Taylor Series Prediction: A Cache Replacement Policy Based on Second-Order Trend Analysis

Qiang Yang, Haining Henry Zhang and Hui Zhang
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada V5A 1S6
(qyang, hzhangb)@cs.sfu.ca, h8zhang@math.uwaterloo.ca

Abstract

Caching is one of the most effective techniques for improving the performance of Internet systems. The heart of a caching system is its page replacement policy*, which decides which page to replace in a cache by a new one. Different caching policies have dramatically different effects on the system performance. In this paper, we extend the well-known GDSF caching policies to include not only access trend information, but also the* dynamics *of the access trend itself to the trends on access trends. The new trend policy that we propose, called Taylor Series Prediction (TSP) policy, provides more accurate prediction on future accessing trends when the access patterns vary greatly. We back up our claims through a series of experiments using web access traces.*

1 Introduction

As the World Wide Web grows at a very rapid rate, researchers have designed various effective caching algorithms to contain network traffic. The idea of web caching is to maintain a highly efficient but small set of retrieved results in a system cache, such that much of the traffic can be *shortcut* if the user query can be directly answered by one of the pages in the cache. This can often result in significant improvement in system performance.

Lying in the heart of caching algorithms is the so called "page replacement policy", which specifies conditions under which a new page will replace an existing one. Many replacement policies have been proposed over the years, including the LRU-K algorithm [OOW93], which rejects the Least-Recently-Used page in most recent K accesses, the GD-size policy [CI97] which considers access costs and varying page sizes, and an enhancement of the GD-size algorithm known as GDSF [ALCJ99, Che98] which incorporates the frequency information. The basic idea of most of these caching algorithms is to rank pages based on their access trend in addition to factors such as size, frequency and cost, so that pages that are "young", relative to its own last access, are ranked higher, and pages that are "old" are ranked lower.

Our key observation is that the trend or page reference rate of a web page can be considered in the same way as the concept of "speed", which captures how fast an object moves in classical mechanics. Although this rate is an adequate measure for how likely a page will be referenced next, it fails to capture the trend in which the reference rate itself changes in the near future. In the analogy with classical mechanics, this reference-rate trend corresponds to the concept of "acceleration."

In the Internet environment, the paging need usually ranges over an extended period of time, and the fetching of a web page can be a long process. For example, a group of users querying a travel web site about potential travel plans will likely pose requests over a period of several days. Once the request is completed, then the reference rate will gradually die out as the interest of users in this page will be reduced. When this happens, it is important for a web server or a proxy server to notice the change in this trend, so as to better prepare for the caching of pages from new sites through methods such as pre-fetching.

We propose a new replacement policy by adding an additional factor to compute the reference rate trend. To do this, we introduce Taylor Series Prediction (TSP) as a mathematical tool for incorporating both the reference rate and the trend in reference rate, and show that this analysis nicely summarizes the LRU-K based policies and compares nicely with the GD-size class of policies. We show experimentally that the new TSP algorithm can perform better in many domains.

The organization of the paper is as follows. In the next Section, we review the work in caching and present our motivation. Then in Section 3 we introduce the formal Taylor Series Analysis Model. Then, in Section 4, we present our experimental results related to this new model, and conclude in Section 5.

2 Background

Caching is a mature technique that has been widely applied in many computer science areas. Among those areas, Operating Systems and Databases are two most important ones. Currently, the World Wide Web is becoming another popular application area of caching. Below, we briefly review caching in these areas.

The widespread LRU (Least-Recently-Used) algorithm is a good approximation to the optimal page replacement algorithms by considering the age as well as the reference frequency of a page. It is based on the assumption that pages which have been heavily used in the past will probably be used again in the near future, and pages which have not been used recently will probably remain unused for a long time. Consequently, in the LRU algorithm, the page which has not been used for the longest period of time is replaced. This algorithm is chosen as the page replacement policy by almost all commercial systems.

The LRU-K algorithm is motivated by knowing that the popular LRU algorithm is not always appropriate for the database environment (for more details, see [Rei76] [Sto81] [SS86] and [CD85]). The key observation is that LRU keeps only the time of last reference to each page when making page replacement decision. Thus, the LRU algorithm cannot well distinguish between frequently referenced pages and infrequently referenced pages due to the limited information it is based on. The basic idea of LRU-K is to consider the time of the last K references to a page and to use such information to make page-replacement decision. To quote the original description in [OOW93] :

> The page to be dropped (i.e., selected as a replacement victim) is the one whose Backward K-distance, is the maximum of all pages in buffer.

