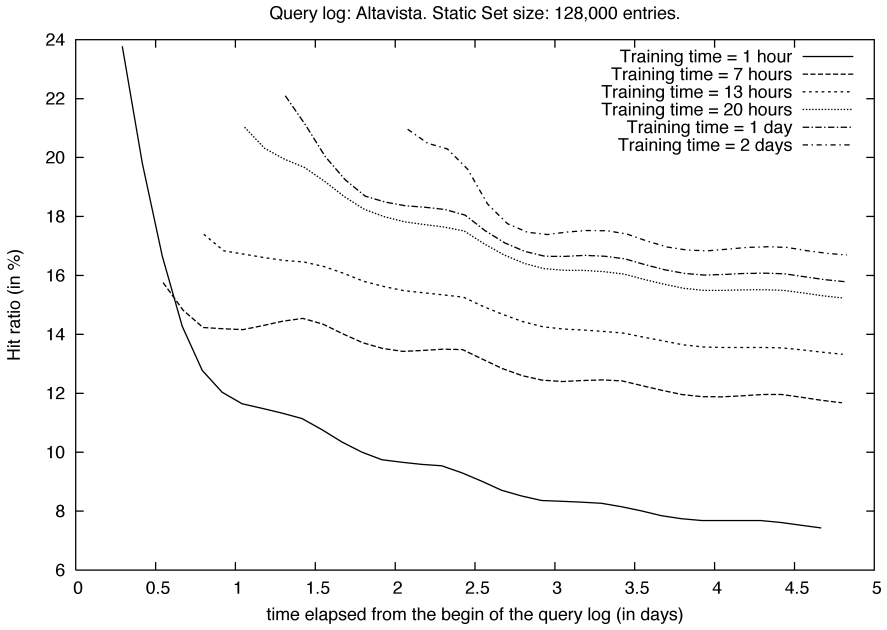



Fig. 10. Static set hit ratio on the *Alta Vista* query log as a function of time. The size of the static set has been set to 128,000 elements, while the training time was varied between 1 h and 2 days.

exploits also the locality of these "burst" queries due to the presence of its dynamic set.

5.3.2 *Daily Patterns.* From Figure 10 we can see that all the curves plotted have a common behavior: there is a sort of periodicity in the hit ratio that results in a smooth sinusoidal trend of the curves.

Around each 24 h the hit ratio reaches a local peak and starts to degrade until a local minimum is reached after about 12 h. We looked at the query log and we observed that the peak time was around midnight. This particular trend demonstrates the presence in the log of groups of frequent queries which are more or less popular in specific hours of each day. The benefits to be derived from the exploitation of the knowledge of such daily patterns merits further investigation. One promising source in this regard are the results presented in Beitzel et al. [2004].

## 5.4 Effectiveness of the Static Set

The rationale of introducing our hybrid, static and dynamic, cache, relies on the hypothesis that some popular queries may not be requested for relatively long time intervals, and might be unprofitably evicted from a purely dynamic cache. On the other hand, we saw that there are queries that are popular only within relatively short time intervals, and thus do not benefit from the use of a purely static cache filled with most popular queries, but can be effectively handled by the dynamic portion of our cache.

