Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

Predictive Caching and Prefetching of Query Results in Search Engines

Ronny Lempel Shlomo Moran

Department of Computer Science

The Technion,

Haifa 32000, Israel

email: {rlempel,moran}@cs.technion.ac.il

ABSTRACT

We study the caching of query result pages in Web search engines. Popular search engines receive millions of queries per day, and efficient policies for caching query results may enable them to lower their response time and reduce their hardware requirements. We present *PDC* (probability driven cache), a novel scheme tailored for caching search results, that is based on a probabilistic model of search engine users. We then use a trace of over seven million queries submitted to the search engine AltaVista to evaluate PDC, as well as traditional LRU and SLRU based caching schemes. The trace driven simulations show that PDC outperforms the other policies. We also examine the prefetching of search results, and demonstrate that prefetching can increase cache hit ratios by 50% for large caches, and can double the hit ratios of small caches. When integrating prefetching into PDC, we attain hit ratios of over 0.53.

1 Introduction

Popular search engines receive millions of queries per day on any and every walk of life. While these queries are submitted by millions of unrelated users, studies have shown that a small set of popular queries accounts for a significant fraction of the query stream. Furthermore, search engines

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

may also anticipate user requests, since users often ask for more than one page of results per query. It is therefore commonly believed that all major search engines perform some sort of search result caching and prefetching. An engine that answers many queries from a cache instead of processing them through its index, can lower its response time and reduce its hardware requirements.

1.1 Search Engine Users

Jónsson et al. [7], in their work on buffering inverted lists for query evaluations, noted that knowledge of the access patterns of the retrieval algorithm to the buffers can be tapped for devising effective buffer replacement schemes. By analogy, understanding the access patterns of search engine users to query results can aid the task of caching search results.

Users submit queries to search engines. From a user's point of view, an engine answers each query with a linked set of ranked result pages, typically with 10 results per page. All users browse the first page of results, and some users scan additional result pages, usually in the natural order in which those pages are presented.

Three studies have analyzed the manner in which users query search engines and view result pages: a study by Jansen et al. [6], based on 51,473 queries submitted to the search engine *Excite*[1]; a study by Markatos [11], based on about a million queries submitted to Excite on a single day in 1997; and a study by Silverstein et al. [14], based on about a billion queries submitted to the search engine *AltaVista*[2] over a period of 43 days in 1998. Two findings that are particularly relevant to this work concern the number of result pages that users view per query, and the distribution of query popularities. Regarding the former, the three studies agree that at least 58% of the users view only the first page of results (the top-10 results), at least 15% of the users view more than one page of results, and that no more than 12% of users browse through more than 3 result pages. Regarding query popularities, it was found that the number of distinct information needs of users is very large. Silverstein et. al report that 63.7% of the search phrases appear only once in the billion query log. These phrases were submitted just once in a period of six weeks. However, popular queries are repeated many times: the 25 most popular queries found in the AltaVista log account

[1] http://www.excite.com/

[2] http://www.altavista.com/

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

for 1.5% of the submissions. The findings of Markatos are consistent with the later figure - the 25 most popular queries in the Excite log account for 1.23% of the submissions. Markatos also found that many successive submissions of the same query appear in close proximity (are separated by a small number of other queries in the query log).

1.2 Caching and Prefetching of Search Results

Caching of results was noted in Brin and Page's description of the prototype of the search engine *Google*[3] as an important optimization technique of search engines [2]. Markatos [11] used the million-query Excite log to drive simulations of query result caches using four replacement policies - LRU (Least Recently Used) and three variations. He demonstrated that warm, large caches of search results can attain hit ratios of close to 30%.

Saraiva et al. [13] proposed a two-level caching scheme that combines caching query results and inverted lists. The replacement strategy they adopted for the query result cache was LRU. They experimented with logged query streams, testing their approach against a system with no caches. Overall, their combined caching strategy increased the throughput of the system by a factor of three, while preserving the response time per query.

In addition to storing results that were requested by users in the cache, search engines may also *prefetch* results that they predict to be requested shortly. An immediate example is prefetching the second page of results whenever a new query is submitted by a user. Since studies of search engines' query logs [14, 6] indicate that the second page of results is requested shortly after a new query is submitted in at least 15% of cases, search engines may prepare and cache two (or more) result pages per query. The prefetching of search results was examined in [10], albeit from a different angle: the objective was to minimize the computational cost of processing queries rather than to maximize the hit ratio of the results cache.

The above studies, as well as our work, focus on the caching of search results *inside* the search engine. The prefetching discussed above deals with how the engine itself can prefetch results from its index to its own cache. The caching or prefetching of search results outside the search engine is not considered here. In particular, we are not concerned with how general Web caches and

[3] http://www.google.com/

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

proxy servers should handle search results. Web caching and prefetching is an area with a wealth of research; see, for example, [16] for a method that allows proxy servers to predict future user requests by analyzing frequent request sequences found in the servers' logs, [15] for a discussion of proxy cache replacement policies that keep a history record for each cached object, and [4] for a proposed scheme that supports the caching of dynamic content (such as search results) at Web proxy servers. In both [17] and [9], Web caching and document prefetching is integrated using (different) prediction models of the aggregate behavior of users. Section 3.2.1 has more details on these two works.

1.3 This work

This work examines the performance of several cache replacement policies on a stream of over seven million real-life queries, submitted to the search engine AltaVista. We integrate result prefetching into the schemes, and find that for all schemes, prefetching substantially improves the caches' performance. When comparing each scheme with prefetching to the naive version that only fetches the requested result pages, we find that prefetching improves the hit ratios by more than what is achieved by a fourfold increase in the size of the cache.

Another contribution of this paper is the introduction of a novel cache replacement policy that is tailored to the special characteristics of the query result cache. The policy, which is termed PDC (Probability Driven Cache), is based on a probabilistic model of search engine users' browsing sessions. Roughly speaking, PDC prioritizes the cached pages based on the number of users who are *currently* browsing *higher ranking* result pages of the *same* search query. We show that PDC consistently outperforms LRU and SLRU (Segmented LRU) based replacement policies, and attains hit ratios exceeding 53% in large caches.

Throughout this paper, a *query* will refer to an ordered pair (t, k) where t is the *topic* of the query (the search phrase that was entered by the user), and $k \geq 1$ is the number of result page requested. For example, the query $(t, 2)$ will denote the second page of results (which typically contains results $11 - 20$) for the topic t.

The rest of this work is organized as follows. Section 2 describes the query log on which we conducted our experiments. Section 3 presents the different caching schemes that this work

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

examined, and in particular, defines PDC and the model on which it is based. Section 4 reports the results of the trace-driven simulations with which we evaluated the various caching schemes. Concluding remarks and suggestions for future research are brought in Section 5.

2 The Query Log

The trace contained 7175151 keyword-driven search queries, submitted to the search engine AltaVista during the summer of 2001. AltaVista returns search results in batches whose size is a multiple of 10. For $r \geq 1$, the results whose rank is $10(r-1)+1, \ldots, 10r$ will be referred to as the r'th result page. A query that asks for a batch of $10k$ results will be thought of as asking for k *logical* result pages (although, in practice, a single large result page will be returned). Each query can be seen as a quadruple $q = (\tau, t, f, \ell)$ as follows:

- τ is a time stamp (date and time) of q's submission.
- t is the topic of the query, identified by the search phrase that was entered by the user.
- f and ℓ define the range of result pages requested ($f \leq \ell$). In other words, the query requests the $10(f-\ell+1)$ results ranking in places $10(f-1)+1, \ldots, 10\ell$ for the topic t. AltaVista allowed users to request up to 100 results (10 logical result pages) in a single query.

