Alcom-FT Technical Report Series
ALCOMFT-TR-01-41

Integrated Prefetching and Caching for the World Wide Web via User Cooperation

Massimiliano Curcio* Stefano Leonardi * Andrea Vitaletti *

March 9, 2001

Abstract

Caching and prefetching have often been studied as separate tools for reducing the latency observed by the users in the access to the World Wide Web. The goal of this work is to show the potentialities and limitations of integrating Caching and Prefetching for improving the performances of web navigation. We propose a new prefetching technique that uses a limited form of user cooperation to establish which documents to prefetch in the local cache at the client side. We show that our prefetching technique is highly beneficial only if integrated with a suitable caching algorithm. We consider two caching algorithms, Greedy-Dual-Size [4, 16] and Least Recently Used, and demonstrate on trace driven simulations that Greedy-Dual-Size with prefetching outperforms both LRU with prefetching and a set of popular caching algorithms. We also develop a prototype of a proxy implementing our prefetching/caching technique to be used with a web browser.

1 Introduction

A key problem of the current Internet is the latency experienced by the users in the access to the World Wide Web. User's latency indicates the time interval between the request for a document till the response arrives. Document caching and prefetching are the most widely applied techniques for latency reduction.

Caching web documents aims to store frequently accessed documents in a position closer to the user. A first common form of caching is at HTTP proxy servers. A HTTP request is first sent to the proxy server. If the requested document is served from the proxy cache, this is directly sent to the client, otherwise the proxy retrieves the page from the server and then sends it to the client. A second common form of caching is at the client side. The local cache is managed by a Web browser to store more frequently accessed documents. Serving the requests at the local cache may take 10 milliseconds compared with 500 milliseconds average TCP/IP round trip time necessary to open a connection with a remote server. Caching at the client side is even of higher importance for terminals of low bandwidth and memory cabability like cellular phones or palmar mobile terminals.

A caching policy is defined by a page replacement rule that selects, when the cache is full, the document to evict. Caching Web documents differs in many aspects from traditional caching. In traditional caching pages have uniform size and every page fault has a uniform cost. The objective is often to reduce the number of page faults through the exploitation of locality of reference in the request sequence. Differently from traditional caching, Web caching deals with a dramatic difference

*Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", via Salaria 113, 00198-Roma, Italia. Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and IST-2000-14084 (APPOL), and by the MURST Projects "Algorithms for Large Data Sets: Science and Engineering" and "Resource Allocation in Computer Networks". E-mail: {leon,vitale}@dis.uniroma1.it

from document to document in size and retrieval cost (that are even not necessarily related). The goal of web caching, is not limited to the reduction of cache misses. Web caching also aims to reduce user perceived latency, that depends from the retrieval cost of the documents that are not cached, and network load, the amount of traffic generated from requests not served from cache. There are also some limitations to caching. Caching applies only to static documents, that is however still a good share of the total. Moreover cached documents must be retrieved again if expired. This may be checked at regular intervals or at the time the document is requested by sending a HEAD command to the Web server.

Traditional caching strategies like Least Recently Used (LRU), that do not consider document size or retrieval cost, are also widely used in proxy caching (e.g. Squid [1]). Very natural criteria to employ to design more efficient web caching algorithms are to evict documents of big size or low retrieval cost. Cao and Irani [4], and independently Young [16], proposed a simple and elegant Cache Replacement algorithm, *Greedy-Dual-Size* (GDS) that tries to evict a page with a combination of low retrieval cost, large document size and reference further in the past. Cao e Irani [4] also make an extensive comparison of Web caching algorithms on trace driven simulations and observe that Greedy-Dual-Size behaves better than other page replacement algorithms in terms of hit rate, reduced latency and network load. A theoretical analysis proving the efficiency from a worst case analysis point of view of Greedy-Dual-Size can be found in the work of Young [16] and of Cao and Irani [4].

Document prefetching is a second effective tool for improving the access at the World Wide Web. Prefetching aims to bring documents at the client side before they are requested. Prefetching can be initiated either at the client side or at the server side (in this last case prefetching is often denoted by pushing). A key ingredient of a prefetching technique is a document selection policy that rules the choice of which document to prefetch. Different prefetching selection policies have been proposed in literature. For instance in [9] some (or all) the links in the document requested by the user are also sent to the client. In [6] and [13] the clients gather information from the server on the most popular links to decide which documents to prefetch, while Bestavros [2] implemented a system in which the server responds to a request with both the requested page and a list of related resources. Mogul and Padmanabhan describe a similar method of prefetching using server suggestions [14] in which the server predicts the next access on the basis of the weight of a dependency graph, while the client decides whether or not to actually prefetch the page. In [15]the authors implemented a deterministic static client initiated prefetching policy, useful to prefetch daily used pages like news or meteo forecasts.

Web prefetching may potentially lead to a consistent increase of network congestion if massively used to download documents that are not used by the client. However this is not the case when prefetch happens on the access network through a dedicate communication link like for modem users. Fan, Jacobson, Cao and Lin [7] consider server initiated prefetching between proxy and low bandwidth modem users. The prefetching technique uses a page predictor at the proxy cache that decides which pages to prefetch on the basis of the analysis of the most recently requested pages. The authors show how prefetching can lead to a saving of up to about 14.6% of the latency, and of 28,6 % if combined with Delta encoding. They assume the LRU caching replacement strategy at the local cache. In their model prefetching may happen even before the user has requested a document for the first time.

Less work has been done on the integration of caching and prefetching techniques. Kroeger, Long and Mogul [11] study the combined effect of caching and prefetching on end user latency. They conclude that caching alone may save user latency up to 26%, while the combined effect of caching and prefetching can lead up to 60% latency reduction. These figures must be intended as a tentative to establish an upper bound on the performance improvement of prefetching and caching in a ideal framework where memory cache is unlimited and full knowledge about the documents that will be

[1] http://www.squid-cache.org

requested is assumed. In [8] the authors investigate the use of the dependencies between resources in making cache replacement decisions. In particular combining knowledge of client behaviour with resource dependencies can be useful to reduce latency by prefetching selected resources.

Prefetching can also be limited to some of the activities involved in the retrieval of a web document. In the retrieval of web pages there are four different contributions to the latency experienced by the user. The first contribution is DNS look up to resolve the address of the server, then establishing TCP/IP connections with the remote server, start of session delay at HTTP servers, and finally the transfer of the content of the document. Cohen and Kaplan [5] analyzed the impact on Web latency given by each of the four contributions and proposed a set of tools for latency reduction involving *prefetching of TCP/IP connections*, pre-solving DNS queries, pre-warming web servers by sending dummy HEAD requests. While all these tools are very relevant and can fruitfully interplay, we stress that for most low bandwidth users prefetching the content of web pages is still a fundamental tool for enhancing the performances of the system.

Differently from previous works, we concentrate on the combined and integrated application of prefetching and caching strategies. Integrated prefetching and caching has recently received a growing attention in the attempt of enhancing the performances of local data base and file systems. Cao, Felten, Karlin and Li [3], proposed prefetching/caching strategies to optimize the access memory blocks stored on magnetic disks. The model considers equal block size and block fetching time from disk to cache. They propose and analyze, both theoretically and empirically, different strategies to decide the time at which the prefetching of the next referenced block not in cache must be initiated. An optimal eviction rule can be implemented at time of prefetch since the whole input sequence is known to the algorithm. In a later work Albers, Garg and Leonardi present for the model introduced in [3] a first polynomial time prefetching/caching algorithm that achieves the optimum elapsed time. Such strategies have also been extended to parallel disk systems, where more blocks can simultaneously be fetched [1] [10] from disks to cache.

Our Contribution. In this work we propose a new technique to integrate prefetching and caching for the access to Web documents. Our attention is focused at the client side. The core of our proposal is the definition of a document selection policy to determine which documents to prefetch and of a technique of integration between prefetching and caching. Although our prefetching technique can be integrated with any caching policy, we will concentrate on two specific caching algorithms, Least Recently Used and Greedy-Dual-Size, the first being the most diffused caching strategy, the second being recently shown to be very effective for caching web pages [4].

We compare our proposed prefetching/caching techniques on trace driven simulation with several popular caching algorithms. The conclusion we draw is that the integration between prefetching and caching is very effective only if a suitable caching algorithm adopted. In particular we demonstrate that the integration with Greedy-Dual-Size outperforms the integration with LRU and all the caching algorithms we consider.

The document selection policy we propose relies on a simple form of cooperation between user and system to gather a partial knowledge of the documents that will be requested in the future, while it does not involve any complex prediction mechanism. The user is simply asked to *mark* a page when abandoned for a newly requested page if it intends to request the page again in the future. Knowing that a page will be requested is a valuable information. However, this does not say when the page will actually be requested, i.e. when the page can conveniently be prefetched. For this purpose we exploit locality of reference. All marked pages that are not in cache, i.e. they have been evicted after the last reference, are organized in a priority queue ordered by the time the pages have been referenced last. *The next page to be prefetched is then the most recently used marked page that is not in cache.*

The system only knows that a document will be requested, not the time of the request a valuable information to decide when the document should be prefetched. We then exploit locality of reference to decide when to prefetch a document. The intuition behind the goodness of this strategy is that this is an optimal choice for selecting the page to prefetch if the user follows a Depth First Search of a hyperlinked structure, i.e. the user follows a hyperlink or uses the back button. We believe that this behaviour approximately represents a typical browsing session of a user searching for an information in a collection of web sites, visiting a portal, or accessing a hyperlinked hyrarchical structure, like news, entertainment, restaurant listing, stock quotation etc.... Of course the form of cooperation we expect from the user is an ideal one, we assume that, in his interest, he will always say the truth. However we believe that even with only a partial cooperation from the client our system would improve system performances.

We also consider that only a fraction of the documents are cacheable and that expired documents must be retrieved again from the server.

We incorporate our integrated prefetching/caching policy into the Web caching simulator developed by Pei Cao and Gideon Glass and used by Cao and Irani [4] for comparing web caching algorithms. We then perform trace driven comparisons between the prefetching/caching algorithms we propose and the web caching algorithms implemented in the simulator. The comparison is done on WWW traces from Berkeley Home IP users used by Cao et al. [7] for experimenting proxy initiated prefetching techniques [2], and traces collected daily within the NLANR project [3].

User/system cooperation can be performed very efficiently in practical system. The document can be marked while waiting for the retrieval of the page requested next. In Section 6 we describe a prototype of the integration of this mechanism in a WWW browser. A small window appears for a limited number of seconds when the user abandons a Web document, allowing either to mark the page or ignore it and then leave the page unmarked.

The increase of network load is a major concern for prefetching applications. However, this is very limited if most pages are prefetched from a proxy server near the access point to the internet. Moreover increasing the cache hit rate has also a positive spin-off in the net reduction of the overall network traffic generated for retrieving documents that are not served from local cache. The prefetching/caching strategy we propose is particularly suitable for modem or mobile users connected through a dedicated telephone link. The time the connection link is idle can be fruitfully used to prefetch those pages that could possibly be requested in the future. Wireless mobile users might benefit from prefetching in different ways. Prefetching improves service availability by allowing off-line navigation when the terminal is disconnected. On the other end prefetching may lead to saving connection costs by maximizing link utilization to retrieve documents that will be accessed later in an off-line navigation.

