Static Caching of Web Servers

Zhen Liu[a], Philippe Nain[a], Nicolas Niclausse[a] and Don Towsley[b]

[a]INRIA Centre Sophia Antipolis
2004 Route des Lucioles
B.P. 93, 06902 Sophia Antipolis, France

[b] Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003, U.S.A.

Proc. Multimedia Computing and Networking (MNCN'98) Conference
San Jose, CA, USA, Jan. 24-30, 1998

ABSTRACT

With the increasing popularity of the World Wide Web, the amount of information available and the use of Web servers are growing exponentially. As a consequence, the number of requests to popular Web servers increases exponentially as well. In order to reduce the overhead induced by frequent requests to the same documents, server caching, also referred to as main memory caching, has been proposed and implemented. In this work, we propose a static caching mechanism which consists of updating the contents of the cache periodically and, at the update time, brings into the cache only the most requested documents in the previous time interval. This caching policy has lower management overhead than others. Under some statistical assumptions we show that static caching has the highest hit rate. We also provide empirical comparison results obtained by trace-driven simulations. It turns out that static caching is more efficient in terms of hit rate than those analyzed in the literature.

Keywords: WWW caching, server caching, static caching, main memory caching

1. INTRODUCTION

The number of servers and the amount of information available in the World Wide Web have been growing exponentially in the last five years. The use of World Wide Web as an information retrieving mechanism has also become popular. As a consequence, popular Web servers have been receiving an increasing number of requests. Some servers receive up to 100 million requests daily (Netscape in mid 1996) which results in more than one request per millisecond on average. Thus, in order for a Web server to be able to respond at such a rate, it should reduce the overhead of handling the requests to a minimum.

Currently, the greatest fraction of server latency for document requests (excluding the execution of CGI-scripts) comes from disk accesses. When there is a request for a document at a Web server, the server makes one or more file system calls to open, read and close the requested file. These file system calls result in disk accesses, and when the file is not on the local disk, file transfers through a network.

Hence, it is important to cache the files in main memory so as to reduce access to the local and remote disks. Such an idea has already been used in some software (for example, harvest (now squid*) httpd accelerator[1]). Such a caching mechanism was called main memory caching[2] or document caching.[3] In this paper we shall refer it as *server caching.*

Server caching might appear to have less impact on the quality of Web applications than the *client caching* (or proxy caching), which aims at reducing network delay by caching remote files (see[4,5]). This indeed seems to be true in

Other author information:
E-mail: {Zhen.Liu, Philippe.Nain, Nicolas.Niclausse}@sophia.inria.fr; http://www.inria.fr/mistral/Les_Gens-eng.html
towsley@cs.umass.edu

*http://www.nlanr.net/Squid

traditional networks where the retrieval time of a document is dominated by transfer time due to the low-bandwidth interconnections. However, even in such a situation, a significant fraction of requests of a Web server may be from local users at academic institutions or large companies. These clients are typically connected to the server through high bandwidth LANs (e.g. FDDI or ATM) so that the retrieval time is likely to be dominated by the server's latency. In the near future, with the deployment of ATM WANs, the information retrieval time is also likely to be dominated by the latency at the server.

While client caching is characterized by relatively low hit rates (varying from 20% to 50%[6–9]), server caching can achieve very high hit rates due to the particularity of Web traffic where a small number of documents of a Web server dominates the requirements of clients. An analysis of request traces from several Web servers has shown that even a small amount of main memory (512 Kbytes) can hold a significant portion (60%) of the documents required.[2] The same phenomenon occurs also in the two Web servers studied in the current paper.

Therefore, the analysis and the implementation of server caching can result in a significant improvement of the service of Web servers. In this paper, we propose a new caching mechanism, referred to as *static caching* for server caching. It has lower management overhead and higher hit rate. Under some statistical assumptions we show that static caching has the highest hit rate. We also provide empirical results obtained by trace-driven simulations, which show that static caching compares favorably to other caching policies.

2. CACHING POLICIES

2.1. Existing Caching Policies

The key issue related to the performance of a caching mechanism is the policy which decides how the documents are brought into and removed from the cache. Between these two types of decisions, the removal policy seems to be more critical to the performance of caching. In order to be beneficial, a caching policy should, on one hand, be efficient in terms of hit rate, i.e. the percentage of documents/bytes transferred from the cache, and on the other hand, incur low management overhead. There are already caching mechanisms in operating systems (e.g. Unix) and databases. However, they are based on fixed size pages and not on variable size files, and, therefore, are not appropriate for a Web server which requires entire files. Another problem with the use of caching mechanism of operating systems is that the same main memory cache is shared with other applications, and is read/write oriented, while Web documents are usually read-only.

Most often, removal policies like LRU (Least Recently Used) used in cache software (squid accelerator, or proxy-cache) are those used in traditional operating systems, and do not take into account the specificities of Web traffic. Studies of the performance of LRU and its new variants have been reported for server caches[3,2] and for client caches[6–9] in the last two years for Web applications.

2.2. Static Caching

In this paper, we propose a new caching policy, referred to as static caching. Assume that we have n documents in the Web server, which are requested at rates $r_1 \geq r_2 \geq \cdots \geq r_n$. These rates can be measured and estimated from the log files of the server. Then, for any given size of the cache, we place the most requested files in the cache until the cache is full. When a cached file is modified on the disk, it is also updated in the cache. The request rates are recomputed at a fixed frequency, say, every day, and the contents of the cache are changed accordingly, say, every night.