In order to ease our simulations, we discarded from the trace all queries that requested result pages beyond result page number 32 (results whose rank is 321 or worse). There were 14961 such queries, leaving us with a trace of 7160190 queries. These queries contained requests for 4496503 distinct result pages, belonging to 2657410 distinct topics.

The vast majority of queries (97.7%) requested 10 results, in a standard single page of results (in most cases, $f = \ell$). However, some queries requested larger batches of results, amounting to as many as 10 logical result pages. Table 1 summarizes the number of logical result pages requested by the various queries. Table 1 implies that the total number of requested logical pages, also termed as *page views*, is 7549873.

As discussed in the Introduction, most of the requests were for the first page of results (the top-10 results). In the trace considered here, 4797186 of the 7549873 views (63.5%) were of first

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

No. result pages	1	2	3	4	5	6	7, 8, 9	10
No. queries	6998473	31477	84795	939	42851	125	0	1530

Table 1: Number of queries requesting different bulks of logical result pages

pages. The 885601 views of second pages accounted for 11.7% of the views. Figure 1 brings three histograms of the number of views for result pages $2-32$. The three ranges are required since the variance in the magnitude of the numbers is quite large. Observe that higher ranking result pages are generally viewed more than lower ranking pages. This is due to the fact that the users who view more than one result page, usually browse those pages in the natural order. This sequential browsing behavior allows for predictive prefetching of result pages, as will be shown in Section 4.

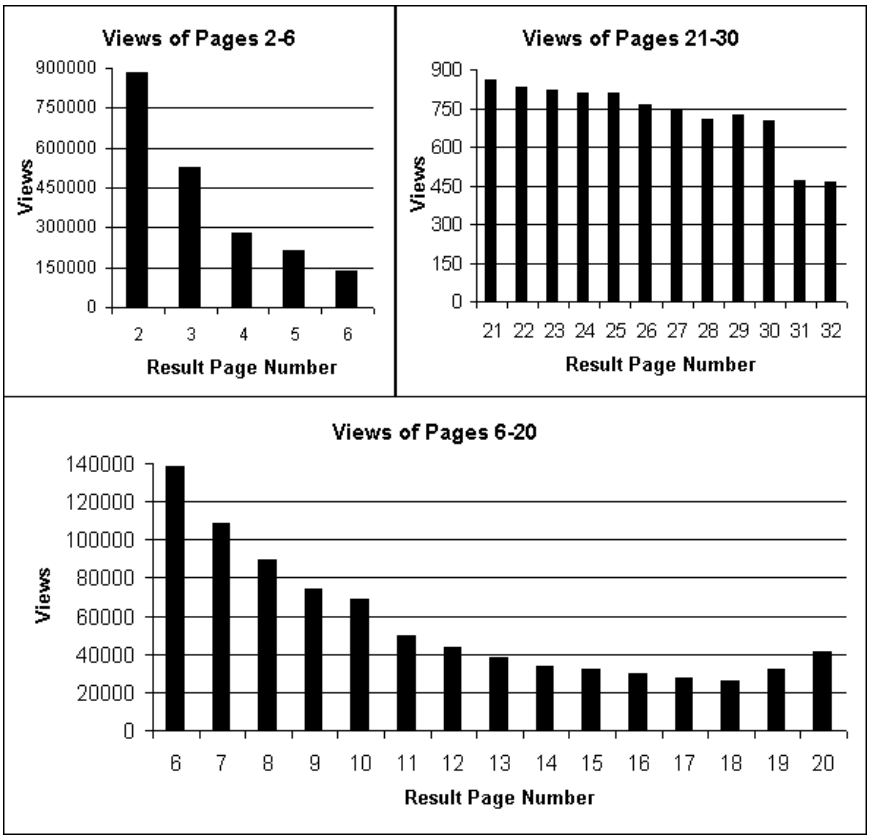

Figure 1: Views of result pages $2-32$

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

Figure 1 shows a sharp drop between the views of result pages 20 and 21. This drop is explained by the manner in which AltaVista answered user queries during the summer of 2001. "Regular" users who submitted queries were allowed to browse through 200 returned results, in 20 result pages containing 10 results each. Access to more than 200 results (for standard search queries) was restricted by the engine.

Having discussed the distribution of views per result page number, we examine a related statistic - the distribution of the number of distinct result pages viewed per topic. We term this as the *population* of the topic. In almost 78% of topics (2069691 of 2657410), only a single page was viewed (usually the first page of results), and so the vast majority of topics have a population of 1. Figure 2 shows the rest of the distribution, divided into three ranges due to the variance in the magnitude of the numbers. Note the unusual strength of topics with populations of 10, 15 and 20.

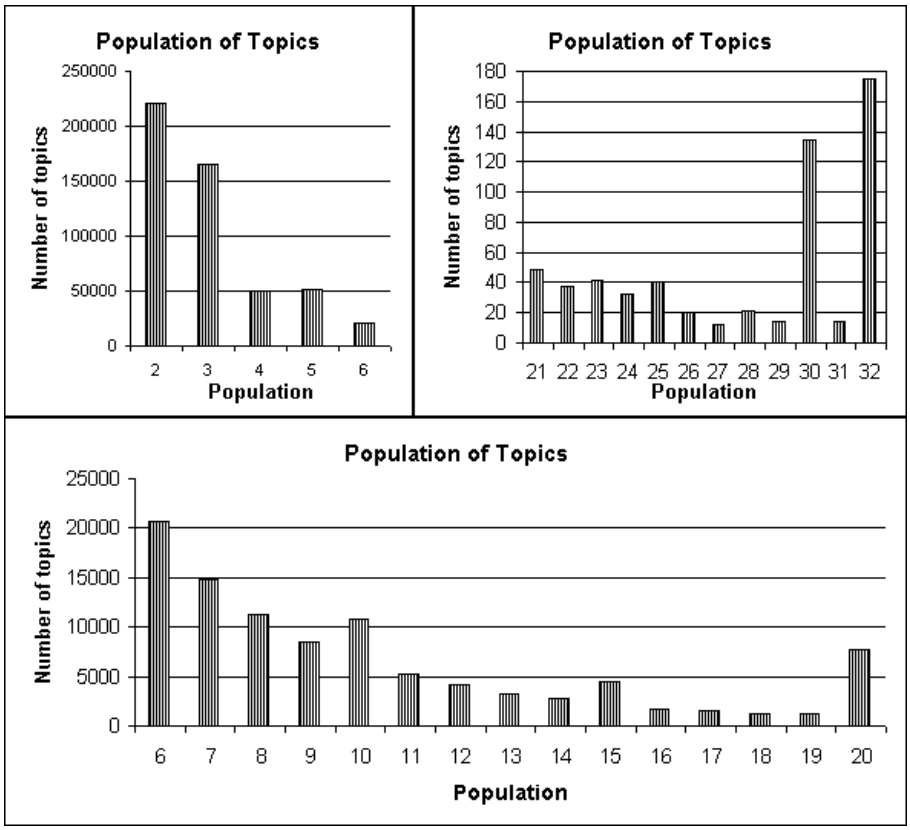

Figure 2: Population of Topics (number of pages requested per topic)

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

From topic populations we turn to topic (and page) popularities. Obviously, different topics (search phrases) and result pages are requested at different rates. Some topics are extremely popular, while the majority of topics are only queried once. As mentioned earlier, the log contained queries for 2657410 distinct topics. 1792104 (over 67%) of those topics were requested just once, in a single query (the corresponding figure in [14] was 63.7%). The most popular topic was queried 31546 times. In general, the popularity of topics follows a power law distribution, as shown in Figure 3. The plot conforms to the power-law for all but the most popular topics, which are over-represented in the log. A similar phenomenon is observed when counting the number of requests for individual result pages. 48% of the result pages are only requested once. However, the 50 most requested pages account for almost 2% of the total number of page views (150131 of 7549873). Again, the distribution follows a power law for all but the most oft-requested result pages (also in Figure 3)[4].