Structure of the Paper. In Section 2 we present our integrated prefetching/caching technique. In Section 3 we describe the web caching simulator and the caching algorithms we compare with. In Section 4 the traces used in the simulation are presented. Section 5 describes the results of the simulation while Section 6 presents a prototype of the system. The whole set of diagrams describing the results of the simulation are in Appendix .

2 Our Integrated Prefetching/Caching Technique

Our document selection policy is based on a form of cooperation between the user and the system. Every time a document is abandoned the user is asked to mark the document if it will be requested again in the future. In that case the document is said *marked*, otherwise it is said *unmarked*. The integration of the prefetching technique with any caching algorithm is done as follows:

[2] http://www.acm.org/sigcomm/ita/
[3] http://www.nlanr.net

1. A requested document is retrieved from the server if either not cached or if cached but expired.
2. A cacheable document is cached if marked by the user.
3. A document retrieved from cache or from the server is discarded if not marked.
4. The eviction rule is invoked when the overall size of cached documents exceeds the size of the cache. Observe that even the document cached last can in principle be evicted.
5. The document that is prefetched is the most recently used non-cached document that is marked. Observe that all non-cached marked documents have been previously evicted from cache.

In Figure 1 we give a formal description of the algorithm. The set of documents that are marked but not cached are organized in a priority queue ordered by time of last reference to the document. The document referenced most recently in the past is then selected for prefetching.

We consider two caching algorithms, LRU and Greedy-Dual-Size. *We denote by GDSP the integration of prefetching with Greedy-Dual-Size and by LRUP the integration of LRU with prefetching.*

The page replacement rule of LRU consists of evicting the page accessed furthest in the past. For this purpose, LRU orders all cached pages in a list. The initialization phase of LRU consists in bringing a cached document at the top of the list. The eviction policy of LRU selects the document at the rear of the list. Observe that if we integrate the prefetching policy with LRU, the priority queue use by the document selection policy can be simply replaced by a stack.

Greedy-Dual-Size [4] [16] considers the retrieval cost, the size of the document and temporal locality in the eviction rule. Let c_p be the retrieval cost of document p and let s_p be the size of document p. A value H_p is associated with every page p in cache. When a page is cached, or it is retrieved from cache, H_p is set to c_p/s_p. When the cache is full, a page with $H_p = 0$ is selected for eviction, if any. Otherwise we compute the minimum ratio $r = min_p H_p$ and reduce every H_p by r. At least one page p will have $H_p = 0$ after this operation is performed. Figure 2 describes the initialization phase and the eviction phase of Greedy-Dual-Size. In our implementation the documents that are cached and the documents in PQ candidated for prefetching are also stored in a linked list.

3 The Web caching simulator

We integrate Greedy-Dual-Size with Prefetching and LRU with Prefetching within the web cache simulator designed by Pei Cao and Gideon Glass [4] used to compare on HTTP traces several caching strategies. We therefore simulate the marking of the pages operated by the user by traversing the trace and observing which pages are requested in the reminder of the sequence. The data structures used in the simulator are also modified to include the information required for our prefetching/caching algorithms to work [5].

All distinct web objects requested by a user within a trace are stored in a vector of records. The record stores the page ID of the object, its size, its retrieval cost, its “last modified” time, its “user's mark” attribute value, its “cacheable” attribute value, the number of requests for this object so far and the UNIX timestamp of the current request.

We compare our prefetching/caching algorithms with several of the most effective web caching strategies, in particular those experimented in [4] and implemented in the simulator. We consider the following web caching algorithms, in addition to Greedy-Dual-Size and LRU already described in the previous section:

[4]The source code of the simulator (in its original form) is available at : ftp://ftp.cs.wisc.edu/pub/cao/webcache-simulator.tar.Z

[5]The new web cache simulator is available at the following URL: http://www.dis.uniroma1.it/..... .

LRU Threshold. This algorithm is a modified version of LRU in which documents larger than a given threshold are never cached.

Size. The eviction rule selects the cached document of biggest size. The objective is to avoid the replacing of a large number of small documents with a document of large size.

Hybrid: The idea of *Hybrid* is to incorporate an estimation of the current network conditions into the caching algorithm. For each document i, the algorithm calculates a function of the connection time to the server storing i, $clat_{ser(i)}$, the bandwidth of the network link $cbw_{ser(i)}$, the document size s_i and the number of references to document i since the last time it was cached $nref_i$. The function is

$$(clat_{ser(i)} + W_B/cbw_{ser(i)}(nref_i^{W_n})/s_i,$$

where W_B and W_N are constants. The document that is evicted is the document in cache that minimizes the above function.

Least Relative Value (LRV). The value of a document, is a function of the probability P_r that the document is accessed again. The *LRV* algorithm selects the document with Lowest Relative Value as the most suitable candidate for replacement. In [12] it is shown how P_r can be efficiently calculated on-line.

4 HTTP traces

As the results from a trace-driven study strongly depend on the traces, we gathered two meaningful set of web traces:

- Berkeley University HomeIP proxy traces [6] (1- 19 Nov 1996), containing four sets of traces, for a total of 5,816,666 registered access from 7,714 users (after pre-processing).
- NLANR daily proxy traces from IRCache group [7]: we considered traces from 1 day of HTTP activity (9/20/2000), for a total of 981,354 registered access from 142 users (after pre-processing).

Berkeley traces show an average number of about 7800 requests per user trace, of which, on average, 16% is non-cacheable. The average number of requests that are referenced only once is about 3600, the so called *one timers*. Berkeley traces are also characterized by a quite large number of documents of big latency due to the slow speed of the modems used from users to connect, from 4 kb to 24 kb.

NLANR traces present a number of about 17720 requests per user, with 15000 distinct requests, 14300 one timers and a percentage of about 31% of non-cacheable requests. These traces contain a much larger number of one timers with respect to Berkeley traces and therefore we expect that the benefit of prefetching and caching will be limited in absolute terms.

We performed some necessary preprocessing over the traces. We considered only HTTP requests, and we left out all the requests with essential informations (such as timestamp) missing and all requests with HTTP commands different from GET and POST. The data in the traces have been converted to the format required by the simulator.

We sorted traces by timestamp and pasted together all files containing a single user's browsing session, obtaining a single file for each user. This doesn't "misrepresent" user's behaviour, because we assume the use of persistent caching (files remain in cache between consecutive browsing sessions, unless user explicitly deletes them).

[6] http://www.acm.org/sigcomm/ita/

[7] http://www.nlanr.net

All requests for URLs whose pathname contains the string "cgi-bi" or the character '?', meaning that server's response is a dynamic page or the result of a query submitted by the user, have been considered non-cacheable by setting to 0 cacheable field, while in all other requests we set this field to 1. We also set the cacheable field to 0 for all the requests with HTTP status different from 200, 203, 206, 300, 301. With a simple modification to the simulator's source code we made it to control this field's value before deciding to insert or not the page currently requested by the user.

A critical issue is the simulation of user's decisions to mark or not to mark the requested pages. The user's decision is simulated by setting to 1 or to 0 a dedicated field of the record corresponding to that request. A simple but quite effective way to simulate this decisional mechanism is to use our "foreknowledge" of the whole sequence of user's requests, obtained by parsing the traces file, to find the web object that have been requested more than once by the user after the first access.

5 Simulation results

In this section we present the results of the simulation of the prefetching and caching algorithms on the set of HTTP traces described in the previous section. We measure the performances of our algorithms in terms of the three following metrics:

- Hit rate: the number of requests that hit in cache as a percentage of the total number of requests.
- Byte hit rate: the number of bytes that hit in cache as a percentage of the total number of bytes requested.
- Reduced latency: the percentage of total retrieval time saved by the algorithm, i.e. the ratio between the overall retrieval time (from servers) of documents served from cache and the overall retrieval time of documents retrieved from servers.

Algorithms GDSP and LRUP are evaluated on these three metrics together with the caching algorithms described in Section 3, namely LRU, $LRU - Threshold$, $Size$, $Hybrid$, GDS and LRV.

We also partition the set of traces by their length. Intuitively, longer sequences should allow to gather more information about the documents requested in the future by the user. For this purpose, we distinguish between traces with a number of requests in the range 5,000-10,000, 10,000-25,0000 and 25,000-50,000.

In appendix A we show the results of the simulation. In the following we give a brief description of these results mostly referring to Berkeley Traces. Similar observations can be done for NLANR traces even though the figures are quite different given the different nature of the traces.

As a benchmark for our comparison we consider an *optimal algorithm* having cache size sufficient to store all the distinct documents in a trace. The optimal algorithm is charged with the retrieval cost of a document only at (1.) the first reference of a document; (2.) any later reference of a document if expired (3.) any later reference of a document if not cacheable. The simulation has been carried out by monitoring each algorithm on different cache memory sizes, expressed as percentage of the *optimal cache size*. This means that the performance of any algorithm converges to the optimum with the increase of the percentage of the optimal cache size that is available.

The goal of our simulation is therefore to establish the rate of convergence of the various algorithms with the increase of the cache size for each of the three metrics. However, in the diagrams shown in appendix A, we restrict our attention, for reasons that will be clear soon, up to 20% of the optimal cache size.

The most remarkable result is that GDSP converges to the optimum hit rate and byte hit rate with about 5% of the optimal cache. GDSP also converges to the optimum reduced latency with 5%

to 10% of the optimal cache on sequences of medium and long size. All other algorithms converge to the optimum with more than 20% of the optimal cache.

For example, in Figure 9, the value of byte hit rate converges to 22% with 5% of the optimal cache, while all other algorithms are still below a byte hit rate of 10% with more than 20% of the optimal cache. A similar behaviour is observed for hit-rate and reduced latency. For example in Figure 5 GDSP converges to a hit rate of 27% with 3% of the optimal cache, in Figure 11, to a reduced latency of about 14% with 10% of the optimal cache. We stress that the values of convergence are very close to the optimum, that is with 100% of the optimal cache size.

It might appear very surprising that GDSP achieves the optimum with only 5% of the optimal cache. This is explained with the large number non-cacheable documents and one-timers in the traces. In Berkeley traces these two kind of documents are about 70% of the total number of requests, even higher figures are reached for NLANR traces. The reason for this behaviour is that the optimal cache size also includes the size of non-cacheable documents and one-timers, while GDSP and LRUP only caches those documents requested again in the reminder of the sequence. This also explains as in many cases LRUP converges faster than all other caching algorithms, even if to a value close to that reached by the other caching algorithms in the interval up to 20% of the optimal cache.

We would also like to comment on the absolute values of our metric of performances. For instance GDSP converges in Figure 5 to a hit rate of about 28% with about 5% of the optimal cache. This should be interpreted as follows. In that sequence only about 40% of the requests are cacheable. GDSP does not hit the first request to a cacheable document but, as the optimum does, retrieves from cache almost all the later references to cacheable documents. In this specific example, we observe that LRUP and all the other caching algorithms hit at most 15% of the requests with 20% of the optimal cache, therefore only about 50% of the requests that are served from cache from GDSP and the optimal algorithm.