The increasing usage of the World Wide Web has led to a great deal of traffic on the Internet, which in turn results in the degradation of network performance. Therefore, it is desirable that the traffic is reduced or smoothed by caching the popular web objects. With this goal, web caching has become an active research area and gained a lot of attention [AWY99, Mar96, Gla94].

Caching can be done at various levels in the network. It can lie on the side of a web server, a caching proxy, or a client. Caching proxies are a special kind of servers that are responsible for locating a cached copy of an object that is required. Web servers and caching proxies are higher level caches, while client caches are at lower levels. Web caching is different from traditional caching in several ways. An important difference is that the size of web objects is not uniform, and the transfer time costs are not unique either. For the purpose of maximizing the hit ratio or byte-hit ratio, it is better for a caching replacement policy to take factors such as size and network cost into account.

One of the most successful algorithms is the GD-size algorithm introduced by Cao and Irani [CI97]. When a new object arrives, GD-size increases the ranking of a new object i by the cost of the removed object. Let S_i be the size of the new object i, C_i be the cost of object i, and K_i, the key value, be the rank of object i. Furthermore, let L be an inflation factor for a newly admitted object, where L is updated as the key value of the most recently removed object. Then, if i is in the cache, then the key value of object i is updated:

$$K_i = L + C_i/S_i \qquad (1)$$

Otherwise, if i is new and not yet in the cache, then

$$L = \min_j K_j \qquad (2)$$

where j are objects in the cache. Then the object l with $K_l = L$ is ejected, and object i is inserted in the cache with K_i set according to Equation 1.

As an enhancement of the GD-size algorithm, Arlitt et. al [ALCJ99] introduced the frequency factor F_i which counts the number of references so far. With this new factor, the key value can be computed as

$$K_i = L + F_i * C_i/S_i \qquad (3)$$

Both algorithms perform very well across a number of domains.

3 Taylor Series Prediction

3.1 Problem Statement

Both the GD-size and GDSF formulas are similar to the LRU algorithm by keeping track of the most recently accessed pages. Higher priority is given to objects that are accessed the latest. In contrast, we wish to design a policy based on a "look forward" estimate. Consider the time difference $\Delta(T) = T_p - T_c$, where T_p is the predicted time of next access, and T_c is the current time. In essence, objects that have large $\Delta(T)$ should be punished, and objects that have small $\Delta(T)$ should be rewarded. Therefore, we have an opportunity to enhance the GDSF formula (Equation 3) even further, by including the $\Delta(T)$ factor in the ranking:

$$K_i = \frac{F_i * C_i}{S_i * \Delta(T)} \tag{4}$$

One of the most important factors in this new formula is $\Delta(T)$, which can in fact be made accurate by including not just the first order estimation but the second. In the first order estimation, only the current time and the last access time T_{c-1} are used to calculate $\Delta(T)$:

$$\Delta(T) = \frac{T_c - T_{c-1}}{1}$$

This difference estimation corresponds to the concept of speed, where the difference in time is divided by a unit distance. To obtain a better estimation of the time, we wish to use the concept of "acceleration", which gives rise to Taylor Series Prediction.

3.2 Taylor Series

The Taylor series is named after the English mathematician Brook Taylor(1685-1731). Its definition was given like this [Ste95]: If f has a power series representation(expansion) at point a and the radius of convergence of the power series is $R > 0$, that is, if

$$f(x) = \sum_{n=0}^{\infty} c_n (x-a)^n \qquad |x-a| < R \tag{5}$$

then its coefficients are given by the formula

$$c_n = \frac{f^{(n)}(a)}{n!}$$

Therefore, f must be in the following form:

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \\ &= f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 \\ &\quad + \frac{f'''(a)}{3!}(x-a)^3 + \cdots \end{aligned} \tag{6}$$

The series in equation 6 is called the Taylor series of the function f at a and function f is called *analytic* at a, if $R > 0$.

From the above definition, it is clear that $f(x)$ is the limit of a sequence of partial sums. In the case of Taylor series, the partial sums are

$$\begin{aligned} f_n(x) &= \sum_{j=0}^{n} \frac{f^{(j)}(a)}{j!}(x-a)^j \\ &= f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 \\ &\quad + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n \end{aligned} \tag{7}$$

f_n is called nth-degree Taylor polynomial of f at a.

