A New Efficient Caching Policy for the World Wide Web

Nicolas Niclausse, Zhen Liu, Philippe Nain
INRIA Centre Sophia Antipolis
2004 Route des Lucioles
B.P. 93, 06902 Sophia Antipolis, FRANCE
{Zhen.Liu, Philippe.Nain, Nicolas.Niclausse}@sophia.inria.fr
http://www.inria.fr/mistral/Les_Gens-eng.html

Abstract

With the increasing popularity of the World Wide Web, the amount of information available and the use of Web servers are growing exponentially. In order to reduce the overhead induced by frequent requests to the same documents by local users, client caching and, more generally, proxy-caching have been proposed and are now widely used. Most implementations use traditional memory paging policies, like the Least Recently Used (LRU) policy. However, due to the heterogeneity of the requests of Web traffic, both in the size of the documents and in the network transfer delays, such caching policies are not very efficient. In this work, we propose a new caching policy which takes into account the network latency, the size of the documents, their access frequencies, and the time elapsed since the last reference to documents in the cache. Through trace-driven simulations and for various standard cost criteria (request hit rate, byte hit rate and latency ratio) we show that our policy performs better than several policies proposed in the literature.

1 Introduction

With the increasing popularity of the World Wide Web, Web traffic has become one of the most resource consuming applications on the Internet. Solutions to improve bandwidth utilization of Web traffic are currently an important issue. Different directions of research are being investigated in this area, including the use of compression and delta-encoding [14], multicast, server-push and new congestion avoidance algorithms.

Two most direct ways of reducing Web traffic have been implemented recently. On the one hand, improvements have been made in the HyperText Transfer Protocol (HTTP), the most used transfer protocol in the Web. The interaction with the lower level protocol TCP has been modified in HTTP1.1 [10] to support, in particular, persistent connections and pipelining, in order to reduce network use (up to 10 times according to recent measurements [15]) and user perceived latency.

On the other hand, caching at different levels is proposed and/or implemented: caching at the client side [8] and in the backbone network [9]. Indeed, according to statistical analysis ([3, 6]), some Web servers and/or Web documents are much more popular than others. Thus, the request stream exhibits some locality [1]: several clients in the same area may require the same files to the same server during a relatively short time interval. A proxy-cache at the client side may therefore avoid superfluous downloads of such files and hence save network bandwidth and server capacity. In a proxy-cache, a single host is used for several users (from one to several hundred clients). Each client sends its requests through the proxy which keeps copies of documents in the cache, cf. Figure 1. A by product of a proxy-cache (when it is well dimensioned) is the decrease of the user perceived network latency. Again, as in memory cache, a hierarchical system can be used in proxy-cache [8].

As in traditional memory caching, a crucial question in Web caching is to decide which documents must be kept in the cache. In other words, when a new document must be brought into the cache and that this one is full (or heavily loaded – see Section 3.1), which document(s) should be removed from the cache ? The strategy to make such decisions is referred to as *caching policy*. This paper focuses on

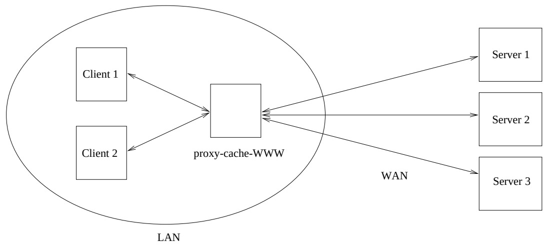

Figure 1: Web cache

caching policies for proxy-caches.

Most current implementations use traditional memory paging policies like the Least Recently Used (LRU) policy for proxy-caching. While LRU has been shown to be very efficient for memory caching (see [16, 17] for a survey), this policy is not as good for Web caching, as already reported in several trace-driven simulations studies ([2, 7, 19, 20]). This follows from the high heterogeneity of both the Web documents (their size varies from hundreds of bytes to several megabytes) and of the transfer delays on the Internet, which is in contrast with memory caching where the objects (pages) have the same size and the same communication delays. It is therefore natural to search for caching policies that integrate the specificities of Web caching. Another specificity of Web caching is that accesses to the documents come from multiple users while they originate from a single program in memory caching.

In this work, we propose a new caching policy – referred to as the MIX policy – which explicitly takes into account the network latency, the size of the documents, their access frequency, and for any document in the cache the time elapsed since it has been last referenced. Through trace-driven simulations and for various standard cost criteria (request hit rate, byte hit rate and latency ratio) we show that our policy performs better than several popular policies proposed in the literature.