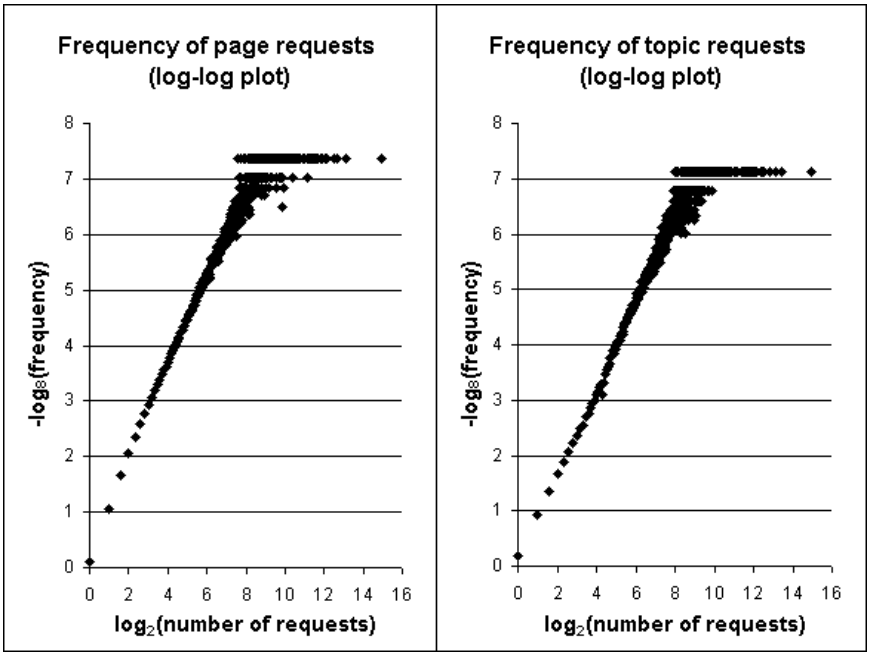

Figure 3: Popularity of Topics/Pages (log-log plot)

[4]The power law for page[topic] popularities implies that the probability of a page[topic] being requested x times is proportional to x^{-c}. In this log, c is approximately 2.8 for page popularities and about 2.4 for topic popularities.

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

The log contained exactly 200 result pages that were requested more than $2^9 = 512$ times. As seen in Figure 3, these pages do not obey the above power law. However, they have a distinctive behavior as well: the number of requests for these pages conforms to a Zipf distribution, as Figure 4 shows [5]. This is consistent with the results of Markatos [11], who plotted the number of requests for the 1000 most popular queries in the Excite log, and found that the plot conforms to a Zipf distribution. Saraiva et. al [13], who examined 100000 queries submitted to a Brazilian search engine, report that the popularities of all queries follow a Zipf distribution.

Studies of Web server logs have revealed that requests for static Web pages follow a power law distribution [1]. The above cited works and our findings show that this aggregate behavior of users carries over to the browsing of dynamic content, where the users define the query freely (instead of selecting a resource from a fixed "menu" provided by the server). We also note that the complex, distributed social process of creating hyperlinked Web content gives rise to power law distributions of inlinks to and outlinks from pages [3]. See [12] for a general survey of power law, Pareto, Zipf and lognormal distributions.

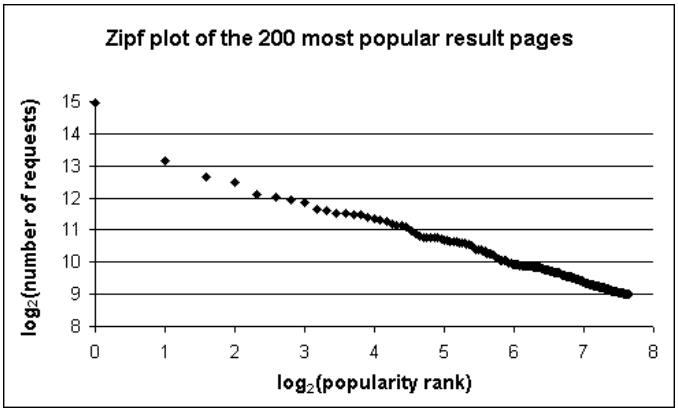

Figure 4: Zipfian behavior of the 200 most popular result pages

[5] The number of requests for the r'th most popular page is proportional to r^{-c}, for $c \approx 0.67$.

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

3 Caching and Prefetching Schemes

3.1 Fetch units and result prefetching

In many search engine architectures, the computations required during query execution are not greatly affected by the number of results that are to be prepared, as long as that number is relatively small. In particular, it may be that for typical queries, the work required to fetch several dozen results is just marginally larger than the work required for fetching 10 results. Since fetching more results than requested may be relatively cheap, the dilemma is whether storing the extra results in the cache (at the expense of evicting previously stored results) is worthwhile. Roughly speaking, result prefetching is profitable if, with high enough probability, those results will be requested shortly - while they are still cached and before the evicted results are requested again. One aspect of result prefetching was analyzed in [10], where the computations required for query executions (and not cache hit ratios) were optimized.

In our simulations, all caching schemes will fetch results in bulks whose size is a multiple of k, a basic *fetch unit*. Formally, let q be a query requesting result pages f through ℓ for some topic. Let α, β be the first and last uncached pages in that range, respectively ($f \leq \alpha \leq \beta \leq \ell$). A k-fetch policy will fetch pages $\alpha, \alpha+1, \ldots, \alpha+mk-1$, where m is the smallest integer such that $\alpha+mk-1 \geq \beta$. Recall that over 97% of the queries request a single result page ($f = \ell$). When such a query causes a cache fault, a k-fetch policy effectively fetches the requested page and prefetches the next $k-1$ result pages of the same topic. When $k = 1$, fetching is performed solely *on demand*.

For every fetch unit k, we can estimate theoretical upper bounds on the hit ratio attainable by any cache replacement policy on our specific query log. Consider a cache of infinite size, where evictions are never necessary. For each topic t, we examine P_t, the subset of the 32 potential result pages that were actually requested in the log. We then cover P_t with the minimal number of fetch units possible. This number, denoted by $f_k(t)$, counts how many k-fetch query executions are required for fetching P_t. The sum $\sum_t f_k(t)$ is a close approximation to the minimal number of faults that any policy whose fetch unit is k will have on our query log. Table 2 brings these theoretical bounds for several fetch units. Note the dramatic improvement in the theoretic hit ratio as the fetch unit grows from 1 to 3.