All the considerations made for Berkeley traces can also be applied to NLANR traces. For NLANR traces we observe that the absolute values for the three metrics are much smaller than for Berkeley traces. For example, for sequences with 5,000 - 10,000 requests, GDSP approaches a hit rate of about 6% with 2% of the optimal cache (see Figure 13), while in Berkeley traces, for sequences of length in the same range, a hit rate of 18% is reached. Of course the explanation is in the bigger number of non-cacheable documents and one-timers with respect to Berkeley traces. The effect of a large number of these two kind of documents is twofold: 1) it decreases the cache size needed to converge to the optimum, since by definition the optimal algorithm caches all the pages; 2) decreases the optimal performances, since only a smaller percentage of the documents can be retrieved from cache.

We finally summarize in the following the four main preliminary conclusions we draw from the empirical results that are presented in the Appendix. A more analytical comment of the figures is postponed to the Appendix.

1. Our prefetching technique is highly beneficial to improve the performances of the system if integrated with a suitable caching algorithm. Greedy-Dual-Size with Prefetching (GDSP) outperforms Least Recently Used with Prefetching (LRUP) and all the caching algorithms. This more remarkable on hit rate. The reason for this behaviour is that, differently from LRU, Greedy-Dual-Size tends to evict documents of small retrieval cost that can be easily prefetched later in the future. On the other end LRUP keeps in cache documents of large size at the expenses of the eviction of a large number of small documents.

2. GDSP outperforms all other algorithms also in terms of reduced latency and byte-hit-rate. This advantage is however less remarkable than in terms of hit-rate. The reason for this is that popular documents have often smaller retrieval cost and size, while large documents, whose

contribution to latency and network load is higher, are less frequently referenced in the sequence and therefore are weaker candidates for prefetching.

3. One-timers are never cached by our prefetching/caching algorithms. This results in a better use of the cache that allows GDSP, and partially also LRUP, to converge much faster than other algorithms. Due to the logarithmic shape of our metric of performances, this allows GDSP a dramatic saving of cache memory with respect to all other algorithms.

4. The advantage over pure Caching strategies increases on longer traces. This is very remarkable for instance in Figures 4, 5, 6, representing the hit rate on traces of $5,000 - 10,000$, $10,000 - 25,000$ and $25,000 - 50,000$ requests, where LRUP, the best of all other algorithms, achieves respectively about 80%, 50% and 25% of the hits of GDSP. This is not surprising since a more extensive information on the documents requested by the user later in the sequence allows to better exploit the potentiality of prefetching. This suggests that the performances of a similar system will improve together with its deployment on use.

6 Integrating prefetching/caching algorithms in a browser

Our prefetching/caching technique has been incorporated in a local proxy application to be used with a commercial Web browser. The Web browser is configured to send all HTTP requests directly to the proxy application. If the requested object cannot be served from the local cache, the request is directed to the remote Web server, the reply is returned to the web browser. A configuration tool is available to select several parameters for the proxy application such as cache size and caching policy, e.g. GDSP, LRUP, GDS, LRU, etc....

The cooperation between user and system happens when the requested document is visualized in the browser's window, a small pop-up dialog window (shown in Figure 3) prompts the user to click on its "OK" button (within a 20 seconds countdown) if he/she means to see that particular document again in the future. In case of positive reply, the user may also ask to not show the dialog window in the future and/or for a periodic refresh of the document's cache copy by clicking on the button corresponding to the desired refresh frequency (4 options are available, from every 15 minutes to every 12 hours); in both cases, the Web document and its embedded objects will be "marked" and inserted in cache. Otherwise, the page will not be inserted in cache at all, or it will be removed from cache if it is a local copy.

The task of prefetching a document, accordingly to the algorithm described in Section 2, is demanded to a different thread, not dealing with the HTTP requests of the user. The document is then inserted in cache after its retrieval. The task of periodically refreshing at regular intervals some of the cache entries is also executed from a separate thread, always searching for those cache entries whose time-to-live-like attribute has expired. The corresponding documents are removed from cache and their URLs inserted in the list of marked pages evicted from cache. A prefetch of the original document from the Web server is then initiated when prescribed and an updated copy of the document is inserted in cache.

References

[1] S. Albers, N. Garg, and S. Leonardi. Minimizing stall time in single and parallel disk systems. *Technical Report MPI-I-97-024, Max-Planck Institute for Computer Science, Saarbruecken, Germany*, 1997.

[2] Azer Bestavros and Carlos Cunha. A prefetching protocol using client speculation for the world wide web. *Technical Report BUCS-TR-1995-011, Boston University, CS Dept, Boston, MA 02215*, April 1995.

[3] P. Cao, E. Felten, A. Karlin, and L. Li. Implementation and performance of integrated application-controlled file caching, prefetching, and disk scheduling. *ACM Transactions on Computer Systems, 14(4)*, November 1996.

[4] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. *Proc. USENIX Symp. on Internet Technologies and Systems*, pages 193–206, December 1997.

[5] E. Cohen and H. Kaplan. Prefetching the means for document transfer: A new approach for reducing web latency. *Proc. of IEEE INFOCOM*, 2000.

[6] Dan Duchamp. Prefetching hyperlinks. *Proc. of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99), Boulder, CO, USA*, 1999.

[7] L. Fan, P. Cao, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: Potential and performance. *Proc. of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99), Atlanta, GA*, May 1999.

[8] J. Hine, C. Wills, A. Martel, and J. Sommers. Combining client knowledge and resource dependencies for improved world wide web performance. *Proc. of the INET '98 Conference, Geneva, Switzerland.Internet Society.*, July 1998. http://www.mcs.vuw.ac.nz/ hine/INET98paper/index.htm.

[9] Ken ichi Chinen and Suguru Yamaguchi. An interactive prefetching proxy server for improvement of www latency. *Proc. of the Seventh Annual Conference of the Internet Society (INET'97), Kuala Lumpur*, July 1997.

[10] T. Kimbrel and A. Karlin. Near-optimal parallel prefetching and caching. *Proc. 37th IEEE Annual Symposium on Foundations of Computer Science*, pages 540–549, 1996.

[11] T. Kroeger, D. Long, and J. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. *Proc. of the USENIX Symposium on Internet Technologies and Systems (USITS), Monterey, CA*, December 1997.

[12] P. Lorenzetti, L. Rizzo, and L. Vicisano. Replacement policies for a proxy cache. *draft, http://www.iet.unipi.it/luigi/caching.ps.gz.*, 1996.

[13] E. Markatos and C. Chronaki. A top-10 approach to prefetching on the web. *Proc. of the INET 98 Conference*, 1998.

[14] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review*, July 1996.

[15] Z. Wang and J. Crowcroft. Prefetching in world wide web. *Proc. IEEE Global Internet 96*, November 1996.

[16] N. Young. On-line file caching. *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms.*, pages 82–86, January 1998.

A APPENDIX

In this section we presents the results of the simulation. First we give a brief description of the results, then we show the graphics of the simulation divided by performance metrics (namely hit rate, byte hit rate and latency) and input traces (i.e 5000-10000, 10000-25000, 25000-50000 requests respectively for Berkeley and NLANR traces). See section 4 for a description of the traces, and section 5 for an interpretation of the simulation results.

A.1 Berkeley traces

Hit Rate. For users with 5000-10000 requests (see Figure 4), GDSP converges to the optimum hit rate of about 18% with a cache memory size of about 5%. LRUP converges to a hit rate of about 15% with cache size of about 5%, where all other algorithms are still below a hit rate of 14 %. For users with 10000-25000 requests (see Figure 5), GDSP converges to the optimum hit rate of about 28% with 4% of the optimal cache, while all other algorithms, LRUP included, are below 15 % even with a cache size of 20 % of the optimum. For users with 25000-50000 requests (see Figure 6), GDSP reaches the optimum performances with 3% of the optimum cache, while all other algorithms are still below 10 % with 20 % of the optimal cache size.

Byte Hit Rate. The rate of convergence to the optimal solution of the performances of GDSP and LRUP in terms of byte hit rate is very similar to that in terms of hit rate. Even for byte hit rate it is possible to observe a consistent improvement in the performances of GDSP while the length of the traces increase (see Figure 7,8,9).

Reduced latency. The rate of convergence of reduced latency is slower than that of the previous two metrics. Reduced latency is limited in these traces since there are several one timers and documents of big size. Reduced latency for GDSP is at about 8% with 10% of the cache for users with 5000-10000 requests (See figure 10) but still far away from convergence that is reached for value of the cache greater than 15%. For users with 10000-25000 requests (See figure 11), GDSP converges at the optimum reduced latency of 14 % with 10 % of the optimal cache, while for users with 25000-50000 requests (See figure 12), it converges to the optimum, about 17 % with 10 % of the optimal cache. Observe that all the other algorithms, LRUP included, are below 10% in traces with 10000-25000 requests, and 6% in traces with 25000-50000 requests, while in in traces with 5000-10000 requests, LRUP is quite close to GDSP, and also the other algorithms are at most 2% below.

A.2 NLANR traces

hit rate. GDSP converges to the optimal hit rate of about 6% for users of 5000-10000 requests (See figure 13) with about 2% of the optimal cache, all other algorithms are still below 2% even with 20% of the optimal cache. For users with 10000-25000 requests (See figure 14) we have a small discrepancy with our results since GDSP converges an hit rate of about 6.5% while all other algorithms are only 1,5 % away from the optimum. For users with 25000-50000 requests (See figure 15) the convergence is also very fast at a hit rate of about 9%, while all other algorithms are still at about 2 % with 20 % of the optimal cache.

Byte hit rate. GDSP still outperforms all other algorithms even though the optimal byte hit rate is very limited due to the nature of these specific traces and to the fact that small documents are requested more often than large documents, for which documents that are served in cache are more often small than large (see Figure 16,17,18).

Reduced Latency. Even for reduced latency, the absolute performance of all algorithms is limited by a quite small absolute value, because of considerations similar to those done for the byte hit rate metric and of the fact that popular documents have often a quite limited latency (see Figure 19,20,21).

```
Let A be a caching algorithm;
Let p be the requested page;
Let init(p, A) be the inizialization phase for page p when cached by A;
Let PQ be the Priority Queue of non-cached marked pages,
ordered by time of last reference to the page.  ;
REMOVE(PQ) returns the page referenced last in PQ;


if ((p is in cache) and (p is not expired)) retrieve p from cache;
else retrieve p from server;

// The user abandons page p and it is asked to mark it.

if (p is in cache) and (p is marked by the user) {
  p remains in cache;
  init(p, A);
}

if (p is not in cache) and (p is marked) {
  if (p is cacheble) {
    insert p in cache;
    init(p, A);
    while (the overall size of cached documents exceeds cache memory size) {
          select a page q to evict applying A's eviction policy;
          INSERT(q, PQ);
    }
  }
  else discard p;
}

if (p is not marked) discard p;

if ((PQ is not empty) and (r = REMOVE(PQ) fits the empty space in cache)) {
  prefetch r;
  insert r in cache;
  init(r, A);
}
```

Figure 1: The prefetching/caching algorithm.