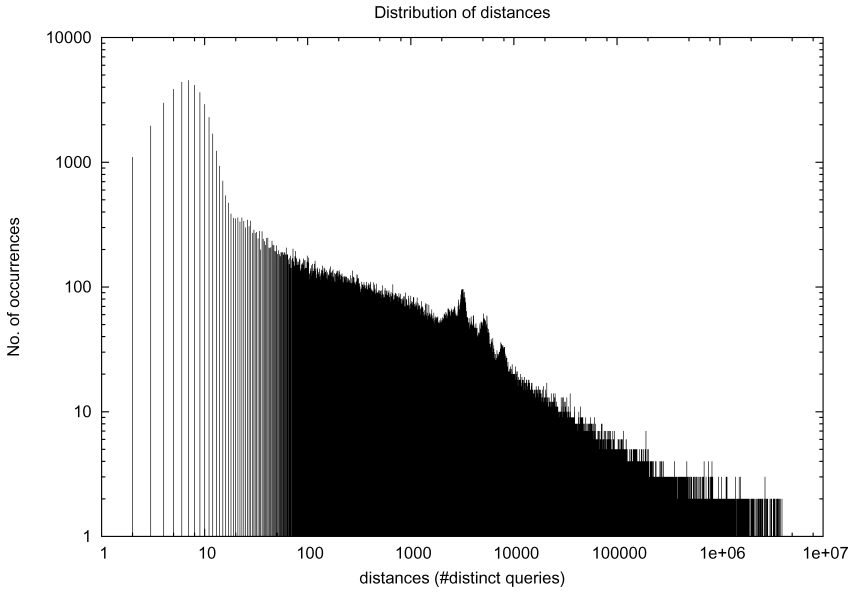



Fig. 11. Distances, expressed as the number of distinct queries received, between successive submissions of each one of the 128,000 most popular queries present in the *Alta Vista* log.

We conducted some tests on the *Alta Vista* log aimed at estimating how many times a *frequent* query $q$ should be evicted from a pure LRU cache because the distance between two successive requests for $q$ (in terms of distinct queries) is greater than the size of the cache itself. To this end, we extracted the 128,000 most popular queries present in the log, and we measured the number of distinct queries received by Alta Vista in the interval between one submission and the next submission of each of these popular queries. Figure 11 plots the cumulative number of occurrences of each distance, measured as the number of distinct queries received by Alta Vista in the interval between two successive submissions of each frequent query. As expected, many popular queries were resubmitted after long intervals. These queries would surely cause cache misses in an LRU cache having a size smaller than the distance plotted on the $x$ axis. In particular, we measured that 698,852 repetitions of the most popular queries occurred at distances less than 128,000, while 164,813 occurred at distances greater than 128,000.[2] Thus, the adoption of an LRU cache of 128,000 blocks should surely result in 164,813 misses for these most popular queries. The distribution of the distances plotted in Figure 11, thus explains the higher hit ratio achieved by our hybrid caching policy with respect to a purely dynamic policy like LRU which is based on recency of references only.

[2]It is worth noting that due to the log-scale, in Figure 11, the black area corresponding to $x \geq$ 128,000 appears to be very small. It is instead about a quarter of the one corresponding to $x <$128,000.

## 6. CONCURRENCY ISSUES

We designed our caching system to allow many concurrent threads to efficiently access its blocks. This is motivated by the fact that a WSE has to process many user queries concurrently, and this is usually achieved by having each query processed by a distinct thread. The methods exported by our caching system are thus thread-safe, and also ensure mutual exclusion whenever that 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 look up 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 speedup 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, a cache hit means the list pointers must be modified in order to sort the cache entries according to the replacement policy adopted.

Our caching system can easily be 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 multithreaded 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 called the thread-safe methods exported by our cache implementation. The WSE core query service, which is invoked to resolve the queries that cause misses, was instead simulated by putting the thread which manages the query to sleep. 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 latency of the WSE core in resolving the query. In these tests we did not activate prefetching; otherwise the value of $\delta$ should be changed accordingly.

The above assumptions are oversimplified, and thus our testing environment cannot be considered very realistic. On the other hand, a WSE is a complex, highly nonlinear environment, and a distributed WSE even more. A simulation approach cannot capture all its aspects and cannot approximate real costs (see Moffat and Zobel [2004] for an in-depth discussion about the difficulties of performance measurement in this field). Thus, the results of the tests we conducted to evaluate the throughput of our implementation have to be considered only a tentative measure of the benefits deriving from the presence of a read-only part of the cache accessed concurrently by many threads. Performance figures obtained cannot be used to draw definite conclusions.