The paper is organized as follows. In Section 2 we first give a brief overview of existing works pertaining to the performance of caching policies, and then introduce the new caching policy MIX. In Section 3 we report the performance comparison results obtained through trace-driven simulations. In Section 4 we point out further research directions in the field.

2 Caching Policies

Previous studies in the area of performance evaluation and optimization of caching policies roughly divide into two categories: analytical studies aiming at determining an optimal caching policy for a given cost criterion (e.g., minimize the number of misses) and evaluation and comparison of the performance of particular caching policies through trace-driven simulations. This paper falls into the second category. Before presenting further our contribution, let us first review some recent results pertaining to the first category.

2.1 Theoretical Results

The following somewhat idealized model of the behavior of a Web proxy-cache has been considered in [12]. There is a finite collection U of objects (the Web pages) that can be accessed via a Web server equipped with a proxy-cache of (finite) size B. Each time a request for a particular object in U is made to the proxy-cache then either the object is already in the cache and the request is successful (hit) or the object is not in the cache in which case the caching policy has to decide which set of objects, if any, should be removed from the cache (e.g. to leave room to the requested object and/or to leave the cache in a better configuration). Each object has a cost associated with it that may depend on its size.

In [12] off-line (the sequence of requests is finite and known in advance) and on-line (the sequence of requests in not known in advance) optimization models are considered and solved via dynamic programming techniques. In the case of an off-line model the authors show that the search for an optimal policy reduces to the solution of a deterministic shortest path problem. In this case, the objective is to minimize the sum of the costs associated with all missed requests. In the case of an on-line model, the authors assume that the requests are generated according to an ergodic Markov chain with the objective of minimizing the expected cost per request caching [12]. Here, the one-step cost is the sum of the costs associated with the requests that are removed from the cache at each miss. As acknowledged by the authors, the search for optimal policies both for the off-line and for on-line cases becomes rapidly unfeasible as the number of objects and controls increase (Bellman's "curse of dimensionality"). Indeed, the size of the graph used in the off-line solution and the number of states of the Markov chain used in the on-line solution are exponential in the number of objects (Web documents). The off-line problem was shown to be NP-hard [11]. Thus, in practice, suboptimal solutions or heuristic methods are required.

Before switching to heuristics and later on to trace-driven simulations we would like to point out that the search for an optimal policy for the on-line model in [12] could benefit a great deal from neuro-dynamic programming techniques [4]. These techniques have proven useful in cases when standard dynamic programming algorithms (value iteration algorithm, policy improvement) fail to work due to large state and control spaces. The authors of this paper are currently studying the applicability of the Q-learning algorithm [4, 18] to the on-line model proposed in [12].

2.2 Heuristics

In the sequel we will focus on the following performance metrics: the *request hit rate*, the *relative request hit rate*, the *byte hit rate*, the *relative byte hit rate* and the *latency ratio*. To define these performance measures more carefully, let m be the total number of requests (documents), a_i the size (in bytes) of the i-th request and let $\delta_i = 1$ if request i is in the cache and $\delta_i = 0$ otherwise.

The request hit rate is defined as the percentage of documents that are in the cache, namely,

$$W^r = \frac{\sum_{i=1}^{m} \delta_i}{m}. \tag{1}$$

The relative request hit rate is defined as $W^r/W^r_{\max}$, where $W^r_{\max}$ is the request hit rate obtained for a cache with an infinite size.

The byte hit rate is the percentage of bytes transferred from the cache, namely,

$$W^b = \frac{\sum_{i=1}^{m} a_i\,\delta_i}{\sum_{i=1}^{m} a_i} \tag{2}$$

The relative byte hit rate is defined as $W^b/W^b_{\max}$, where $W^b_{\max}$ is the request hit rate obtained for a cache with an infinite size.

Last, the latency ratio is defined as the ratio of the sum of download time of missing documents over the sum of all downloading latencies, namely,

$$W^t = \frac{\sum_{i=1}^{m} lat_i\,(1-\delta_i)}{\sum_{i=1}^{m} lat_i} \tag{3}$$

with lat_i the time to download the i-th referenced document from its server to the proxy-cache.

Several caching policies have been proposed in the literature, some of them being implemented in real systems (e.g. LRU in *Squid* – see Section 3). Below we list five policies we used in our simulations, and for each of them we point out its main advantages and drawbacks.

- LRU: The Least Recently Used documents are removed first.

  Advantages: Simple to implement and efficient in the case of uniform objects like in memory cache, used in other areas of caching for decades.

  Drawback: Does not consider size or download latency of documents.