Set $L = 0$ at the beginning of the algorithm;

$init(p, GREEDY - DUAL - SIZE)$ {
    Let $H_p = L + c_p/s_p$;
}

// returns the page to be evicted
page $evict - policy(GREEDY - DUAL - SIZE)$ {
    Let $L = min_{q \in cache} H_q$;
    return q such that $H_q = L$;
}

Figure 2: Greedy-Dual-Size

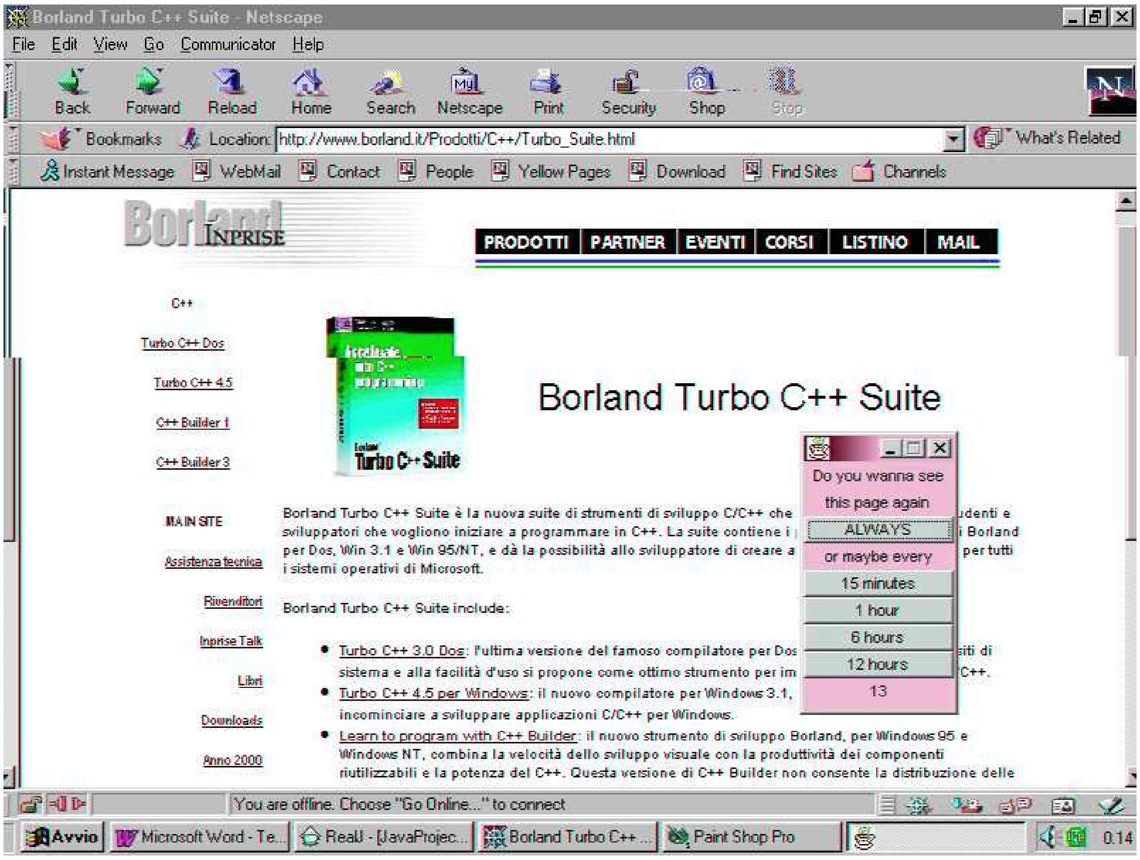

Figure 3: The prototype.

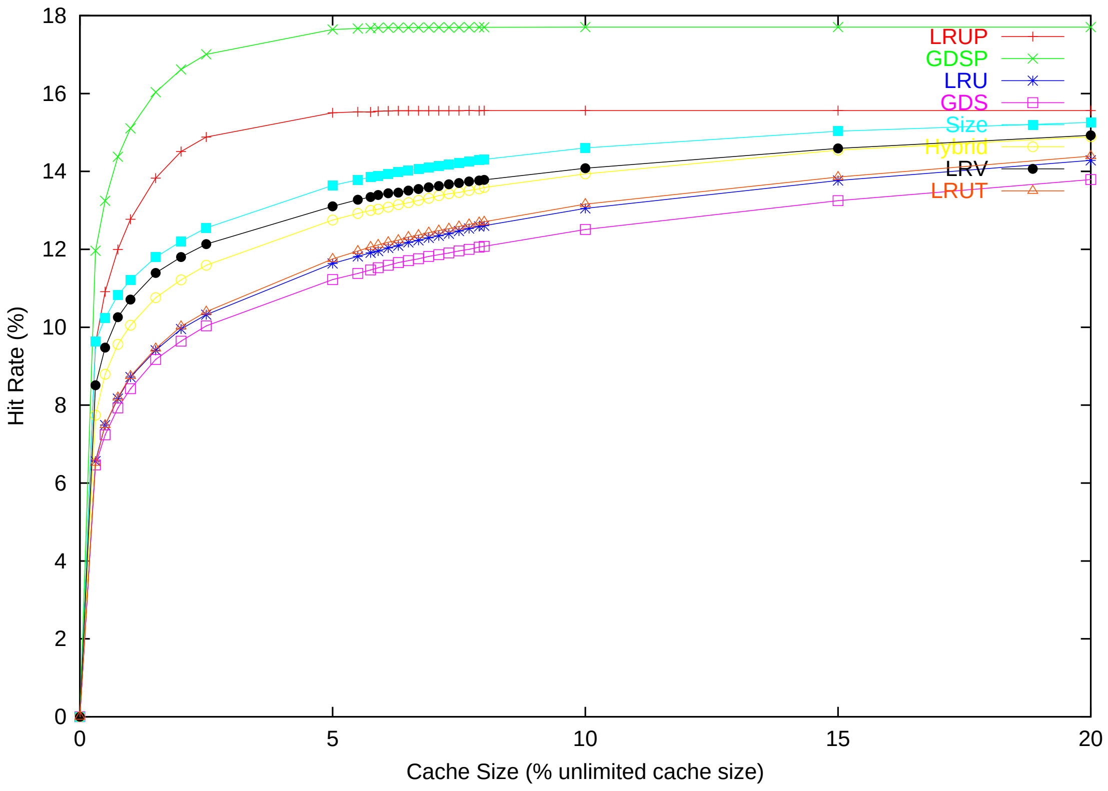

Figure 4: Berkeley (5000-10000 users): Hit Rate.

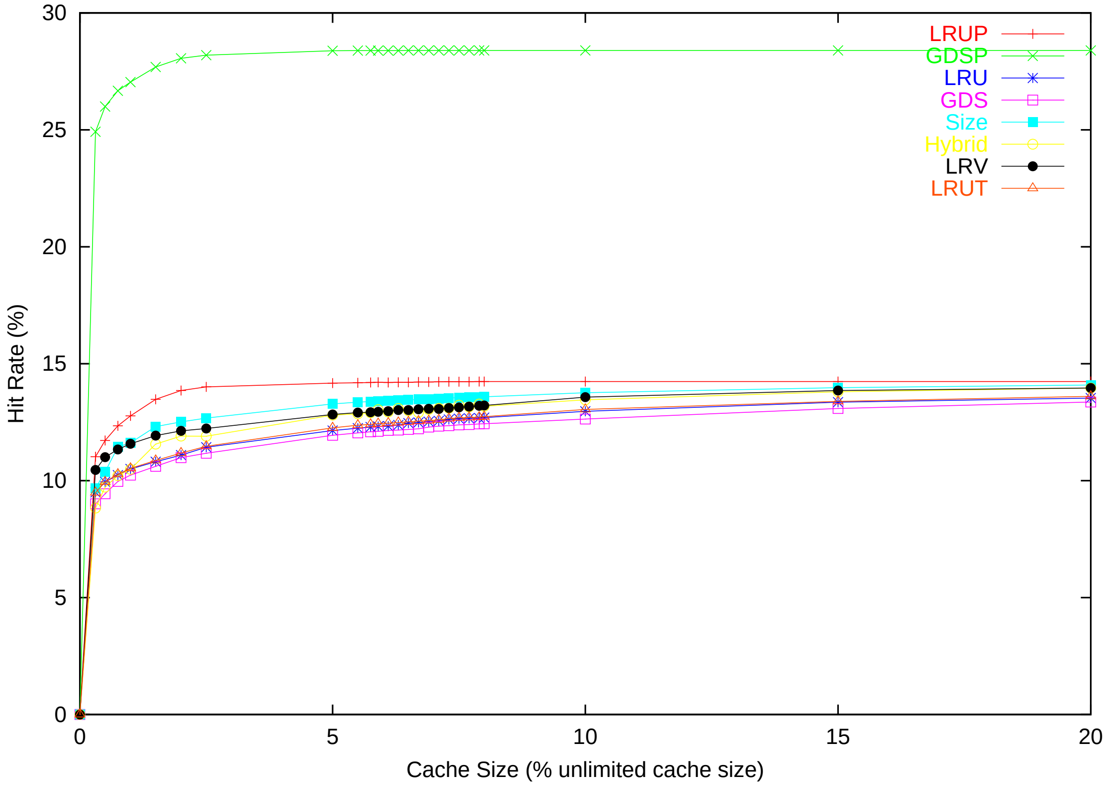

Figure 5: Berkeley (10000-25000 users): Hit Rate.

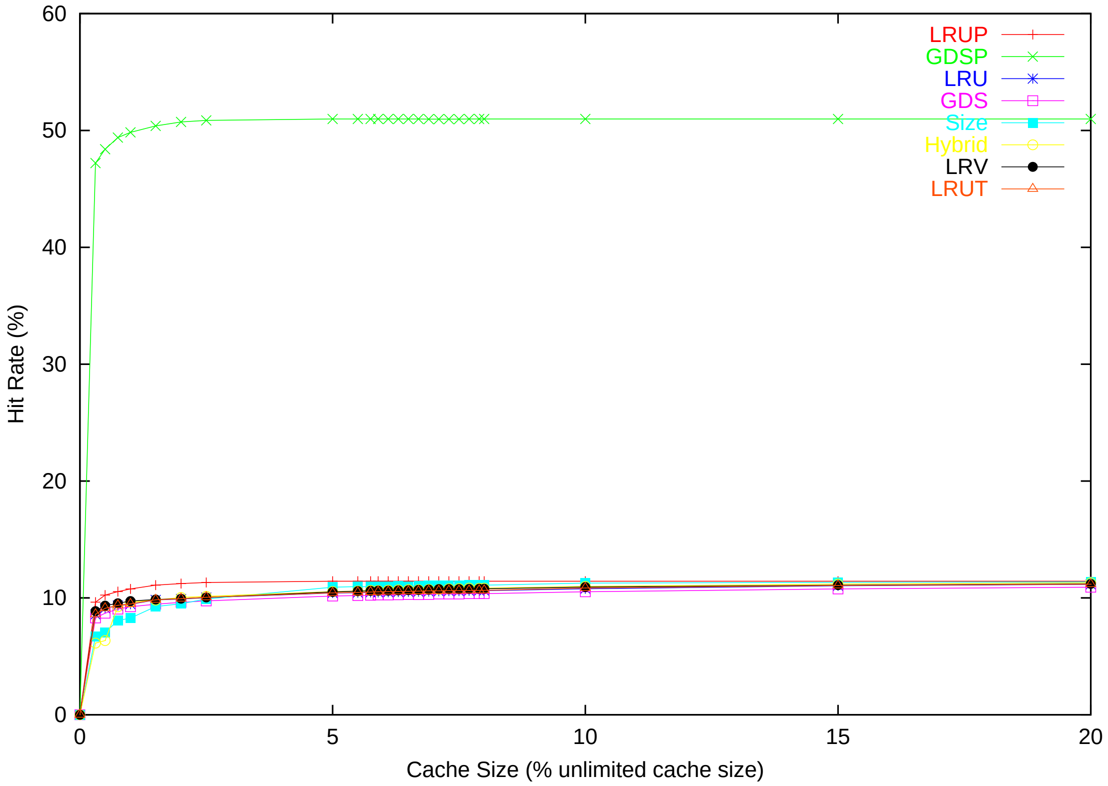

Figure 6: Berkeley (25000-50000 users): Hit Rate.