Now consider the *reference time function* $T(n)$, where n is a count on the number of access. For each page i, its associated reference time function is defined as $T_i(n)$ where we drop the index i when it is clear about the object being referred to. $T_i(n)$ is simply the time in which the object i was accessed the n^{th} time.

Given a time series $T(1), T(2), \cdots, T(n)$, where $T(n)$ is the time of the nth reference of a page P, we use Taylor Series to predict the time of the $(n+1)$th reference to this page, $T(n+1)$. Here we use the second-degree Taylor polynomial to approximately predict $T(n+1)$. Since $T(n)$ is a discrete function, the Taylor series expression is:

$$T(n+1) \approx T(n) + \frac{T'(n)}{1!} + \frac{T''(n)}{2!} \tag{8}$$

where

$$T'(n) = \frac{T(n) - T(n-k)}{k}$$

$$\begin{aligned} T''(n) &= \frac{T'(n) - T'(n-k)}{k} \\ &= \frac{T(n) - 2T(n-k) + T(n-2k)}{k^2} \end{aligned}$$

Now let us set $k = 1$, and let $T(n)$ be T_c, the current time (that is, we predict the $(n+1)$th referrence when the page is accessed the nth time). Let T_{c-1} be the time an object is accessed the last time, and T_{c-2} be the time when the object is accessed two times before. Then the predicted time is

$$\begin{aligned} T_p &\approx T_c + (T_c - T_{c-1})/1! \\ &\quad + ((T_c - T_{c-1}) - (T_{c-1} - T_{c-2}))/2! \\ &= (5T_c - 4T_{c-1} + T_{c-2})/2 \end{aligned} \tag{9}$$

Dataset	Requests	Distinct URLs	Total Hours
EPA	43202	4381	24
NASA	1140284	4400	417
CSSFU	17330	4453	12
GZ	4334	250	6

Table 1: Experimental data properties

This is the formula we use in calculating T_p in the TSP-estimation formula Equation 4.

The complexity of this formula is linear in the size of the cache, since when a new object arrives, priorities have to be recalculated for all objects.

4 Experiments

We have conducted a series of experimental comparisons with the four web server logs we are able to obtain. These server access log data are listed in Table 1. In the experiments, the EPA (United States Environmental Protection Agency) data contains a day's worth of all HTTP requests to the EPA WWW server located at Research Triangle Park, NC. The NASA data is from NASA Kennedy Space Center WWW server in Florida (due to memory limitation, we picked out requests for the first 417 hours to do our experiments). The CSSFU data is the web server data from the School of Computing Science at Simon Fraser University for a time span from Sept. 27, 1999 0:00am to 12pm. GZ data is from a public ISP web server in Guangzhou City, PRC.

Before simulations, we removed uncacheable URLs from the access logs. A URL is considered uncacheable when it contains dynamically generated content such as CGI queries. We also filtered out requests with unsuccessful HTTP response code. The overall statistics of the cleaned logs are shown in Table 1.

The performance metrics we used are *hit ratio*(HR) and *byte hit ratio*(BHR). *Hit ratio* is defined as the number of requests responsed directly from cache as a percent of the total number of requests. *Byte Hit ratio* is the number of bytes responsed directly from cache as a percent of the total number of bytes of all requests. The results illustrating both hit ratios and byte hit ratios are shown in Figures 1 to 4. In these figures, X-axis is the cache size as a percent of the total data size. The algorithms under comparison are GDSF, GDSize, TSP, LRU, and LFUDA where LFUDA is the Least-Frequently-Used with Dynamic Aging method [ALCJ99].