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

Fetch unit	No. Fetches	Hit Ratio	Fetch unit	No. Fetches	Hit Ratio
1	4496503	0.372	5	2861390	0.600
2	3473175	0.515	10	2723683	0.620
3	3099268	0.567	20	2658159	0.629
4	2964553	0.586	32	2657410	0.629

Table 2: Upper bounds on hit ratios for different values of the fetch unit

3.2 Caching Replacement Policies

We experimented with five cache replacement policies. The first four are adaptations of the well-known LRU and SLRU policies [8]. The complexity of treating a query is $\mathcal{O}(1)$ for each of those policies. The fifth policy, which we call *Probability Driven Cache (PDC)*, is a novel approach tailored for the task of caching query result pages. It is more complex, requiring $\Theta(\log(\text{size of the cache}))$ operations per query. The following describes the data structures that are used in each scheme, and the manner in which each scheme handles queries. For this we define, for a query q that requests the set of pages $P(q)$, two disjoint sets of pages $C(q)$ and $F(q)$:

1. $C(q) \subseteq P(q)$ is the subset of the requested pages that are cached when q is submitted.
2. Let $F'(q)$ denote the set of pages that are fetched as a consequence of q, as explained in Section 3.1. $F(q)$ is the subset of the *uncached* pages of $F'(q)$. [6]

Page LRU (PLRU) This is a straightforward adaptation of the Least Recently Used (LRU) policy. We allocate a *page queue* that can accommodate a certain number of result pages. For every query q, the pages of $C(q)$ are moved back to the tail of the queue. They are joined there by the pages of $F(q)$, which are added to the tail of the queue. Once the queue is full, cached pages are evicted from the head of the queue. Thus, the tail of the queue holds the most recently requested (and prefetched) pages, while its head holds the least recently requested pages. PLRU with a fetch unit of 1 was evaluated in [11], and attained hit ratios of around 30% (for warm, large caches).

[6] As a byproduct, the cached entries of the pages of $F'(q) \setminus F(q)$ are refreshed.

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

Page SLRU (PSLRU) The SLRU policy maintains two LRU segments, a *protected* segment and a *probationary* segment. The pages of $F(q)$ are inserted into the (tail of the) probationary segment. The pages of $C(q)$ are transfered to the tail of the protected segment. Pages that are evicted from the protected segment remain cached - they are demoted to the tail of the probationary segment. Pages are removed from the cache only when they are evicted from the probationary segment. It follows that pages in the protected segment were requested at least twice since they were last fetched. PSLRU with a fetch unit of 1 was also evaluated in [11], where it consistently outperformed PLRU (in large caches, however, the difference was very small).

Topic LRU (TLRU) This policy is a variation on the PLRU scheme. Let $t(q)$ denote the topic of the query q. TLRU performs two actions for every query q: (1) the pages of $F(q)$ are inserted into the page queue, and (2) any cached result page of $t(q)$ is moved to the tail of the queue. Effectively, each topic's pages will always reside contiguously in the queue, with the blocks of different topics ordered by the LRU policy.

Topic SLRU (TSLRU) This policy is a variation on the PSLRU scheme. It performs two actions for every query q (whose topic is $t(q)$): (1) the pages of $F(q)$ are inserted into the probationary queue, and (2) any cached result page of $t(q)$ is moved to the tail of the protected queue.

3.2.1 Probability Driven Cache (PDC)

Cache replacement policies attempt to keep pages that have a high probability of being requested in the near future, cached. The PDC scheme is based on a model of search engine users that concretely defines the two vague notions of "probability of being requested" and "near future".

The model behind PDC Users submit queries to a search engine in an asynchronous manner. Every user u issues a series of $\{q_i^u = (\tau_i^u, t_i^u, f_i^u, \ell_i^u)\}_{i \geq 1}$ queries, where τ_i^u is the time of submission of q_i^u, t_i^u is the topic of the query, and $f_i^u \leq \ell_i^u$ define the range of result pages requested. Consider q_i^u and q_{i+1}^u, two successive queries issued by user u. In this model, q_{i+1}^u may either *follow-up* on q_i^u, or start a *new search session* of u that is unrelated to the previous query:

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

1. q^u_{i+1} is a follow-up on q^u_i (denoted $q^u_i \rightarrow q^u_{i+1}$) if both are on the same topic, q^u_{i+1} asks for the result pages that immediately follow those requested in q^u_i, and q^u_{i+1} is submitted no more that W time units after q_i. Formally, $t^u_{i+1} = t^u_i$, $f^u_{i+1} = \ell^u_i + 1$ and $\tau^u_{i+1} \leq \tau^u_i + W$. We say that $q^u_i, \ldots, q^u_j (i \leq j)$ constitute a *search session* whenever $q^u_k \rightarrow q^u_{k+1}$ for all $i \leq k < j$, while $q^u_{i-1} \not\rightarrow q^u_i$ and $q^u_j \not\rightarrow q^u_{j+1}$.

2. q^u_{i+1} starts a new search session whenever $f^u_{i+1} = 1$. [7]

This model of search behavior limits the "attention span" of search engine users to W time units: users do not submit follow-up queries after being inactive for W time units. Long inactivity periods indicate that the user has lost interest in the previous search session, and will start a new session with the next query. Following are several implications of this model:

- The result pages viewed in every search session are pages $(t,1), \ldots, (t,m)$ for some topic t and $m \geq 1$.

- At any given moment τ, every user has at most one query that will potentially be followed upon. Formally, let U denote the set of users, and let q^u_τ denote the most recent query submitted by user $u \in U$ prior to time τ (q^u_τ may be *nil* if u has not submitted queries prior to τ). The set of queries that will potentially be followed upon is defined by

$$Q \stackrel{\triangle}{=} \{q^u_\tau, u \in U \ : \ q^u_\tau \text{ was submitted after } \tau - W\} \tag{1}$$

The model assumes that there are topic and user independent probabilities $s_m, m \geq 1$ such that s_m is the probability of a search session requesting exactly m result pages. Furthermore, the model assumes that it is familiar with these probabilities.

For a query $q \in Q$, let $t(q)$ denote the query's topic and let $\ell(q)$ denote the last result page requested in q. For every result page (t,m), we can now calculate $\mathcal{P}_Q(t,m)$, the probability that (t,m) will be requested as a follow-up to *at least one* of the queries in Q:

$$\mathcal{P}_Q(t,m) = 1 - \prod_{q \in Q} (1 - P[(t,m) \text{ will follow-up on } q]) = 1 - \prod_{q \in Q : t(q) = t, \ell(q) < m} (1 - P[m|\ell(q)]) \tag{2}$$

[7] We assume that the first query of every user u, q^u_1, requests the top result page of some topic.

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

$$\text{where } P[m|\ell(q)] \;=\; \frac{\sum_{i \geq m} s_i}{\sum_{j \geq \ell(q)} s_j}$$

$P[m|\ell]$ is the probability that a session will request result page m, given that the last result page requested so far was page ℓ. $\mathcal{P}_Q(t,m)$ depends on the number of users who are currently searching for topic t, as well as on the last t-page requested by every such user. $\mathcal{P}_Q(t,m)$ will be large if many users have recently (within W time units) requested t-pages whose number is close to (but smaller than) m. Note that for all t, $\mathcal{P}_Q(t,1) = 0$; the model does not predict the topics that will be the focus of future search sessions. PDC prioritizes cached $(t,1)$ pages by a different mechanism than the $\mathcal{P}_Q(t,m)$ probabilities.

Kraiss and Weikum also mention setting the priority of a cached entry by the probability of at least one request for the entry within a certain time horizon [9]. Their model for predicting future requests is based on continuous-time Markov chains, and includes the modeling of new session arrivals and current session terminations. However, the main prioritization scheme that they suggest, and which is best supported by their model, is based on the *expected* number of requests to each cached entry (within a given time frame). Prioritizing according to the probability of at least one visit is quite complex in their model, prompting them to suggest a calculation which approximates these probabilities. As the model behind PDC is simpler than the more general model of [9], the calculations it involves are also significantly less expensive.