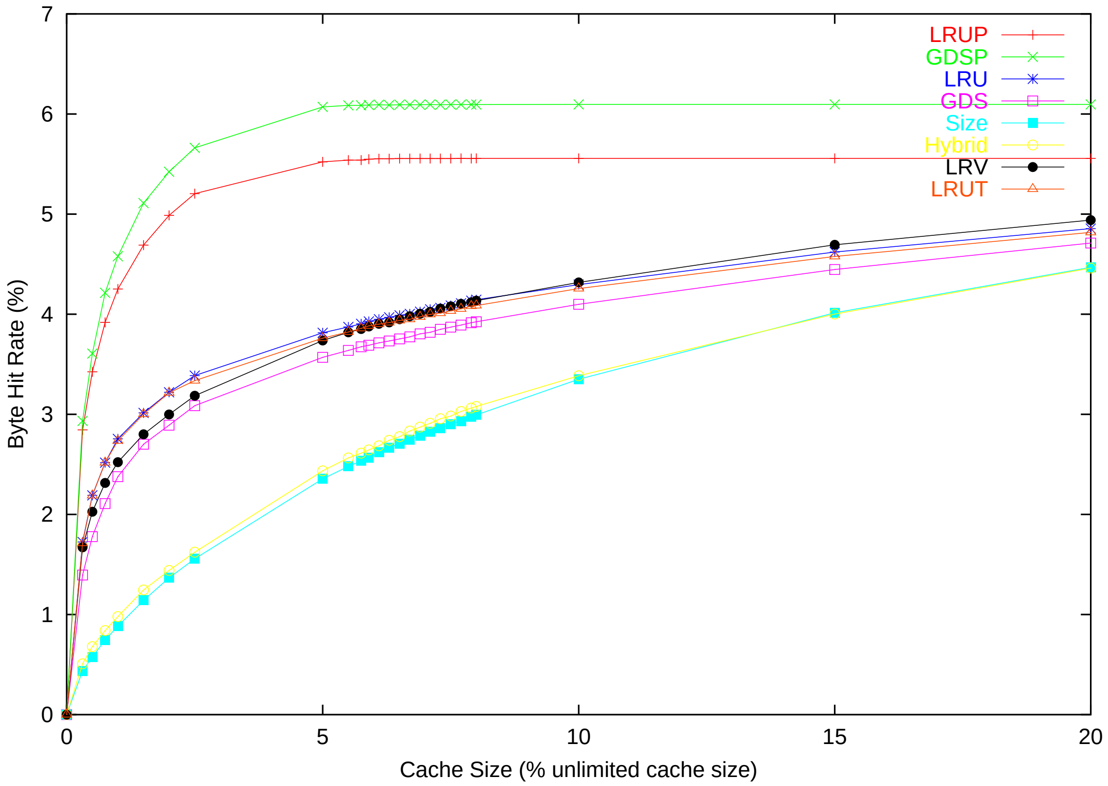

Figure 7: Berkeley (5000-10000 users): Byte Hit Rate.

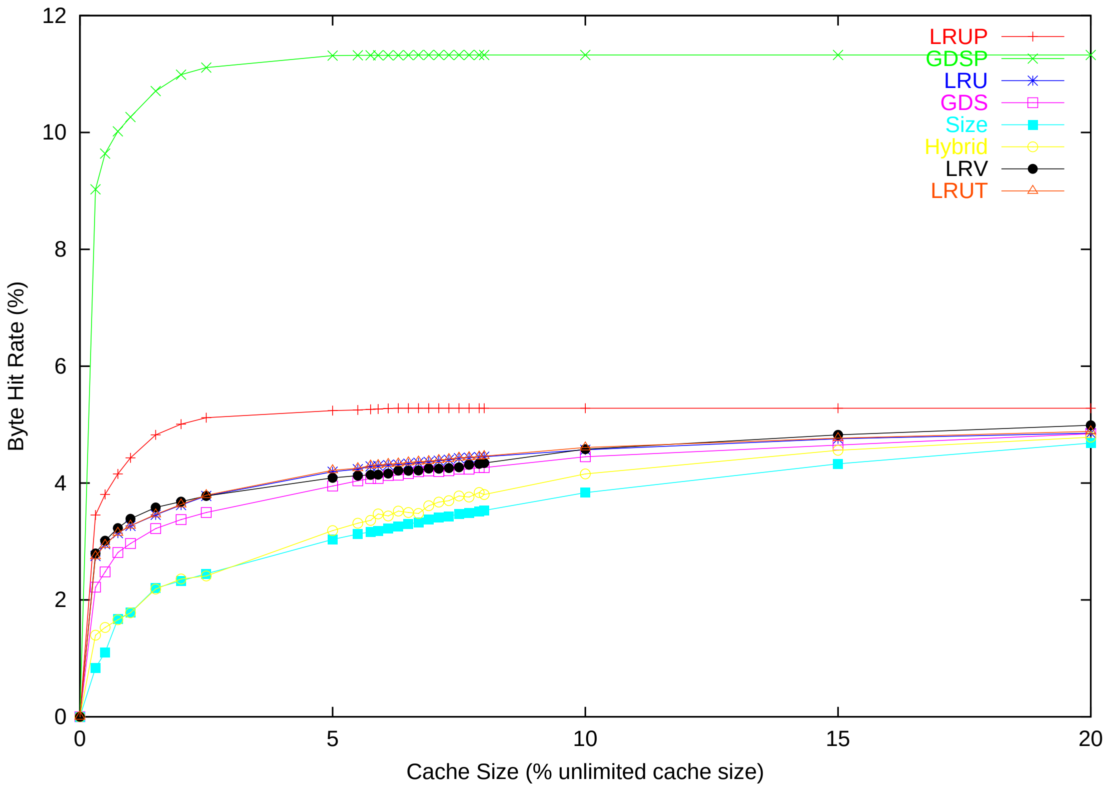

Figure 8: Berkeley (10000-25000 users): Byte Hit Rate.

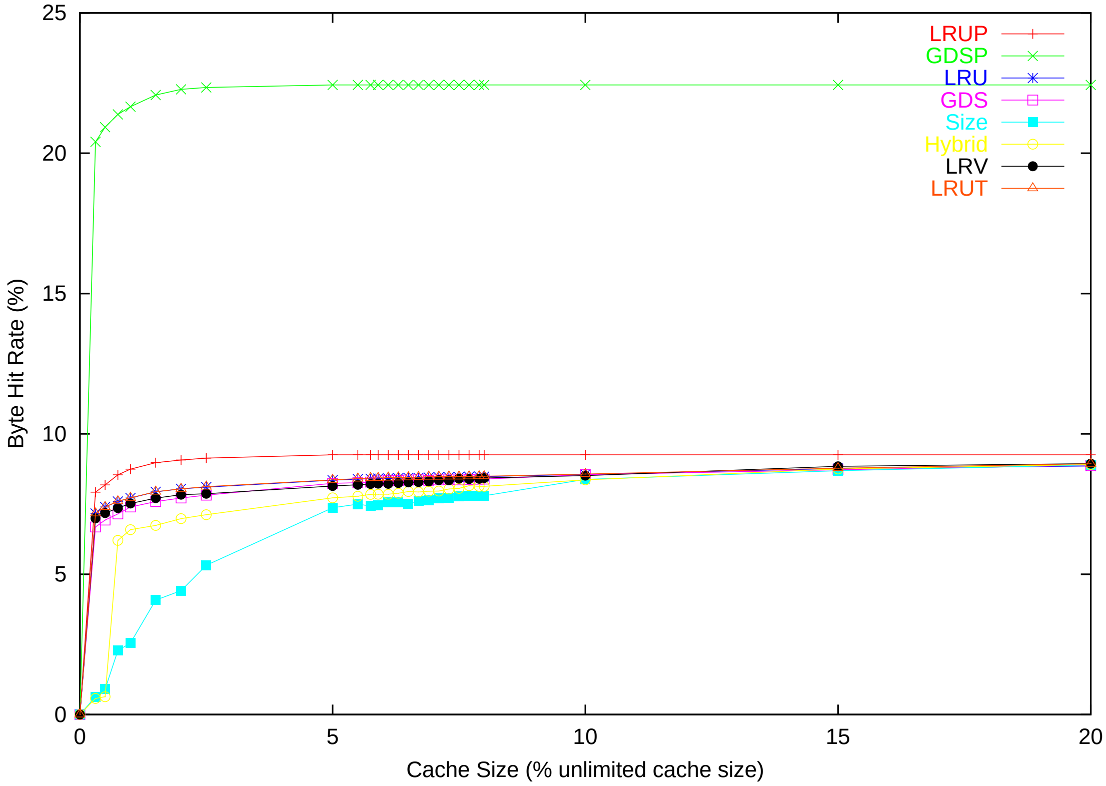

Figure 9: Berkeley (25000-50000 users): Byte Hit Rate.

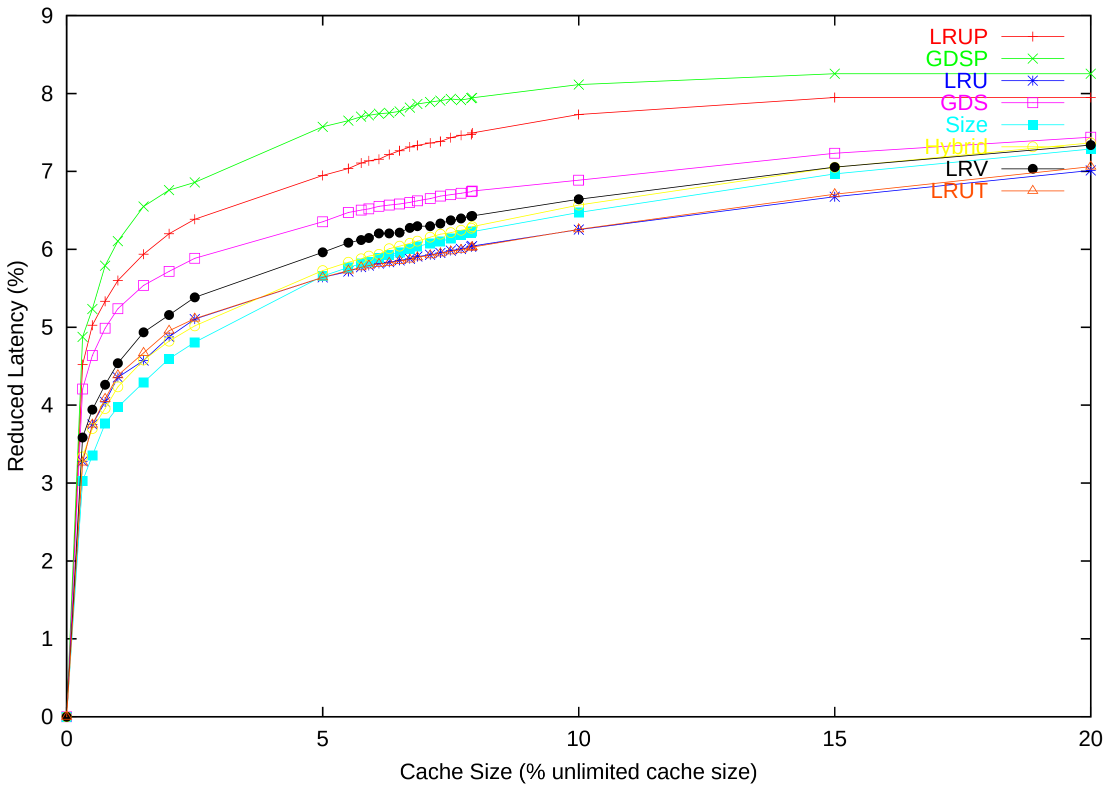

Figure 10: Berkeley (5000-10000 users): Reduced Latency.