- LFU: The Least Frequently Used documents are removed first.

  Advantages: Simplicity.

  Drawback: Does not consider size or download latency of documents and may keep obsolete documents infinitely in the cache (if no expiration mechanism is used).

- SIZE [19]: Big documents are removed first.

  Advantages: Removes big documents, therefore keeps a lot of small files in the cache, resulting in high request hit rate.

  Drawbacks: May keep small documents indefinitely in the cache (even if they are never accessed again). Low byte hit rate.

- GD-SIZE [7]: Greedy-Dual algorithm. A value H is associated with each document in the cache. When a page is brought into the cache, H is set to the cost function equals to $1/SIZE$. When a replacement has to be made, the document with the smallest H (call it min_H) is removed from the cache, and every document has its H reduced by min_H. If a document is accessed again, then its H is restored to its cost.

  Advantages: Intends to remove from the cache the documents which are no longer required by clients, and therefore overcome the drawback of

SIZE policy. More general cost functions can be used. However, the simple $1/SIZE$ variant yields the best performance with respect to all performance measures considered in [7].

Drawbacks: Does not take into account the delays induced by the network and the frequency at which documents are accessed.

- HYBRID [20]: For each document j in the cache, HYBRID computes its cost

  $$nref_j^{C_2} * (rtt_i + C_1/bw_i)/size_j$$

  where C_1 and C_2 are given constants (with default values 8196 and 0.9, respectively), i is the label of the server where document j has been found, $nref_j$ is the number of references made to document j since it has been brought in the cache, rtt_i and bw_i are estimates of the round-trip delay and of the available bandwidth between server i and the proxy-cache, respectively (see [20] for details on the computation of these estimates). The HYBRID policy removes first the documents in the cache with the smallest costs.

  Advantages: Maintains latency and throughput statistics for each server.

  Drawback: Needs additional data to be kept in memory and more computing. Parameters need also to be tuned carefully. Does not consider last access time.

Several other policies have been proposed in the literature (see [5, 13] among others) but have not been implemented in our simulator. The interested reader is referred to [7] for a more complete survey.

2.3 New Caching Policy: MIX

We conclude this section by introducing a new caching policy, called the MIX policy. Our motivation has been twofold. On the one hand, we try find a policy that takes into account, for every document in the cache, important parameters such as its size ($size_j$ for document j), the number of times it has been referred since it has been last brought in the cache ($nref_j$), the time elapsed since the last reference to that document ($tref_j := date - date\_last\_ref_j$) and the download latency of the last access to that document (lat_j). On the other hand, we would like to find a good tradeoff so that the caching policy is efficient for most important performance measures.

A natural cost function for MIX is typically a function that is nondecreasing in the parameters $size$ and lat and non-increasing in the parameters $tref$ and $size$. Indeed, it seems a priori better to keep in the cache documents that are costly to retrieve (lat is large) and/or often requested ($nref$ is large) and to first remove from the cache documents that have not been accessed for a long time ($tref$ large) and/or that are big ($size$ large).

There are infinitely many cost functions fulfilling the above requirements. The key issue is to find a good tradeoff between these parameters. We propose to use the following function:

$$\frac{lat_j^{r_1} * nref_j^{r_2}}{tref_j^{r_3} * size_j^{r_4}} \qquad (4)$$

for document j. As usual, MIX removes first from the cache documents with the smallest costs.

Tuning the parameters r_i ($i = 1, \ldots, 4$) in (4) is not an easy task. Through many experiments (see Section 3) we have observed that the key parameter is r_1. It indeed turned out in all these experiments that selecting a value for r_1 which is much smaller than the values for r_2, r_3, r_4 significantly increased the hit ratios. We ended up using $r_i = 1$ for $i = 2, 3, 4$ and $r_1 = 0.1$, a choice which yields good performance for each cost in Eqs. (1)-(3). Numerical results comparing MIX with these values for r_i ($i = 1, \ldots, 4$) to LRU, LFU, SIZE, GD-SIZE and HYBRID policies are given in the next section.

3 Trace-Driven Simulations

3.1 About the Simulator

We have developed a simulator (in C++) which models the behavior of a proxy-cache server. We use the approach of trace-driven simulations in which the stream of requests is taken from a log file.

The evolution of a proxy cache is simulated as follows. when a new request arrives in the simulator, it checks the contents of the cache to see whether the referenced document is already in it (according to its URL). If yes, then the cache is left unchanged and the counters of the referenced document is updated, e.g., $nref$ is increased by one, and lat and $date\_last\_access$ are refreshed. Otherwise, the document corresponding to the request is fetched into the cache, pushing out one or more documents according to the caching policy if the cache is full.