Figure 12 reports the results of some of the 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

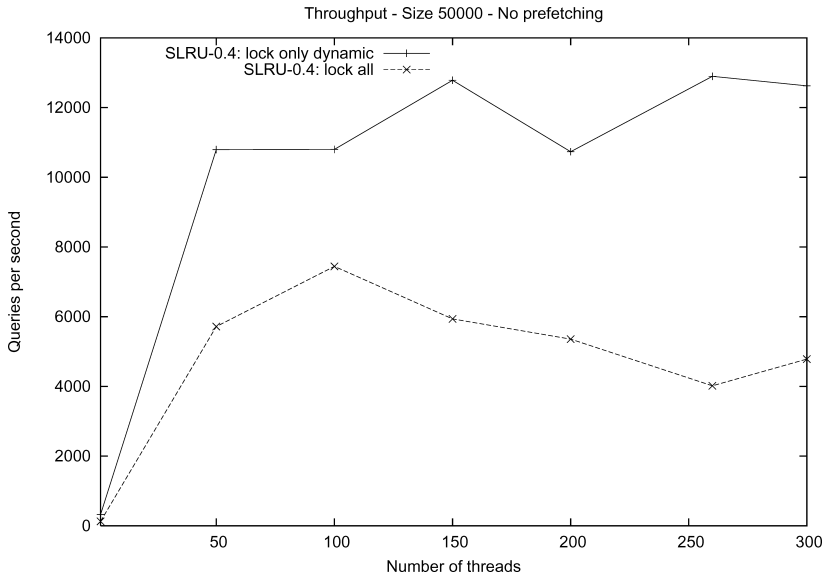



Fig. 12. Throughput of our caching system (in number of queries served per second) for $f_{static} = 0.6$ as a function of the number of concurrent threads used and different locking policies.

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 in these tests the presence of the static set, which does not need to be accessed in a critical section, permits the approximate doubling of the number of queries served per second. Moreover, the caching system 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 article, we have presented SDC, a new strategy 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 real query logs. 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. In all the cases, our strategy remarkably outperformed either purely static or dynamic caching policies. 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 an adaptive prefetching strategy which constitutes a good tradeoff between the maximization of the benefits of prefetching and the reduction of the additional load on the WSE. Moreover, we analyzed the reasons justifying the greater performances of our hybrid policy with respect to purely static and dynamic ones, and studied how the statistical data about query frequency, which are used to fill the static portion of the cache, gets older as time progresses. Since in our tests the degradation was shown to be slow, our hybrid caching approach was viable and effective. Finally, we tentatively evaluated the cost and scalability of our cache implementation when executed in a multithreaded environment. The tests demonstrated that our SDC implementation is very efficient: measured cache hit and miss times were very low. Moreover, the presence of the read-only static cache allowed the number of synchronizations between multiple threads concurrently accessing the cache to be remarkably reduced.

Our work suggests some further research directions. We know that a distributed WSE is is a complex, highly nonlinear environment, where actual query processing times are characterized by high variance and may be very hard to predict. Since differences in query execution times can be of one or more orders of magnitude, it would be interesting to try to exploit the knowledge of such costs in the caching policy. By preferably caching queries that require large execution times, we could in fact enhance the global throughput of the WSE, although the hit ratios achieved can be expected to decrease. In SDC this could be done in two different ways. On the one hand, the frequency-based policy used to fill the static set could be modified to consider a weighted contribution of query execution cost so that frequent queries which are also expensive to process are preferred. On the other hand, the query processing cost could be also exploited by the replacement policy adopted for managing page eviction from the dynamic set, which can give a higher priority to cached queries which are expensive to process.

Finally, the presence in the query stream of time-of-day patterns (see Section 5.3) merits further investigation. We observed that groups of frequent queries present in the *Alta Vista* log are characterized by repetition rates that vary regularly across the day. We expect that similar time-of-day patterns are not a peculiarity of the *Alta Vista* log. The knowledge of such patterns can be used to construct different instances of the static sets which SDC can schedule and use during the day in order to better fulfill these patterns.

## ACKNOWLEDGMENTS

We acknowledge Ideare S.p.A. for providing us with the *Tiscali* query log, and Ronny Lempel for useful discussions and for providing us with the *Alta Vista* query log. Finally, we are grateful to the Associate Editor and the anonymous referees of this journal, who allowed us to improve this article with their helpful comments.