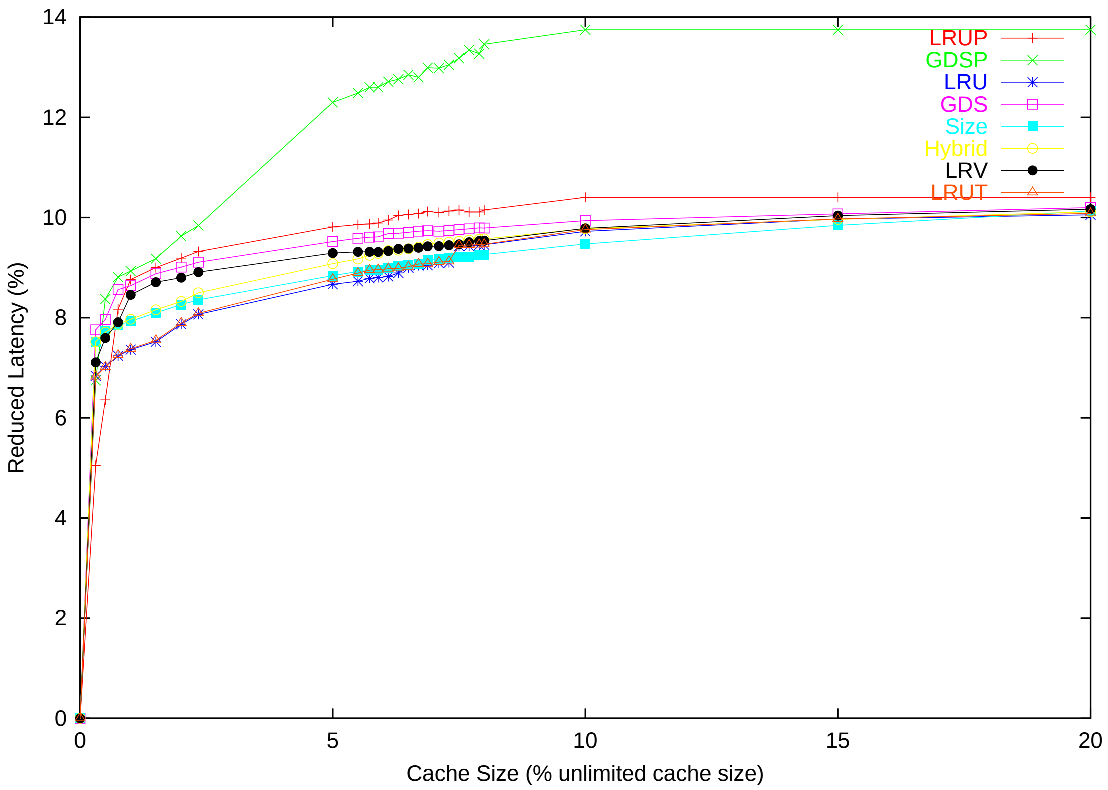

Figure 11: Berkeley (10000-25000 users): Reduced Latency.

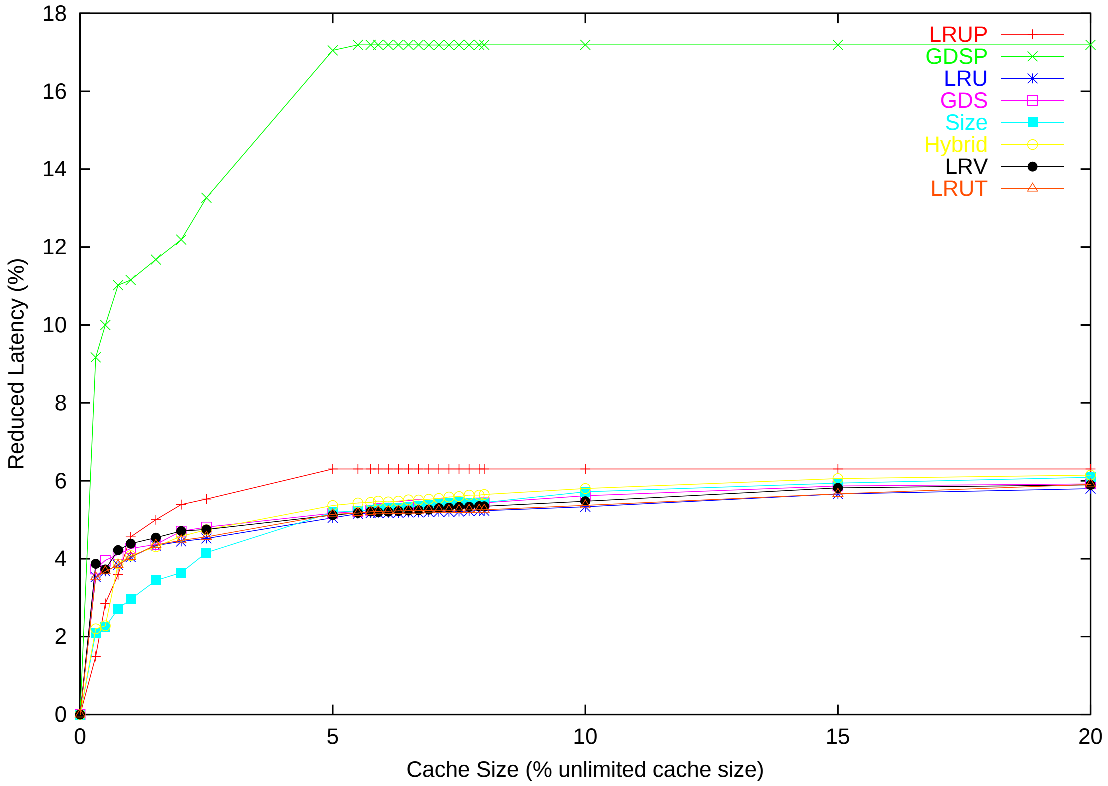

Figure 12: Berkeley (25000-50000 users): Reduced Latency.

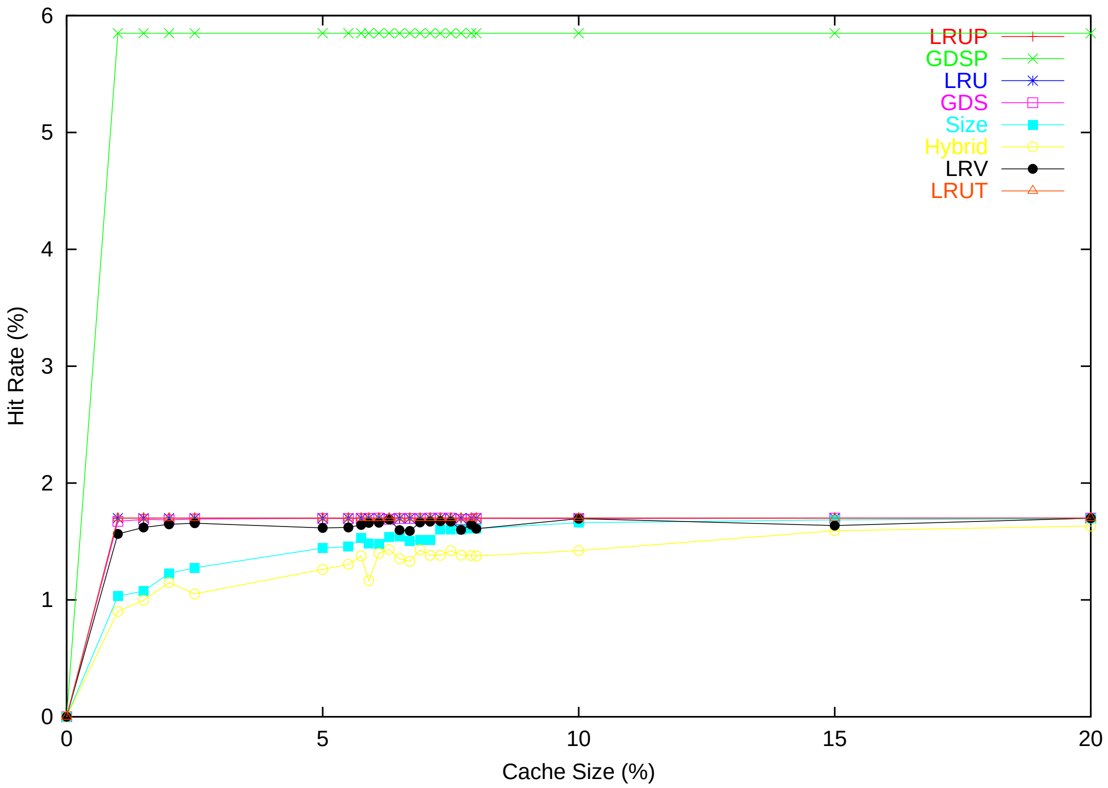

Figure 13: NLANR (5000-10000 users): Hit Rate.