Documents in the caches may become obsolete due to changes in the original servers. In order to deal with this, the simulator checks whether a document in the cache, when it is requested, is modified since the last reference. If it is the case, then the request is considered to be a miss and the new version of the document has to be retrieved.

In practice, a proxy-cache of capacity around 30 GBytes can contain several hundreds of thousand documents and can serve several hundreds requests per second. Thus, it is time consuming to invoke the replacement algorithm each time a new document is fetched. Rather, it is useful to remove a significant proportion of documents, when the replacement algorithm is invoked, so as to make enough room for future new requests.

This is for instance the case for Squid[1], the most widely used caching software (freely available for all UNIX systems). There are a lower threshold and an upper threshold in it. As soon as the cache occupancy exceeds the upper one, Squid starts removing documents according to the LRU algorithm until the cache occupancy drops below the lower threshold. More precisely, the LRU algorithm is applied to at most 256 documents to avoid sorting a too long list; the 8 documents with the smallest costs out of these 256 documents are then removed from the cache. This procedure is then repeated until the cache occupancy drops below the lower threshold. This "batch" removal mechanism may however lead to longer "removal sessions" during which the performance of the cache drops dramatically. Hence, the last version of Squid (1.1) features another mechanism aiming at removing files in a "smoother" way. It regularly checks small batches of documents. If the time elapsed since the last request of a document exceeds a threshold (the LRU expiration age), then it is discarded from the cache. This mechanism is carried out in a way that every document in the cache is checked once a day. Furthermore, to keep the cache occupancy between the lower limit and the upper limit, the LRU expiration age is dynamically updated so as to minimize the number of "removal sessions".

[1]http://squid.nlanr.net/Squid/

In view of these practical considerations, our simulator implements caching policies (LRU, LFU, SIZE, GD-SIZE, HYBRID or MIX) in the following way. It uses a double threshold mechanism: when the cache occupancy exceeds the upper threshold, then the simulator sorts all the documents according to the cost function associated with the caching policy (see Section 2), The documents with the lowest costs are removed until the cache occupancy drops below the lower threshold. We use the same values for the lower and upper thresholds (90% and 95%) as in Squid.

With such an implementation, all the above mentioned caching policies have the same time complexity, namely the cost for sorting existing documents in the cache. Note however that the number of parameters used and the cost of computing the sorting criteria are different. It is also possible to implement the policies as in Squid so that whenever the cache has to remove documents, only a small portion of documents (say, 256 documents) is sorted. Again, in such a case, these policies have the same time complexity, but with different coefficient due to computational cost of the sorting criteria. In this regard, the new algorithm MIX and the algorithm HYBRID are the most expensive ones due to the computation of exponentials.

3.2 Simulation Results

We used public trace files available on the Web. Each item in a trace file contains the URL of the document, its size, download duration, and type of the request. Dynamically generated documents (mostly CGI scripts) were removed from the trace files. The following traces were used:

- DEC[2] proxy-cache traces. These traces are from an instrumented version of the Squid proxy, and record all external web requests generated within Digital Equipment Corporation
- BU[3] client traces collected in 1995. These

[2]ftp://ftp.digital.com/pub/DEC/traces/proxy/-webtraces.html

[3]ftp://cs-ftp.bu.edu/techreports/95-010-www-client-traces.tar.gz

traces were collected from the clients at Boston University (each of them were using a modified version of the Mosaic web browser).

- NLANR[4] top-level proxy-cache traces.
- INRIA traces (not public). Traces from our local proxy-cache at INRIA/Sophia-Antipolis (with size 1.2GB, and more than 150 clients)

In these traces the download time is the sum of the time taken to retrieve a document and of the time needed to transmit this document to the client. Since in our traces the clients were connected to the proxy via either a LAN or high speed network (NLANR), the second component of the download time is actually small compared to the first component. Therefore, the download time in this paper is essentially the time required to retrieve document, which is the important parameter.

Table 1 summarizes basic statistics of these traces (length, total size, and the maximum hit rates obtained with an infinite-size cache). The low hit rates observed on the NLANR traces are possibly due to the fact that the corresponding proxy-cache is on a higher level.