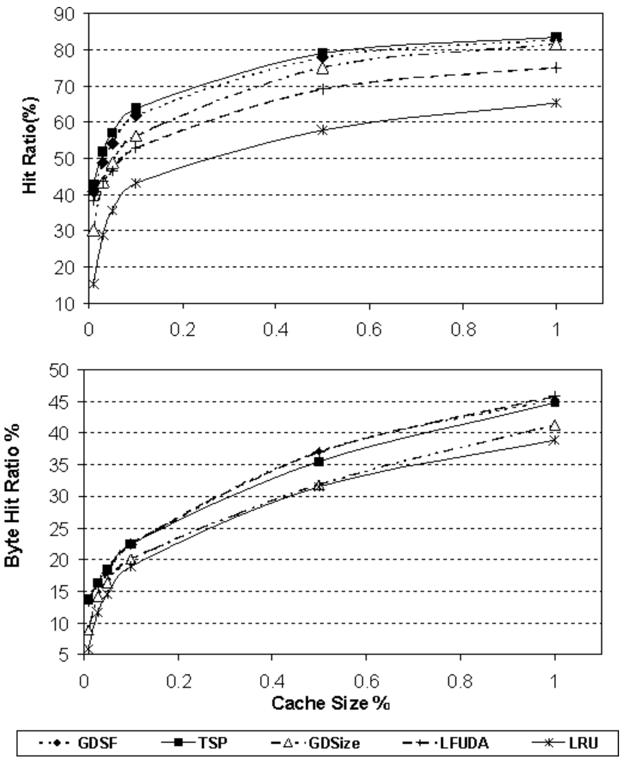

Figure 1: EPA hit ratio and byte hit ratio comparison

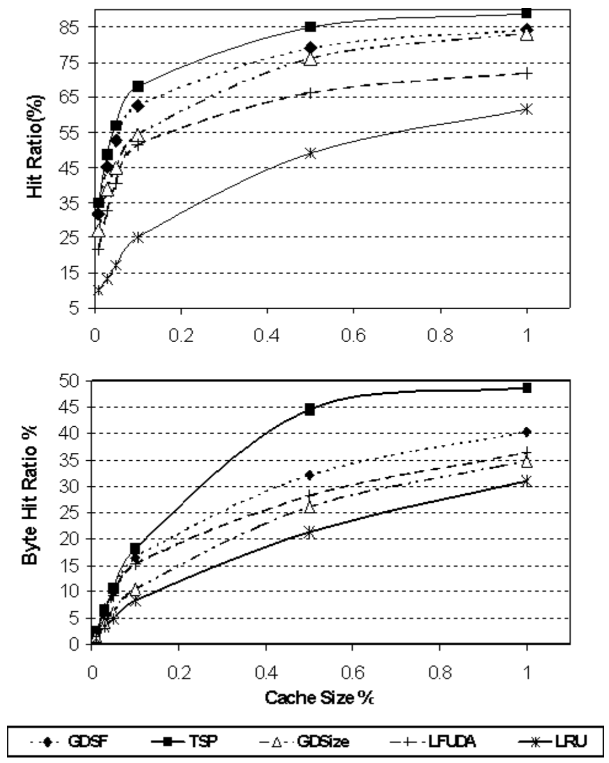

Figure 2: NASA hit ratio and byte hit ratio comparison

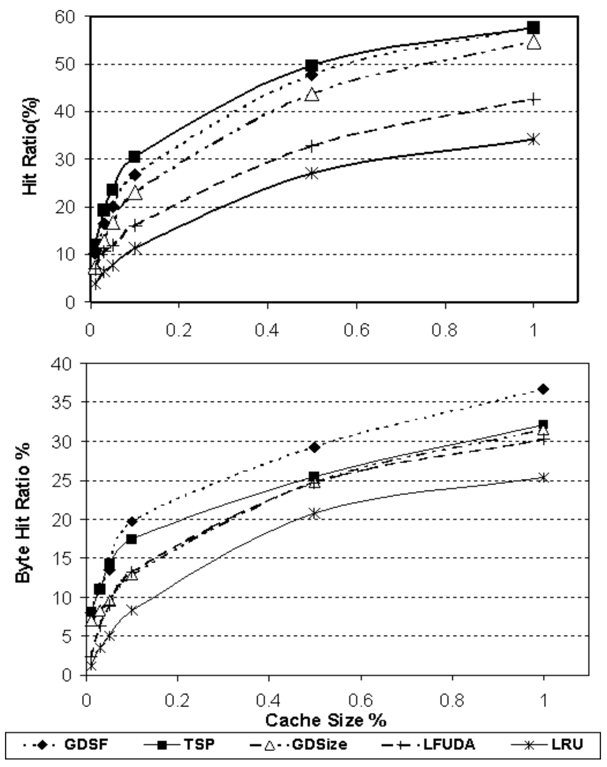

Figure 3: CSSFU hit ratio and byte hit ratio comparison

Data Set	Mean Difference %	Std Dev
EPA	1.87	0.97
CSSFU	2.36	1.49
GZ	7.01	7.31
NASA	4.49	1.11