## 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.), 22–28.

BEITZEL, S. M., JENSEN, E. C., CHOWDHURY, A., GROSSMAN, D., AND FRIEDER, O. 2004. Hourly analysis of a very large topically categorized web query log. In *SIGIR '04: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, New York, NY, 321–328.

HÖLSCHER, C. 1998. How Internet experts search for information on the Web. In *Proceedings of WebNet 98—World Conference on the WWW and Internet & Intranet* (Orlando, FL, Nov. 7–12).

JOHNSON, T. AND SHASHA, D. 1994. 2Q: A low overhead high performance buffer management replacement algorithm. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*. Morgan Kaufmann, San Francisco, CA, 439–450.

KAREDLA, R., LOVE, J., AND WHERRY, B. 1994. Caching strategies to improve disk system performance. *IEEE Comput. 27*, 3, 38–46.

LEMPEL, R. AND MORAN, S. 2003. Predictive caching and prefetching of query results in search engines. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*. ACM Press, New York, NY, 19–28.

LONG, X. AND SUEL, T. 2005. Three-level caching for efficient query processing in large Web search engines. In *WWW '05: Proceedings of the 14th International Conference on World Wide Web*. ACM Press, New York, NY, 257–266.

MARKATOS, E. P. 2000. On caching search engine results. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*. Go online to `http://www.iwcw.org/2000/Proceedings/proceedings.html`.

MARKATOS, E. P. 2001. On caching search engine results. *Comput. Commun. 24*, 2, 137–143.

MOFFAT, A. AND ZOBEL, J. 2004. What does it mean to "measure performance"? In *Proceedings of the International Conference on Web Informations Systems*, X. Zhou, S. Su, M. P. Papazoglou, M. E. Owlowska, and K. Jeffrey, Eds. Lecture Notes in Computer Science, vol. 3306. Springer, Berlin, Germany, 1–12.

O'NEIL, E. J., O'NEIL, P. E., AND WEIKUM, G. 1993. The LRU–KS page replacement algorithm for database disk buffering. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. ACM Press, New York, NY, 297–306.

ORLANDO, S., PEREGO, R., AND SILVESTRI, F. 2001. Design of a parallel and distributed Web search engine. In *ParCo2001: Proceedings of the International Conference Parallel Computing: Advances and Current Issues*. Imperial College Press, London, U.K., 197–204.

PODLIPNIG, S. AND BOSZORMENYI, L. 2003. A survey of web cache replacement strategies. *ACM Comput. Surv. 35*, 4, 374–398.

RAGHAVAN, V. V. AND SEVER, H. 1995. On the reuse of past optimal queries. In *SIGIR '95: Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, New York, NY, 344–350.

ROBINSON, J. T. AND DEVARAKONDA, M. V. 1990. Data cache management using frequency-based replacement. In *SIGMETRICS '90: Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. ACM Press, New York, NY, 134–142.

SARAIVA, P. C., DE MOURA, E. S., ZIVIANI, N., MEIRA, W., FONSECA, R., AND RIBERIO-NETO, B. 2001. Rank-preserving two-level caching for scalable search engines. In *SIGIR '01: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, New York, NY, 51–58.

SILVERSTEIN, C., MARAIS, H., HENZINGER, M., AND MORICZ, M. 1999. Analysis of a very large Web search engine query log. *SIGIR Forum 33*, 1, 6–12.

SILVESTRI, F. 2004. High performance issues in Web search engines: Algorithms and techniques. Ph.D. dissertation. Università degli Studi di Pisa—Facoltà di Informatica, Pisa, Italy.

SPINK, A., WOLFRAM, D., JANSEN, M. B. J., AND SARACEVIC, T. 2001. Searching the Web: The public and their queries. *J. Amer. Soc. Inform. Sci. Tech. 52*, 3, 226–234.

WITTEN, I. H., MOFFAT, A., AND BELL, T. C. 1999. *Managing Gigabytes—Compressing and Indexing Documents and Images*, 2nd ed. Morgan Kaufmann, San Francisco, CA.

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*.

Received July 2004; revised June 2005, October 2005; accepted October 2005