Traces	Requests	Mbytes	$W^r_{\max}$	$W^b_{\max}$
Digital	196656	2415	0.33	0.20
BU	79536	1156	0.44	0.34
NLANR	166506	2527	0.20	0.12
INRIA	449928	4895	0.42	0.18

Table 1: Main Statistics of the Traces

In order to compare the performance induced by the different caching policies (LRU, LFU, SIZE, GD-SIZE, HYBRID, MIX) we ran the simulator with different cache sizes ranging from 5% to 35% of the total number of bytes contained in each trace. For all of them, we have plotted the relative request hit rate $W^r/W^r_{\max}$ (resp. the relative request byte hit rate $W^b/W^b_{\max}$ and the latency ratio as defined in Eq. (3)), versus the cache size.

We observe from Figures 2—5 that for the request hit rate MIX and GD-SIZE clearly outperform the other policies. LRU is the worst one. The improvement over LRU is quite significant: the MIX policy with a cache size of 125MB (5%) yields the same result as LRU with 580MB (23%) (NLANR trace, figure 5).

[4]ftp://oceana.nlanr.net/Traces/Caching/

When the byte hit rate is the targeted cost function (Figures 6—9) MIX yields the best results (except for the NLANR trace in Figure 9, where LFU is sometimes slightly better). SIZE and HYBRID behave poorly for this performance measure (Figures 6—9). For small cache sizes MIX yields a 100% improvement over SIZE and HYBRID. MIX always performs better than GD-SIZE (especially for small cache sizes in the BU trace – see Figure 6). It is also worth noting that for a given cache size the relative request hit rate is always higher than the relative byte hit rate. This is especially true for DEC and NLANR traces.

Figures 10—13 display the latency ratio as a function of the (normalized) cache size. It is not surprising that policies which take into account the network latency in their cost function, i.e. MIX and HYBRID, give better results. As observed in [7], GD-SIZE is very good for this measure as well. Nevertheless, our MIX policy is slightly better for every trace used in our experiments. In all the traces, LRU and LFU appear to be the worst policies.

The simulation results indicate that for the three performance measures under investigation, the MIX policy provides the best results in most of the cases. The improvement over LRU for the byte hit rate criterion is relatively marginal but it is rather high for the request hit rate and latency ratio criteria. GD-SIZE has satisfactory performance as well (which confirms results in [7]) but is less efficient for the byte hit rate. HYBRID has better results than LFU, LRU and SIZE for the latency ratio, but behaves poorly for the byte hit rate criterion.

We also observe from the figures that if the cache size is large enough then all policies exhibit the same behavior. Since the disk capacity increases at a very fast rate nowadays, one may argue that, after all, the best thing to do is to use the policy that requires the fewer memory and CPU usage, namely, LRU. Another view is to say that since bandwidth is the critical resource in today's networks it is worth using the best "saving bandwidth policy", namely, MIX. It is worthwhile noticing that Web traffic and the volume of information available on the Web have been increasing exponentially fast, so that careful design of caching policies remains an important issue.

4 Conclusions and Future Work

We have proposed and evaluated the caching policy MIX for Web client caching (proxy-caching). The MIX caching policy takes into account the network latency, the size and the accessed frequencies of the documents as well as the time elapsed since the last referenced to documents in the cache. Through trace-driven simulations we have shown that MIX exhibit uniformly better performance than several known policies for several standard metrics (request hit rate, byte hit rate and latency ratio).

A lot of work still remains to be done both theoretically and experimentally. From a theoretical point of view we need to develop appropriate data access models for Web applications. Such models will be crucial for devising optimal and/or suboptimal solutions of the caching problem. Such models will also be useful for the performance evaluation of caching policies, in addition to trace-driven simulations.

The caching problem has intrinsically large size: the number of documents available in all the Web servers and number of requests served by a proxy-cache are both large. Thus, further research should be made on efficient heuristics, not only for single client cache but also for hierarchical cache systems.

The impact of the hierarchical cache system and, more generally, the architecture of multiple caches, need to be investigated. The interaction between caching policies and the communication protocols (such as multicast protocol) is also an important issue that needs to be addressed.

Last, it will also be interesting to analyze the influence of the HTTP protocol on the caching policy, in particular, the issues related to persistent connections and pipelining.

References

[1] V. Almeida, M.E. Crovella, A. Bestavros, and A. de Oliveira. Characterizing reference locality in the www. In *Proceedings of PDIS'96: The IEEE Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996.

[2] M. Arlitt and C. Williamson. Trace-driven simulation of document caching strategies for internet web servers. Technical report, Dept. of Computer Science, University of Saskatchewan, Canada, 1996.

[3] M. Arlitt and C. Williamson. Web server workload characterisation: The search for invariants. In *Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Philadelphia, May 1996.

[4] D. Bertsekas. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, Massachusetts, 1996.