Table 2: Hit ratio differences of (TSP - GDSF) over four data sets

Data Set	Mean Difference %	Std Dev
EPA	-0.37	0.65
CSSFU	-1.65	2.25
GZ	0.80	1.81
NASA	3.81	5.30

Table 3: Byte hit ratio differences of (TSP - GDSF) over four data sets

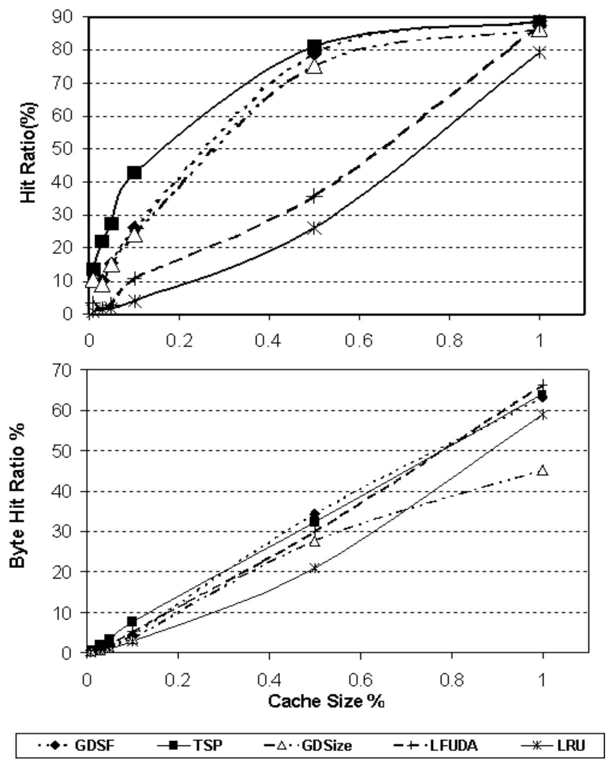

Figure 4: GZ hit ratio and byte hit ratio comparison

Overall, the differences in hit ratios and byte hit ratios between TSP and GDSF are summarized in Tables 2 and 3. Mean difference is computed as the average of HR or BHR difference between TSP and GDSF across varied cache sizes. As can be seen, overall, TSP performed very well, outperforming all in NASA and GZ data sets. However, in byte-hit ratios, TSP is still not performing as well as GDSF in EPA and CSSFU data. This can be explained by the fact that by including the $\Delta(T)$ factor in the GDSF formula as we have done in Equation 4, we are reducing the importance of the size factor. Therefore TSP pays less attention to size differences than GDSF does. We are working on new policies to remove this deficiency.

We have also compared TSP using first-order prediction and second order prediction. First order prediction uses only the first derivative in the Taylor series formula. This comparison helps justify why we use TSP with second order prediction for the caching policy. Figure 5 shows the difference of hit ratio and byte hit ratio for the EPA data between first and second order TSP predictions. As can be seen, the difference is rather large. The same can be observed from NASA data (see Figures 6).