Implementing PDC The PDC scheme is based on the model described above. PDC attempts to prioritize its cached pages using the probabilities calculated in Equation 2. However, since these probabilities are zero for all $(t,1)$ pages, PDC maintains two separate buffers: (1) an SLRU buffer for caching $(t,1)$ pages, and (2) a priority queue PQ for prioritizing the cached $(t, m > 1)$ pages according to the probabilities of Equation 2. The relative sizes of the SLRU and PQ buffers are subject to optimization, as will be discussed in Section 4.2. For the time being, let C_{PQ} denote the size (capacity) of PQ. In order to implement PQ, PDC must set the probabilities $s_i,\ i \geq 1$ and keep track of the set of queries Q, defined in Equation 1:

- Every search engine can set the probabilities $s_i,\ i \geq 1$ based on the characteristics of its log. In our implementation, we approximated s_i by the proportion of views of i'th result pages in

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

the first million queries. [8] In light of this, the PDC simulations whose results are reported in Section 4.2 skip these queries, and are driven by the remaining 6160190 queries of the log.

- PDC tracks the set Q by maintaining a query window QW, that holds a subset of the queries submitted during the last W time units. The exact subset that is kept in QW will be discussed shortly. For every kept query $q = (\tau, t, f, \ell)$, its time τ and last requested page (t, ℓ) are saved.

Each query $q = (\tau, t, f, \ell)$ is processed in PDC by the following four steps:

1. q is inserted into QW, and queries submitted before $\tau - W$ are removed from QW. If there is a query q' in QW such that the last page requested by q' is $(t, f-1)$, the least recent such query is also removed from QW. This is the heuristic by which we associate follow-up queries with their predecessors, since the queries in our log are anonymous.

2. Let T denote the set of topics whose corresponding set of QW queries has changed (t belongs to T, and other topics may have had queries removed from QW). The priorities of all T-pages in PQ are updated according to Equation 2, with the set of queries in QW assuming the role of the query set Q.

3. If $f = 1$ and page $(t, 1)$ is not cached, $(t, 1)$ is inserted at the tail of the probationary segment of the SLRU. If $(t, 1)$ is already cached, it is moved to the tail of the protected segment of the SLRU.

4. Let $(t, m), 1 < m \leq \ell$ be a page requested by q that is not cached. Its priority is calculated in light of the window QW, and if it merits so, it is kept in PQ (causing perhaps an eviction of a lower priority page).

In order to implement the above procedure efficiently, PDC maintains a *topic table*, where every topic (1) links to its cached pages in PQ, (2) points to its top result page in the SLRU, and (3) keeps track of the QW entries associated with it. When the number of different result pages that may be cached per topic is bounded by a constant, and when PQ is implemented by a heap, the

[8] The behavior of the number of views per result page number in the entire log was discussed in Section 2 and Figure 1.

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

amortized complexity of the above procedure is $\Theta(\log C_{PQ})$ per query [9]. See [5] for discussions of the heap data structure and of amortized analysis.

As noted above, the cache in PDC is comprised of two separate buffers of fixed size, an SLRU for $(t, 1)$ pages and a priority queue for all other pages. Two buffers are also maintained in [17]; there, the *cache* buffer is dedicated to holding pages that were actually requested, while the *prefetch* buffer contains entries that the system predicts to be requested shortly. Pages from the prefetch buffer migrate to the cache buffer if they are indeed requested as predicted.

4 Results

This section reports the results of our experiments with the various caching schemes. Each scheme was tested with respect to a range of cache sizes, fetch units and other applicable parameters. Subsection 4.1 concretely defines the terms *cache size* and *hit ratio* in the context of this work. Subsection 4.2 discusses the schemes separately, starting with the LRU-based schemes, moving to the more complex SLRU-based schemes, and ending with the PDC scheme. A cross-scheme comparison is brought in Subsection 4.3.

4.1 Interpretation of Reported Statistics

Cache sizes Cache sizes in this work are measured by *page entries*, which are abstract units of storage that hold the information associated with a cached result page of 10 search results. Markatos[11] estimates that such information requires about 4 kilobytes of storage. Since the exact amount of required memory depends on the nature of the information stored and its encoding, this figure may vary from engine to engine. However, we assume that for every specific engine, the memory required does not vary widely from one result page to another, and so we can regard the page entry as a standard memory unit. Note that we ignore the memory required for bookkeeping in each of the schemes. In the four LRU-based policies, this overhead is negligible. In PDC, the query window QW requires several bytes of storage per kept query. We ignore this, and only

[9] Amortized analysis is required since the number of topics affected by the QW updates while treating a single query, may vary. Other implementations may choose to have QW hold a fixed number of recent queries without considering the time frame. Such implementations will achieve a non-amortized complexity of $\Theta(\log C_{PQ})$.

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

consider the capacity of the SLRU and PQ buffers of PDC.

Hits and Faults As shown in Table 1, the queries in our trace file may request several (up to 10) result pages per query. Thus, it is possible for a query to be partially answered by the cache - some of the requested result pages may be cached, while other pages might not be cached. In our reported results, a query counts as a cache *hit* only if it can be *fully* answered from the cache. Whenever one or more of the requested pages is not cached, the query counts as a cache *fault*, since satisfying the query requires processing it through the index of the engine. It follows that each query causes either a hit or a fault. The *hit ratio* is defined as the number of hits divided by the number of queries. It reflects the fraction of queries that were fully answered from the cache, and required no processing whatsoever by the index.

Cold and Warm caches We begin all our simulations with empty, *cold* caches. In the first part of the simulation, the caches gradually fill up. Naturally, the number of faults during this initial period is very high. When a cache reaches full capacity, it becomes *warm* (and stays warm for the rest of the simulation). The hit ratios we report are for full, warm caches only. The definition of a full cache for the PLRU and TLRU schemes is straightforward - the caches of those schemes become warm once the page queue is full. The PSLRU and TSLRU caches become warm once the probationary segment of the SLRU becomes full for the first time. The PDC cache becomes warm once either the probationary segment of the SLRU or the PQ component reach full capacity.

4.2 Results - Scheme by Scheme

PLRU and TLRU Both schemes were tested for the eight cache sizes $4000 * 2^i$, $i = 0, \ldots, 7$, and for fetch units of $1, 2, 3, 4, 5, 10$ and 20 (56 tests per scheme). The results are shown in Figure 5. The qualitative behavior of both schemes was nearly identical. We therefore discuss them jointly. In the following, we denote the hit ratio of either policy with fetch unit f and cache size s by $LRU(s, f)$.

- The results clearly demonstrate the impact of the fetch unit: $LRU(s, 3)$ is always higher than $LRU(4s, 1)$. In fact, $LRU(16000, 3)$ is higher than $LRU(512000, 1)$, despite the latter cache being 32 times larger.