[5] J-C. Bolot and P. Hoschka. Performance engineering of the world wide web: Application to dimensionning and cache design. In *Proc. of the 5th WWW Conference*, Paris, May 1996.

[6] H-W Braun and K.C. Claffy. Web traffic characterization: An assessment of the impact of caching documentts from ncsa's web server. *Computer Networks and ISDN Systems*, 28:37–51, 1995.

[7] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pages 193–206, December 1997.

[8] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical internet object cache. Technical Report 95-611, Computer Science Department, University of Southern California, Los Angeles, California, March 1995.

[9] A. Cormack. Caching on janet - acn report. Technical report, University of Wales, Cardiff, 1996.

[10] J. Gettys, J. Mogul, R. Fielding, H. Frystyk, and T. Berners-Lee. Hypertext transfert protocol http/1.1. RFC 2068, W3C, 1997.

[11] S. Hosseini. New results on generalized caching. Technical Report WUCS-96-25, Washington University in St. Louis, 1996.

[12] S. Hosseini and J.R. Cox. Optimal solution of off-line and on-line generelized caching. Technical Report WUCS-96-20, Washington University in St. Louis, 1996.

[13] P. Lorenzetti and L. Rizzo. Replacement policies for a proxy cache. Technical Report LR-960731, Univ. di Pisa, 1996.

[14] J.C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta-encoding and data compression for http. In *Proceedings of the ACM SIGCOMM '97*, Cannes, France, September 1997.

[15] H. Frystyk Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, and C. Lilley. Network performance effects of http/1.1, css1, and png. In *Proceedings of the ACM SIGCOMM '97*, Cannes, France, September 1997.

[16] A.J. Smith. Bibliography on paging and related topics. *Operating Systems Reviews*, 12:39–56, October 1978.

[17] A.J. Smith. Second bibliography for cache memories. *Computer Architecture News*, 19(4), June 1991.

[18] C.J.H.H. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, Cambridge, England, 1989.

[19] S. Williams, M. Abrams, C. Standridge, G. Abdulla, and E. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of the ACM SIGCOMM '96*, Stanford University, 1996.

[20] R. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *Proceedings of the 6th World Wide Web Conference*, Santa Clara, California, 1997.