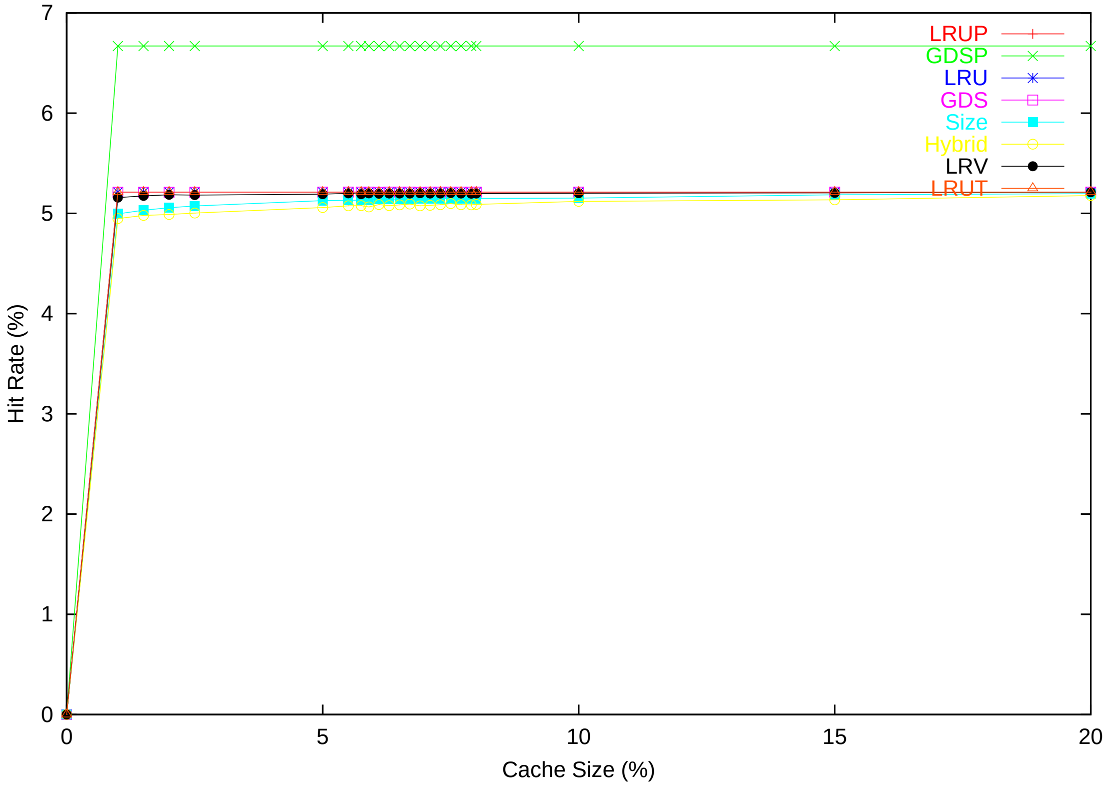

Figure 14: NLANR (10000-25000 users): Hit Rate.

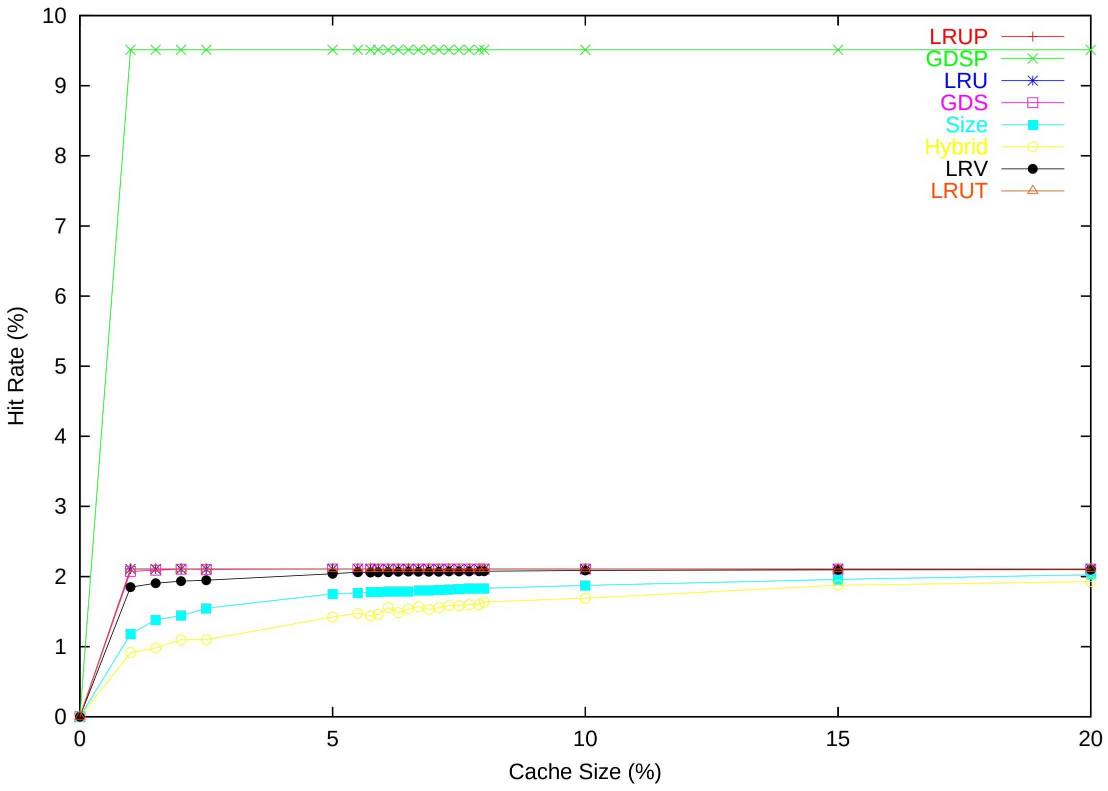

Figure 15: NLANR (25000-50000 users): Hit Rate.

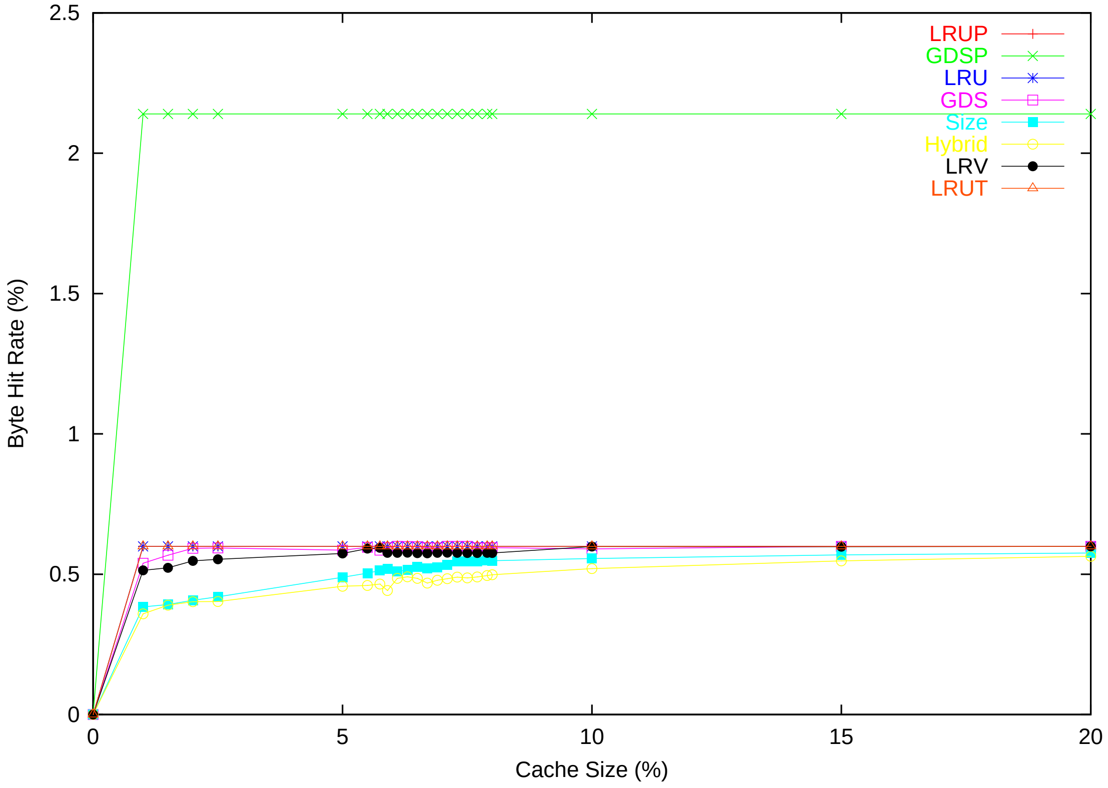

Figure 16: NLANR (5000-10000 users): Byte Hit Rate.

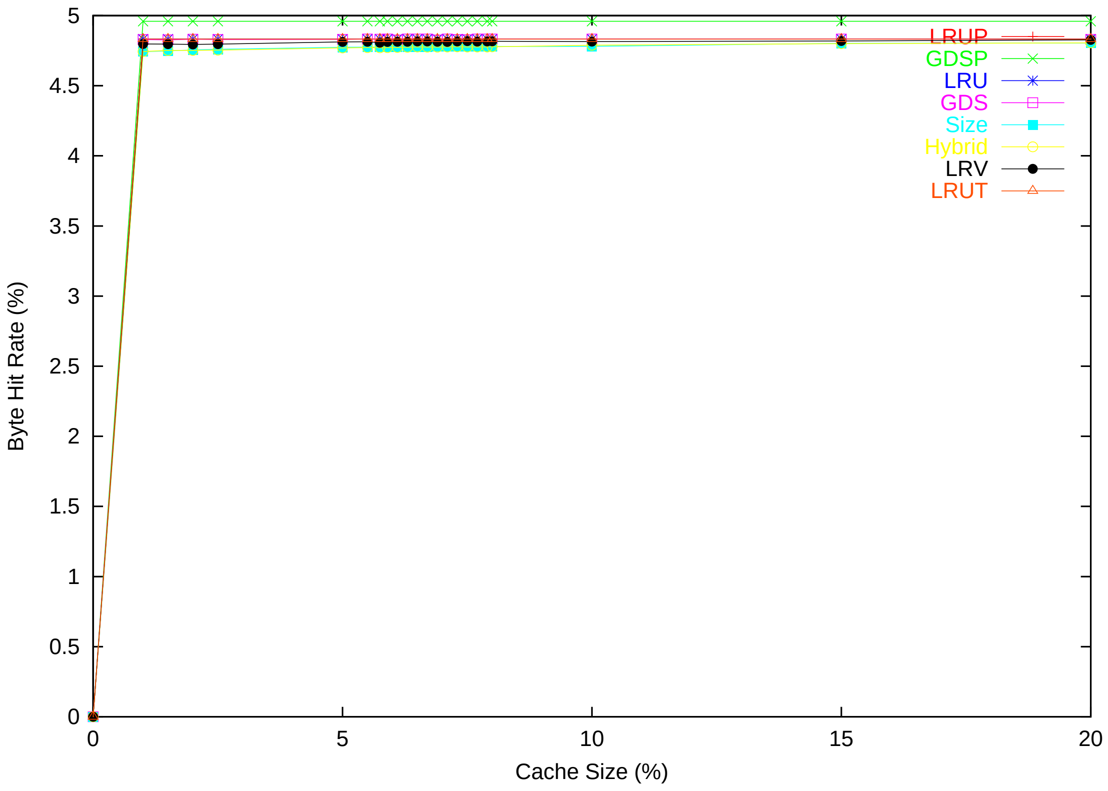

Figure 17: NLANR (10000-25000 users): Byte Hit Rate.