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

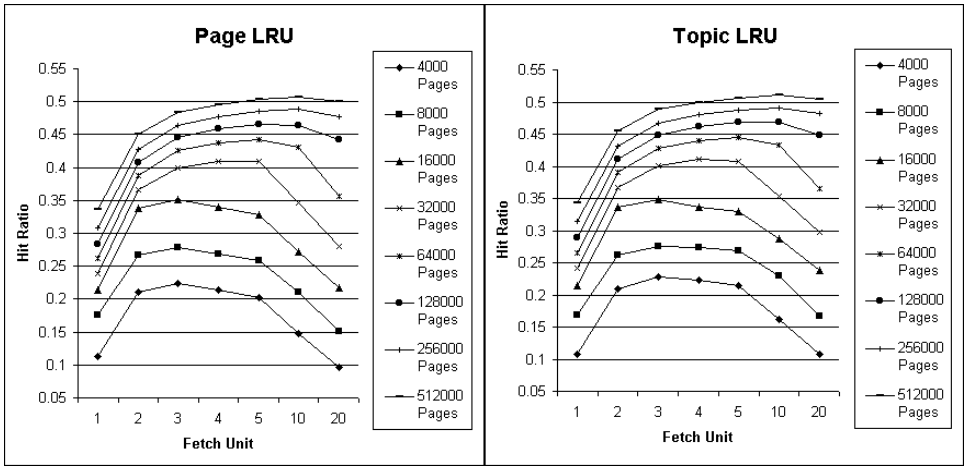

Figure 5: Results of Page LRU and Topic LRU caching schemes

- For $s = 4000$, the optimal fetch unit is 3, and $LRU(4000, 3)$ is about $2 \cdot LRU(4000, 1)$. For large sizes, optimizing the fetch unit can increase the hit ratio by about 0.17 as compared with $LRU(\text{size}, 1)$ - an increase of over 50% in the hit ratio.

- The optimal fetch unit increases as the size of the cache increases. For the three smallest sizes, the optimal fetch unit was 3. As the caches grew, the optimal fetch unit (of those examined) became 4, 5 and 10.

- The increase in performance that is gained by doubling the cache size is large for small caches (an increase of 0.05 and beyond in the hit ratio for caches holding 16000 pages or less). However, for large sizes, doubling the cache size increases the hit ratio in just about 0.02.

Table 3 summarizes the effect of the fetch unit on the hit ratios of PLRU and TLRU. For each cache size s, the hit ratio $LRU(s, 1)$ is compared to the hit ratio achieved with the optimal fetch unit (f_{opt}). Note that the former ratios, which are between 0.3 and 0.35 for the large cache sizes, are consistent with the hit ratios reported by Markatos [11]. Also note that PLRU outperforms TLRU for small caches, while TLRU is better for large caches (although the difference in performance is slight).

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

Cache size (s)	PLRU(s,1)	PLRU f_{opt}, resulting hit ratio	TLRU(s,1)	TLRU f_{opt}, resulting hit ratio
4000	0.113	3, 0.224	0.107	3, 0.228
8000	0.176	3, 0.278	0.168	3, 0.276
16000	0.215	3, 0.350	0.215	3, 0.349
32000	0.239	5, 0.410	0.241	4, 0.411
64000	0.261	5, 0.442	0.265	5, 0.445
128000	0.284	5, 0.465	0.288	10, 0.469
256000	0.308	10, 0.488	0.314	10, 0.491
512000	0.336	10, 0.508	0.343	10, 0.511

Table 3: Impact of the fetch unit on the performance of PLRU and TLRU

PSLRU and TSLRU These two schemes have an additional degree of freedom as compared with the LRU based schemes - namely, the ratio between the size of the protected segment and that of the probationary segment. Here, we tested four cache sizes ($s = 4000 * 4^i,\ i = 0, \ldots, 3$) while varying the size of the probationary segment of the SLRU from $0.5s$ to $0.9s$, in $0.1s$ increases. Having seen the behavior of the fetch unit in the LRU-based schemes, we limited these simulations to fetch units of $2, 3, 4, 5$ and 10. Overall, we ran 100 tests [10] per scheme. As with PLRU and TLRU, here too there was little difference between the behavior of PSLRU and TSLRU. Therefore, Figure 6 shows results of a single scheme for each cache size. The label of each plot describes the corresponding relative size of the probationary SLRU segment.

The effect of the fetch unit is again dramatic, and is similar to that observed for the PLRU and TLRU schemes. As before, the optimal fetch unit increases as the cache size grows. Furthermore, the optimal fetch unit depends only on the cache size, and is independent of how that size is partitioned among the two SLRU segments.

The effect of the relative sizes of the two SLRU segments on the hit ratio is less significant. To see that, we fix the optimal fetch unit for each cache size s, and examine the different hit ratios that are achieved with different SLRU partitions. For $s = 4000$ and $s = 16000$, the hit ratio increased

[10] 4 cache sizes $\times$ 5 fetch units $\times$ 5 probationary segment sizes

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

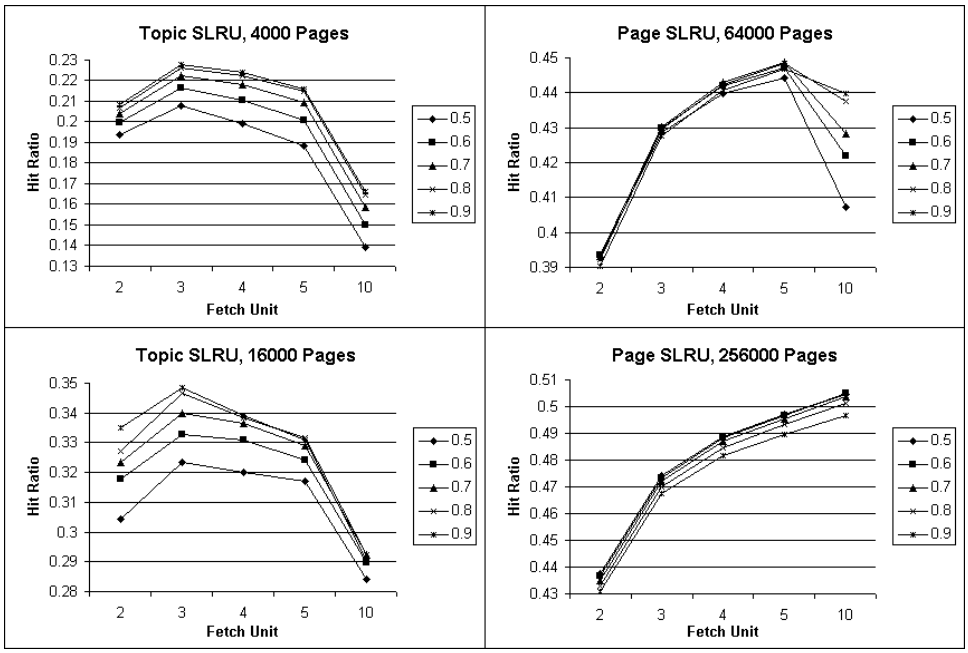

Figure 6: Results of the PSLRU and TSLRU caching schemes

as the probationary segment grew. However, the increase in hit ratio between the best and worst SLRU partitions was at most 0.025. For the larger sizes of $s = 64000$ and $s = 256000$, the results were even tighter. The optimal sizes of the probationary segment were $0.7s$ and $0.6s$ respectively, but all five examined SLRU partitions achieved hit ratios that were within 0.01 of each other. We conclude that once the fetch unit is optimized, the relative sizes of the two cache segments are marginally important for small cache sizes, and hardly matter for large caches.