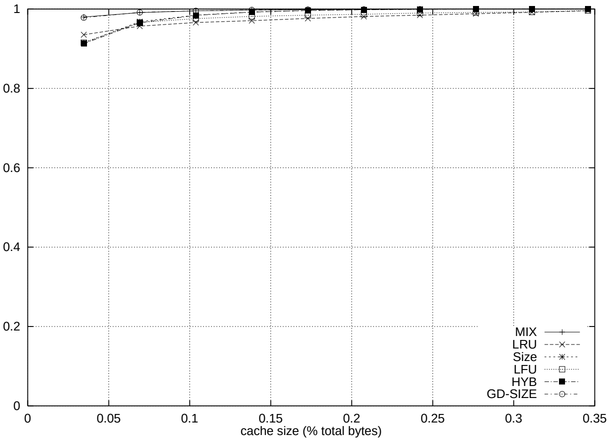

Figure 2: *BU traces, Relative Request Hit Rate*

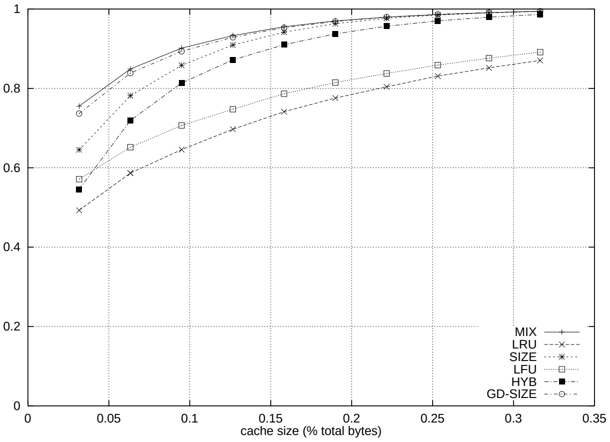

Figure 5: *NLANR traces, Relative Request Hit Rate*

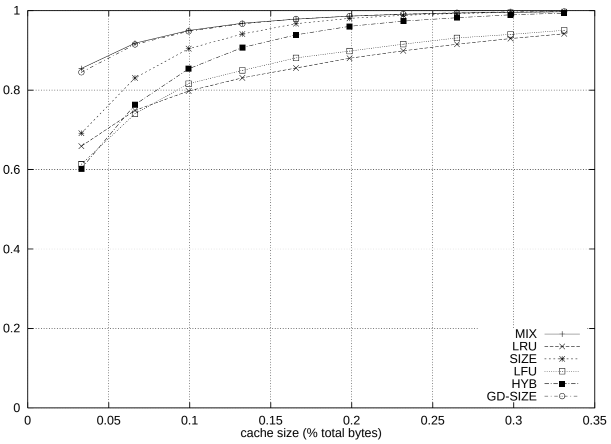

Figure 3: *DEC traces, Relative Request Hit Rate*

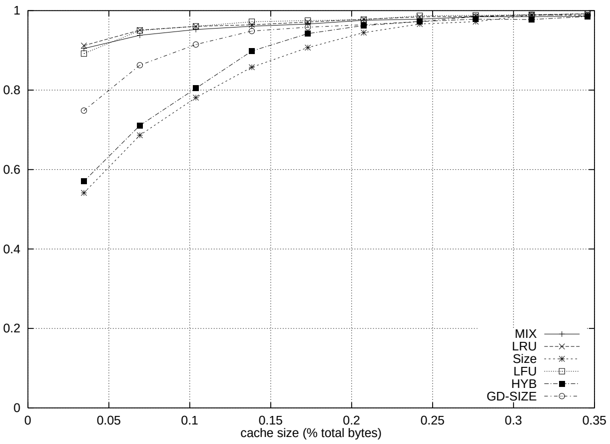

Figure 6: *BU traces, Relative Byte Hit Rate*

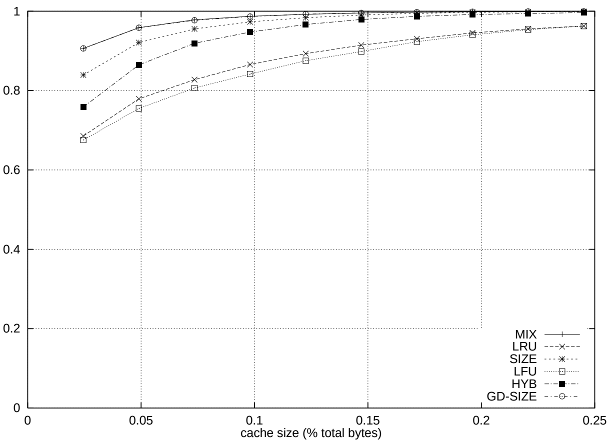

Figure 4: *INRIA traces, Relative Request Hit Rate*

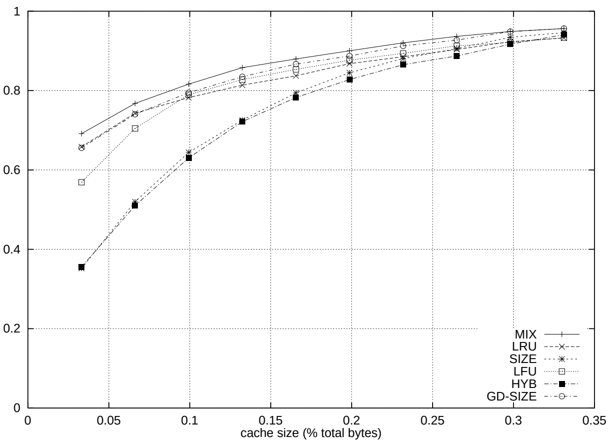

Figure 7: *DEC traces, Relative Byte Hit Rate*