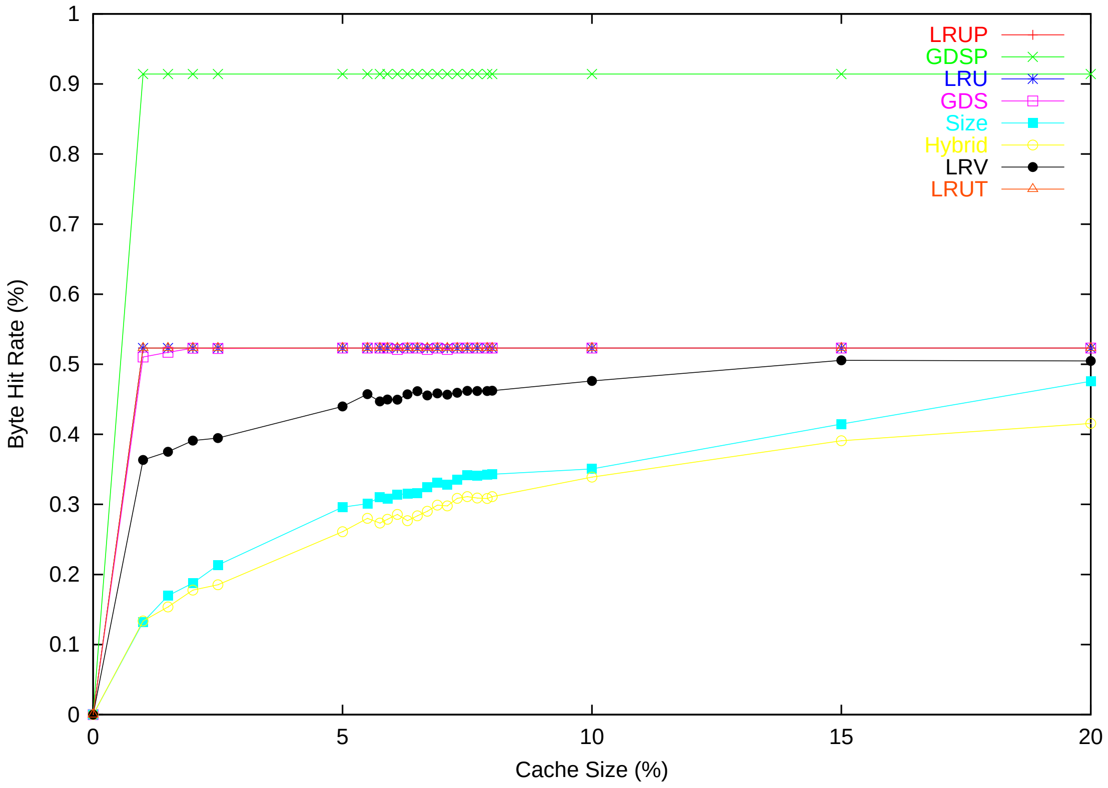

Figure 18: NLANR (25000-50000 users): Byte Hit Rate.

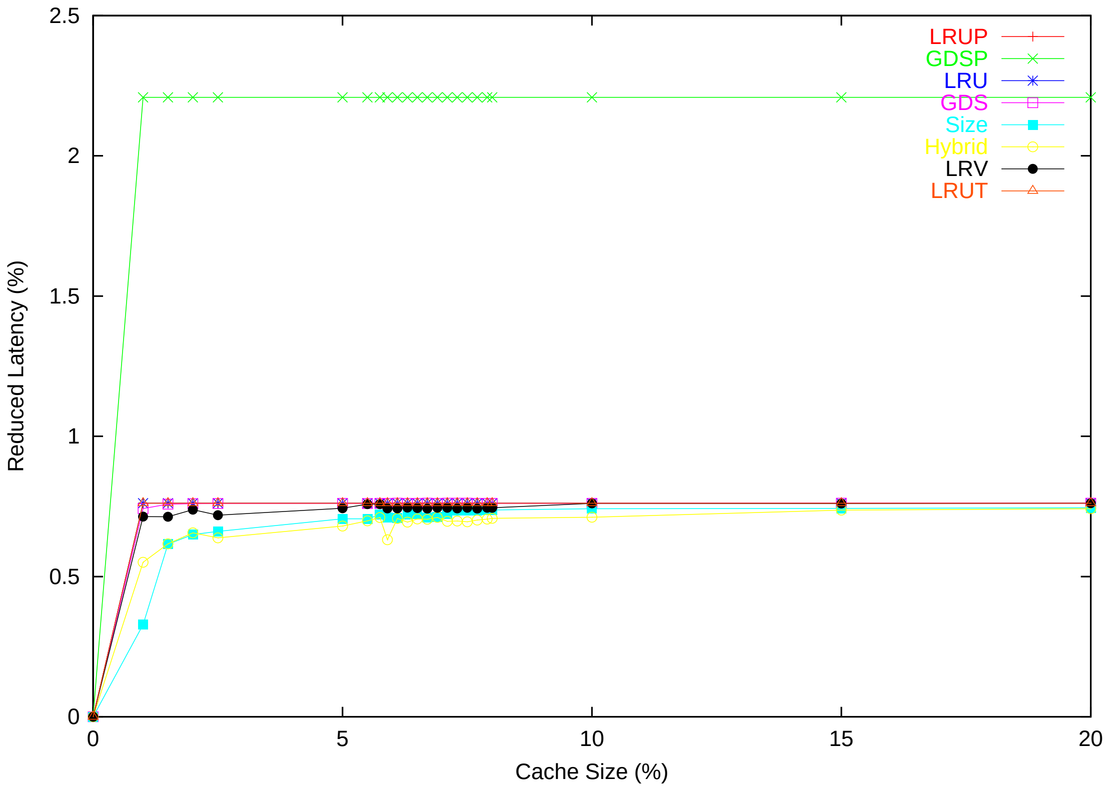

Figure 19: NLANR (5000-10000 users): Reduced Latency.

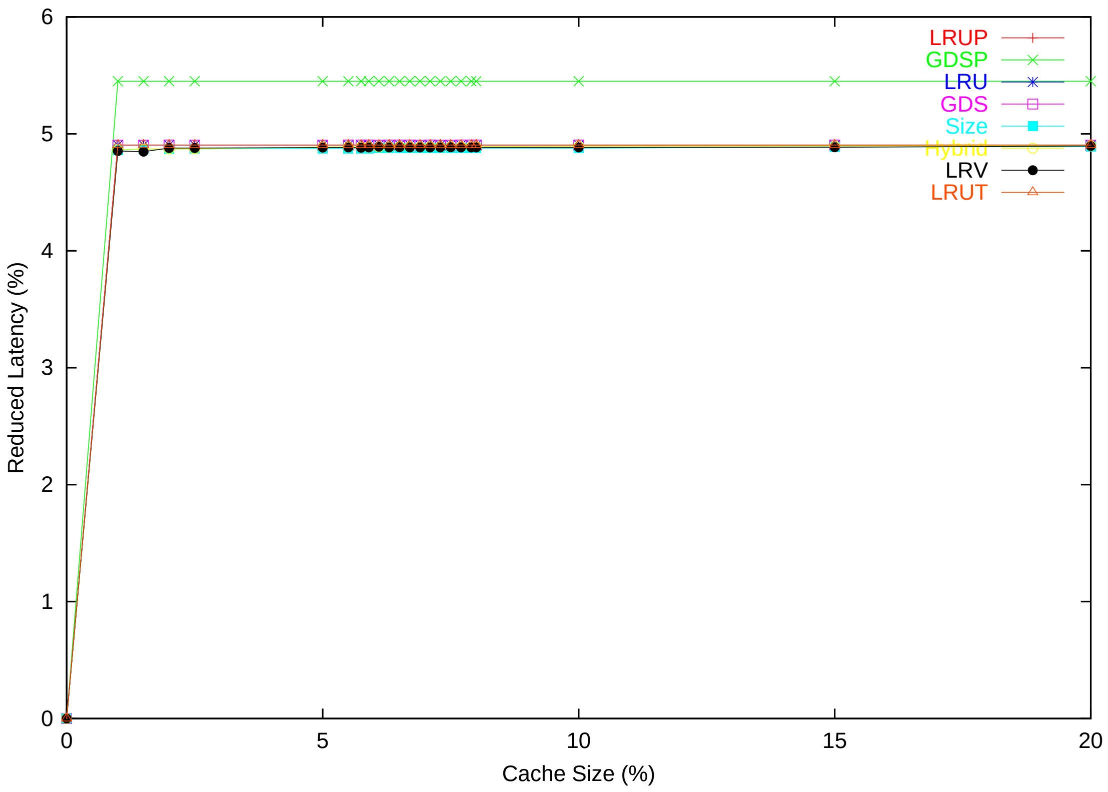

Figure 20: NLANR (10000-25000 users): Reduced Latency.

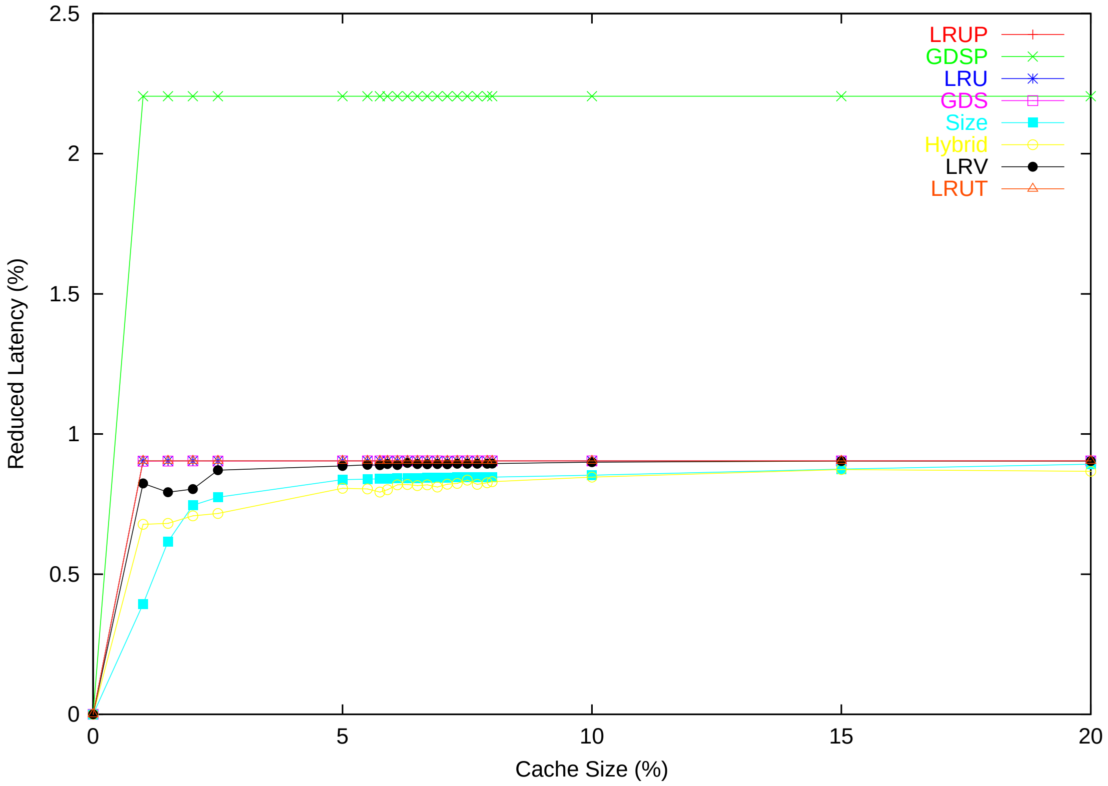

Figure 21: NLANR (25000-50000 users): Reduced Latency.