PDC This scheme has two new degrees of freedom which were not present in the SLRU-based schemes: the length of the window QW, and the ratio between the capacity of the SLRU, which holds the $(t, 1)$ pages of the various topics, and the capacity of the priority queue PQ, that holds all other result pages. We tested three window lengths (2.5, 5 and 7.5 minutes), four cache sizes ($s = 4000 * 4^i,\ i = 0, \ldots, 3$), and 7 fetch units $(1, 2, 3, 4, 5, 10, 20)$. For every one of the 84 combinations of window length, cache size and fetch unit, we tested 20 partitionings of the cache: four PQ sizes ($0.3s, 0.35s, 0.4s$ and $0.45s$), and the same 5 internal SLRU partitionings as examined for the PSLRU and TSLRU schemes. To summarize, 560 different simulations were executed for each

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

window size, giving a total of 1680 simulations. Our findings are described below.

The most significant degree of freedom was, again, the fetch unit. As with the previous schemes, the optimal fetch unit grew as the size of the cache grew. Furthermore, the optimal fetch unit depended only on the overall size of the cache, and not on the specific boundaries between the three storage segments. When limiting the discussion to a specific cache size and the corresponding optimal fetch unit, the hit ratio is quite insensitive to the boundaries between the three cache segments. The difference in the hit ratios that are achieved with the best and worst partitions of the cache was no more than 0.015. As an example, we bring the hit ratios for a PDC of 64000 total pages, with a window of 5 minutes. The optimal fetch unit proved to be 5. Figure 7 brings the hit ratios of 20 different partitionings of the cache - four PQ sizes (corresponding to $0.45, 0.4, 0.35$ and 0.3 of the cache size), and five partitionings of the remaining pages in the SLRU (the x-axis corresponds to the relative size of the probationary segment of the SLRU). The trends are clear - a large PQ and an equal partitioning of the SLRU outperform smaller PQs or skewed SLRU partitionings. However, all 20 hit ratios are above 0.453 and below 0.468.

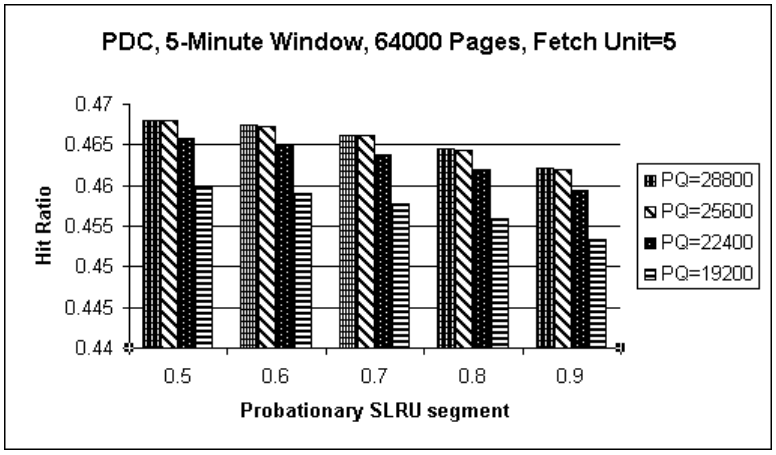

Figure 7: Insensitivity of the PDC scheme to internal partitions of the cache

Table 4 brings the optimal parameters for the 12 combinations of cache sizes and window lengths that were examined, along with the achieved hit ratio. The PQ size is relative to the size of the cache, while the size of the probationary segment of the SLRU is given as a fraction of the SLRU size (not of the entire cache size). The results indicate that the optimal window length grows as the

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

window (minutes)	cache size (result pages)	fetch unit	PQ size	SLRU probationary segment size	hit ratio
2.5	4000	3	0.45	0.8	0.243
2.5	16000	4	0.45	0.6	0.3728
2.5	64000	5	0.45	0.5	0.4668
2.5	256000	10	0.35	0.5	0.53
5	4000	3	0.45	0.8	0.229
5	16000	5	0.45	0.6	0.3691
5	64000	5	0.45	0.5	0.468
5	256000	10	0.35	0.5	0.5309
7.5	4000	3	0.45	0.9	0.2165
7.5	16000	5	0.45	0.6	0.3627
7.5	64000	5	0.45	0.5	0.4627
7.5	256000	10	0.35	0.5	0.5319

Table 4: Optimal parameters and hit ratios for the tested PDC settings

cache size grows. For the smaller two caches, the 2.5-minute window outperformed the two larger windows (with the margin of victory shrinking at 16000 pages). The 5-minute window proved best when the cache size was 64000 pages, and the 7.5-minute window achieved the highest hit ratios for the 256000-page cache. With small cache sizes, it is best to consider only the most recent requests. As the cache grows, it pays to consider growing request histories when replacing cached pages.

As for the internal partitioning of the cache, all three window sizes agree that (1) the optimal PQ size shrinks as the cache grows, and (2) the probationary segment of the SLRU should be dominant for small caches, but both SLRU segments should be roughly of equal size in large caches.

4.3 Cross-Scheme Comparison

Figure 8 shows the optimal hit ratios achieved by 7 cache replacement policies: PLRU, TLRU, PSLRU, TSLRU and PDC schemes with windows of $2.5, 5$ and 7.5 minutes. Results for 4 cache sizes are shown: $4000 \cdot 4^i,\ i = 0, \ldots, 3$. For each cache size and policy, the displayed hit ratios are the highest that were achieved in our experiments (with the optimal choice of the fetch unit

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

and the partitioning of the cache). The optimal fetch unit was consistent almost throughout the results - for cache sizes of $4000, 64000$ and 256000 pages, the optimal fetch units were $3, 5$ and 10 respectively, in all schemes. For caches of 16000 pages, the optimal fetch unit varied between 3 and 5, depending on the scheme.

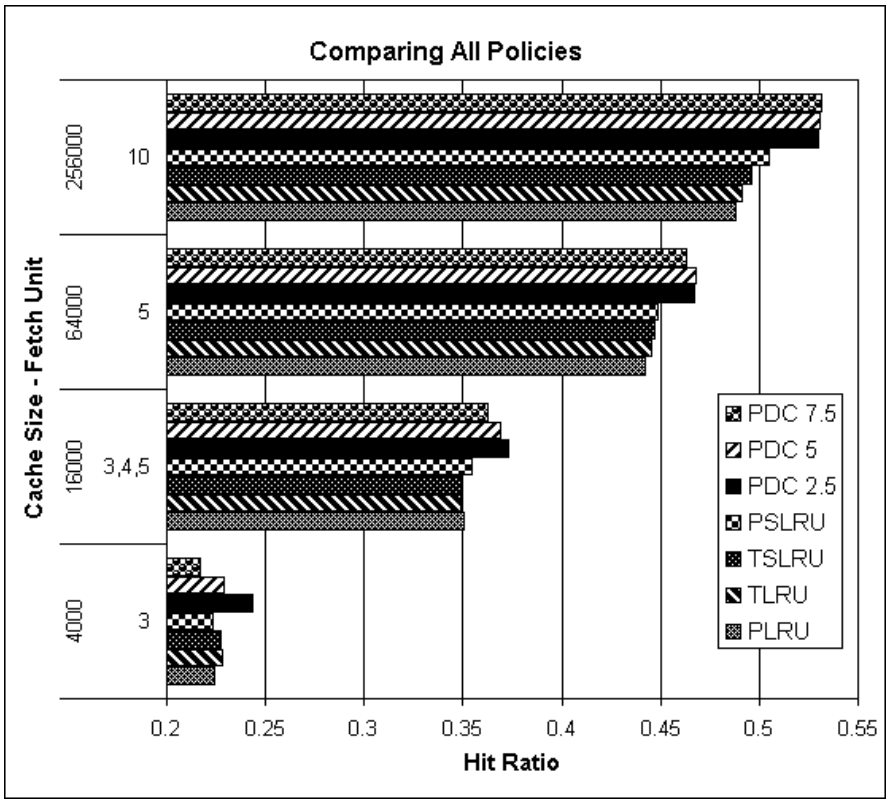

Figure 8: Comparison of all Policies

As Figure 8 shows, the best hit ratios for all cache sizes were achieved using PDC. In fact, the 2.5 and 5 minute PDCs outperformed the four LRU and SLRU based schemes for all cache sizes, with the 7.5-minute PDC joining them in the lead for all but the smallest cache. Furthermore, in all but the smallest cache size, the hit ratios are easily clustered into the high values achieved using PDC, and the lower (and almost equal) values achieved with the other four schemes. For large cache sizes, The comparison of PDC and TLRU (which is better than PLRU at large sizes) reveals that PDC is competitive with a twice-larger PLRU: the optimal 64000-page PDC achieves a hit ratio of 0.468 while the 128000-page TLRU has a hit ratio of 0.469, and all three 256000-page

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

PDC schemes outperform the 512000-page TLRU (0.53 to 0.51).