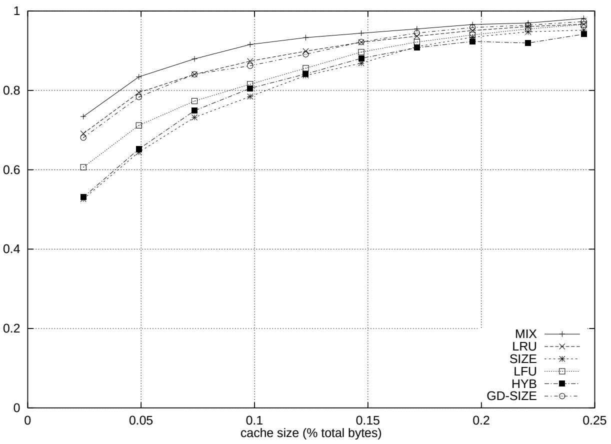

Figure 8: *INRIA traces, Relative Byte Hit Rate*

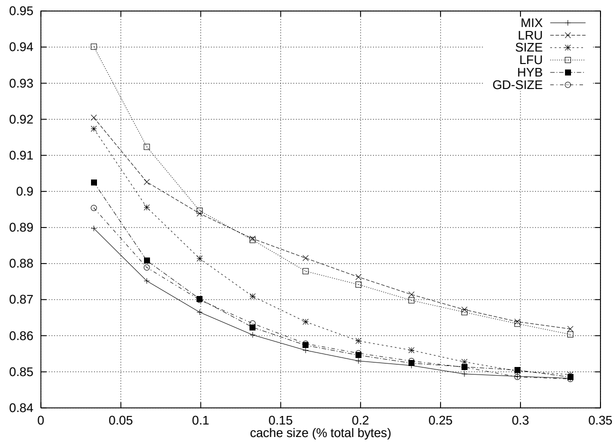

Figure 11: *DEC traces, Latency Ratio*

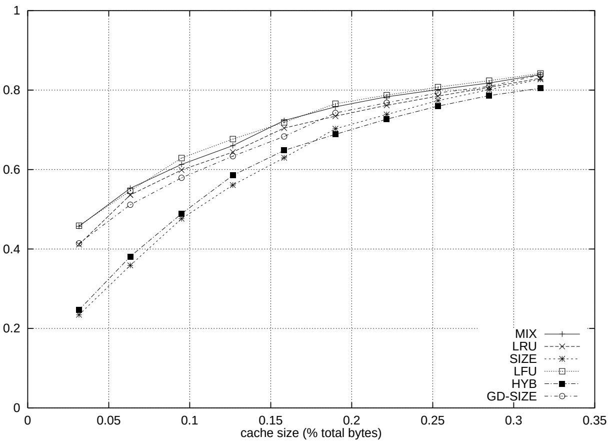

Figure 9: *NLANR traces, Relative Byte Hit Rate*

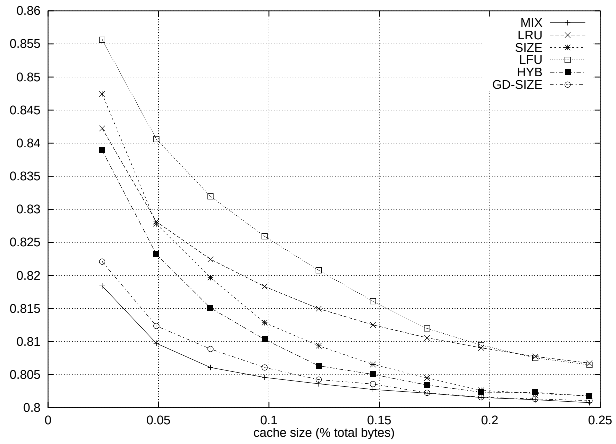

Figure 12: *INRIA traces, Latency Ratio*

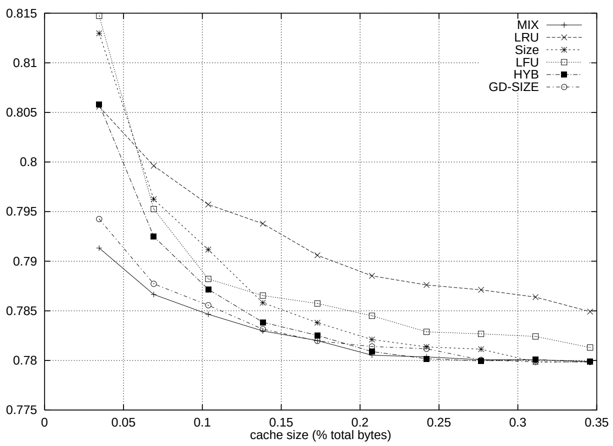

Figure 10: *BU traces, Latency Ratio*

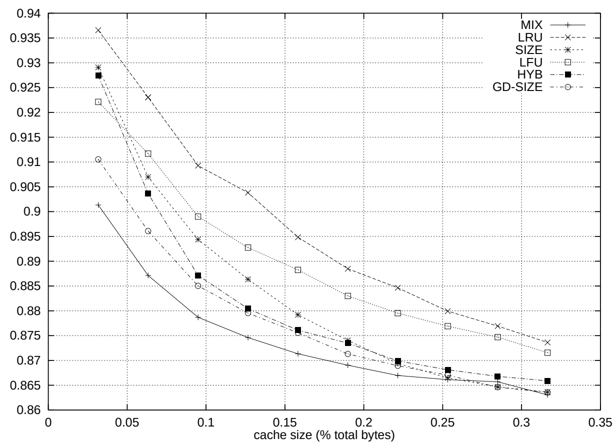

Figure 13: *NLANR traces, Latency Ratio*