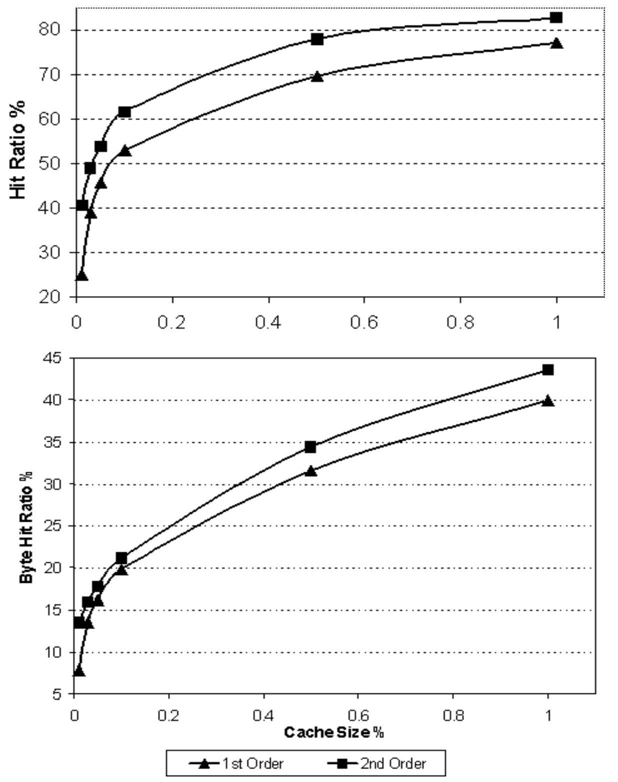

Figure 5: EPA hit ratio comparison between first and second order predictions

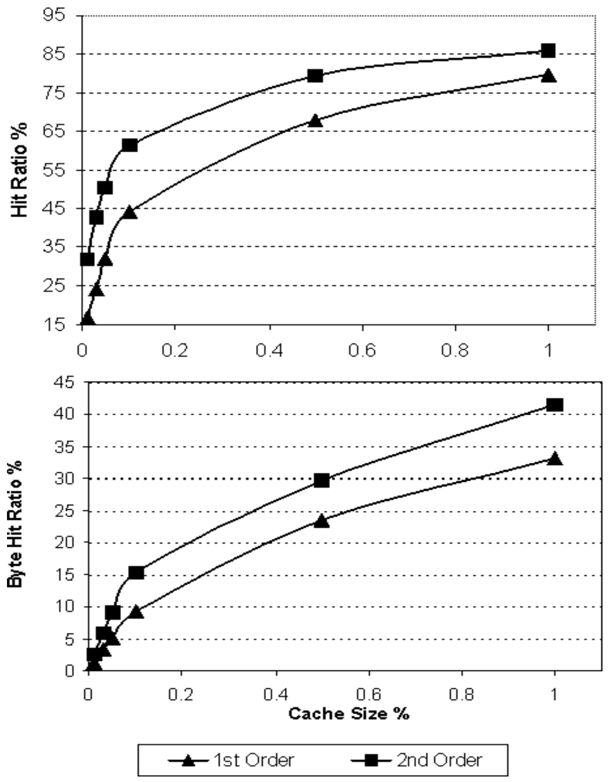

Figure 6: NASA hit ratio comparison between first and second order predictions

5 Conclusions and Future Work

In this paper we have introduced a new caching technique that improves the well-known LRU, GDSF and GDSize algorithms. Our basic observation is that by taking into account the reference rate trend analysis, the caching policy can be made more accurate in predicting future hitting patterns. The experiments support our claim. We conclude that in application domains where there is often a clear trend pattern in the usage of data pages, such as in the World Wide Web environment, our TSP based caching policy will show improvements.

In the future, it is possible for us to take into account other statistical features such as the data transmission rates as evidenced in applications on the Internet.

Acknowledgments

The authors would like to thank our funding partners: NSERC, BC ASI, Rogers Cablesystems Ltd., Canadian Cable Labs Fund, IRIS/PRECARN and Simon Fraser University.

References

[ALCJ99] M. Arlitt, R. Friedrich L. Cherkasova, J. Dilley, and T. Jin. Evaluating content management techniques for web proxy caches. In *HP Technical report*, Palo Alto, Apr. 1999.

[AWY99] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the World Wide Web. In *IEEE Transactions on Knowledge and Data Engineering*, volume 11, pages 94–107, 1999.

[CD85] H. T. Chou and D. J. DeWitt. An evaluation of buffer management strategies for relational database systems. In *Proceedings of the Eleventh International Conference on Very Large Databases*, pages 127–141, August 1985.

[Che98] L. Cherkasova. Improving www proxies performance with greedy dual size frequency caching policy. In *Technical Report*, HP Laboratories, Nov. 1998.

[CI97] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, Dec. 1997.

[Gla94] S. Glassman. A caching relay for the World Wide Web. In *The First International World Wide Web Conferencing, Geneva*, 1994.

[Mar96] E. Markatos. Main memory caching of web cocuments. In *Computer networks and ISDN Systems*, volume 28, pages 893–905, 1996.

[OOW93] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 297–306, May 1993.

[Rei76] A. Reiter. A study of buffer management policies for data management systems. In *Technique Summary Report*, number 1619, Mathematics Research Center, University of Wisconsin, Madison, March 1976.

[SS86] G. M. Sacco and M. Schkolnick. Buffer management in relational database systems. In *ACM Transactions on Database Systems*, volume 11, pages 473–498, December 1986.

[Ste95] J. Stewart. *Calculus: Early transcendentals (3rd edition)*. Brooks/Cole Publishing Company, Pacific Grove, CA, USA, 1995.

[Sto81] M. Stonebraker. Operating system support for database management. In *Communications of the ACM*, volume 24, pages 412–418, July 1981.