The hit ratios of all three 256000-page PDC schemes were above 0.53, and were achieved using a fetch unit of 10. Recall that Table 2 brought upper bounds on the hit ratio that is achievable on the query log examined. These bounds correspond to infinite caches with prior knowledge of the entire query stream. For a fetch unit of 10, the upper bound was 0.62, and the bound for *any* fetch unit was 0.629. Thus, PDC achieves hit ratios that are beyond 0.84 of the theoretic upper bound.

When limiting the discussion to the LRU and SLRU based schemes, Page SLRU is the best scheme for all but the smallest cache size. This is consistent with the results of Markatos[11], where PSLRU outperformed PLRU (with the fetch unit fixed at 1).

5 Conclusions and Future Research

We have examined five replacement policies for cached search result pages. Four of the policies are based on the well-known LRU and SLRU schemes, while the fifth is a new approach called PDC, which assigns priorities to its cached result pages based on a probabilistic model of search engine users. Trace-driven simulations have shown that PDC is superior to the other tested caching schemes. For large cache sizes, PDC outperforms LRU-based caches that are twice as large. It achieves hit ratios of 0.53 on a query log whose theoretic hit ratio is bounded by 0.629.

We also studied the effect of other parameters, such as the fetch unit and the relative sizes of the various cache segments, on the hit ratios. The fetch unit proved to be the dominant factor in optimizing the caches' performance. Optimizing the fetch unit can double the hit ratios of small caches, and can increase these ratios in large caches by 50%. With optimal fetch units, small caches outperform much larger caches whose fetch unit is not optimal. In particular, a size-s cache with an optimal fetch unit will outperform caches of size $4s$ whose fetch unit is 1. The impact of the fetch unit on the hit ratio is much greater than the impact achieved by tuning the internal partitioning of the cache. Furthermore, the optimal fetch unit depends only on the total size of the cache, and not on the internal organization of the various segments.

An important benefit that a search engine enjoys when increasing the hit ratio of its query result cache, is the reduction in the number of query executions it must perform. However, while large fetch units may increase the hit ratio of the cache, the complexity of each query execution

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

grows as the fetch unit grows [10]. Although the increase in the complexity of query executions may be relatively small in many search engine architectures, it should be noted that the hit ratio is not the sole metric by which the fetch unit should be tuned.

An important and intuitive trend seen throughout our experiments is that large caches can take into account longer request histories, and prepare in advance for the long term future. Planning for the future is exemplified by the increase of the optimal fetch unit as the cache size grows. In all schemes, the optimal fetch unit grew from 3 to 10 as the cache size increased from 4000 to 256000 pages. Since the fetch unit essentially reflects the amount of prefetching that is performed, our results indicate that large caches merit increased prefetching. As for considering longer request histories, this is exemplified by the PDC approach, where the optimal length of the query window increased as the cache size grew.

The schemes employing some form of SLRU (PSLRU, TSLRU and PDC) also exhibit an increased "sense of history" and "future awareness" as their caches grow. In these schemes, the relative size of the protected segment increased with the cache size. A large protected segment is, in a sense, a manner of planning for the future since it holds and protects many entries against early removal. Additionally, only long request histories contain enough repeating requests to fill large protected segments.

The following is left for future research:

- The model of search engine users that gave rise to the PDC scheme is fairly simple. In particular, our modeling of the time that is allowed between successive queries in a search session is simplistic. While PDC outperformed the other schemes we tested, more elaborate models may result in further improvement in performance.
- The query trace available to us held anonymous information - users (or user-IDs) were not associated with queries. Search engines, however, can associate (to some extent) queries with users through the use of session-IDs, cookies, and other mechanisms. Caching policies that are aware of the coupling of queries and users can track the "active search sessions", and can tap this knowledge to improve performance.

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

- Our results demonstrate the high impact of prefetching on the performance of the cache. Policies in which the fetch unit depends on the number of the uncached page that is requested (rather than being constant, as in this paper) should also be examined.

Acknowledgments

We thank Andrei Broder [11], Farzin Maghoul and Prashanth Bhat from AltaVista for helpful discussions and insights on query result caches, and for providing us with the query log for our experiments.

References

[1] Lada A. Adamic and Bernardo A. Huberman. The nature of markets in the world wide web. *Quarterly Journal of Economic Commerce*, 1:5–12, 2000.

[2] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Proc. 7th International WWW Conference*, 1998.

[3] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Proc. 9th International WWW Conference*, 2000.

[4] Pei Cao, Jin Zhang, and Kevin Beach. Active cache: Caching dynamic contents on the web. In *Proc. of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pages 373–388, 1998.

[5] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

[6] Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. Real life, real users, and real needs: A study and analysis of user queries on the web. *Information Processing and Management*, 36(2):207–227, 2000.

[7] Björn THór Jónsson, Michael J. Franklin, and Divesh Srivastava. Interaction of query evaluation and buffer management for information retrieval. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA*, pages 118–129, June 1998.

[8] Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, 1994.

[9] Achim Kraiss and Gerhard Weikum. Integrated document caching and prefetching in storage hierarchies based on markov-chain predictions. *VLDB*, 7(3):141–162, 1998.

[10] Ronny Lempel and Shlomo Moran. Optimizing result prefetching in web search engines with segmented indices. In *Proc. 28th International Conference on Very Large Data Bases, Hong Kong, China*, 2002.

[11] Andrei Broder is currently with IBM Research

Technion - Computer Science Department - Technical Report CS-2002-16 - 2002

[11] Evangelos P. Markatos. On caching search engine query results. *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, May 2000.

[12] Michael Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Invited Talk in the 39th Annual Allerton Conference on Communication, Control and Computing*, October 2001.

[13] P. Saraiva, E. Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Ribeiro-Neto. Rank-preserving two-level caching for scalable search engines. In *Proc. 24rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, Louisiana, USA*, pages 51–58, 2001.

[14] Craig Silverstein, Monika Henzinger, Hannes Marais, and Michael Moricz. Analysis of a very large altavista query log. Technical Report 1998-014, Compaq Systems Research Center, October 1998.

[15] Athena Vakali. Proxy cache replacement algorithms: A histroy-based approach. *World Wide Web*, 4(4):277–297, 2001.

[16] Yi-Hung Wu and Arbee L.P. Chen. Prediction of web page accesses by proxy server log. *World Wide Web*, 5(1):67–88, 2002.

[17] Qiang Yang and Henry Hanning Zhang. Integrating web prefetching and caching using prediction models. *World Wide Web*, 4(4):299–321, 2001.