Our static approach has several advantages over previous cache management policies. First, it is very simple to implement. The cache does not have to be updated at all during the time of high server load, as opposed to dynamic policies where cache update (or garbage collection) algorithms are frequently invoked which results in a heavy overhead. Indeed, as all "missed" documents are brought into the cache, the cache becomes frequently full. Each time it is full, the removal algorithm is activated to remove some of the documents from the cache in order to make room for the new arrivals. Some techniques were used to remedy this problem. One is not to bring into the cache documents exceeding a certain size.[2] Another one is to use higher/lower marks (as in squid): when the cache reaches the higher mark, it activates its removal policy until it reaches the lower mark. Therefore, the removal policy will be activated less often, but the hit rate will be lower as there are fewer documents in the cache on the average.

Another source of cache management overhead is the computational cost of removal policies. They usually maintain a (possibly sorted) list of documents in the cache and make removal decisions based on the counters associated with the documents. Thus, their computational time is often $O(n)$ or $O(n \log n)$, where n is the number of documents in the cache. Sophisticated removal policies tend to require more computational time. In this regard, static caching is the best as the contents of the cache are modified at a low frequency, and the estimation of the request rates can be computed off line.

In the next two sections we shall evaluate our static caching policy through trace-driven simulations. To conclude this section, we present a theoretical justification of the optimality of static caching.

2.3. Optimality of Static Caching

Assume that exactly n documents reside in the Web server which are requested at rates $r_1 \geq r_2 \geq \cdots \geq r_n$. Assume that the sizes of the documents are identically distributed random variables. Moreover, we shall assume that the cache can hold a fixed number m of documents at any time (note that this assumption clearly does not hold for the "smallest size" caching policy). Let X_i denote the index of the document of the i-th request, and $Y_{k,i}$ the indicator function of whether document k is in the cache at the i-th request. We shall assume that $\{X_i\}_i$ is a sequence of independent and identically distributed random variables. Then, the empirical hit rate of caching policy π, denoted by H_π, can be expressed as

$$H_\pi = \lim_{T \to \infty} \frac{1}{T} \sum_{i=1}^{T} \sum_{k=1}^{n} \mathbf{1}(X_i = k) Y_{k,i}.$$

Assume that the sequence $\{X_i, Y_{k,i}\}_i$ is (jointly) stationary and ergodic so that the theorem of Kingman[10] applies. Then,

$$H_\pi = \lim_{T \to \infty} \frac{1}{T} \sum_{i=1}^{T} \sum_{k=1}^{n} E\left[\mathbf{1}(X_i = k) Y_{k,i}\right].$$

For any non-anticipative caching policy π, (i.e., π has no knowledge of future requests), $Y_{k,i}$ depends only on the previous requests $X_1, \ldots, X_{i-1}$ for any i. Let $q_k(\pi) := E_\pi\left[Y_{k,i}\right]$ be the probability that k is kept in the cache. It can be shown[11] that

$$H_\pi = \sum_{k=1}^{n} r_k q_k(\pi) \leq \sum_{i=1}^{n} (r_i - r_{i+1}) \min(i, m) = \sum_{i=1}^{m} r_i$$

where, by convention, $r_{n+1} = 0$.

By definition, the hit rate of the static policy H_s is $H_s = \sum_{i=1}^{m} r_i$. Thus, for any non-anticipative caching policy π, $H_\pi \leq H_s$.

Note that under the assumption that the documents have identical size, the "request hit rate" and the "byte hit rate" (see definitions below) coincide. Note also that the assumption of the stationarity of the request sequence requires that the pattern of client's requests remain statistically the same while shifting the time. It practice, this assumption is fulfilled for time scale of several days.

We assume now that the documents have different sizes, say b_i bytes for document i. As in the previous case, we assume that r_i is decreasingly ordered. Let B be the size of the cache. Then, the optimal static caching can be formulated as a 0-1 programming problem:

$$\begin{array}{ll} \max & \sum_{i=1}^{n} r_i b_i z_i \\ \text{s.t.:} & \sum_{i=1}^{n} b_i z_i \leq B, \\ & \forall i,\, z_i = 0 \text{ } or \text{ } 1 \end{array}$$

where z_i is the indicator function of whether document i is in the cache. The cost to be maximized is proportional to the byte hit rate

$$\frac{\sum_{i=1}^{n} r_i b_i z_i}{\sum_{i=1}^{n} r_i b_i}.$$

Let S denote the cost of the solution of this problem. It is easy to see that such a problem is the well-known knapsack problem, which is NP-hard in general. Thus, in our implementation, we consider an approximate solution resulting from relaxation. Indeed, by relaxing the 0-1 constraint on z_i, i.e., by assuming that z_i is a real number ($z_i \in [0, 1]$), we can easily see that an optimal solution is $\{z_1 = 1, z_2 = 1, \ldots, z_k = 1, z_{k+1} = (B - \sum_{i=1}^{k} b_i)/b_k), z_{k+2} = 0, \ldots, z_n = 0\}$, where $k = \max\{j \leq n | \sum_{i=1}^{j} b_i z_i \leq B\}$.

Let S_r be the hit rate of this optimal static caching obtained by relaxation. It is clear that $S \leq S_r$.

Our approximate solution, denoted by S_a, is obtained by caching documents $1, 2, \ldots, k$ and some further documents after k that can be filled in the cache. Again, it is clear that $S_a \leq S \leq S_r$.

This approximate solution is suboptimal. In practice, however, as the number of WWW documents in a server is usually large, this approximate solution S_a turns out to be very close to the upper bound S_r (the differences being smaller than 0.01% in our numerical experimentation). Therefore, we shall simply take the approximate solution as the optimal static solution. In Section 4, the performance of this solution will be reported.

In the above discussions, we assumed that the cost to be maximized is the byte hit rate. We can do similar analysis for the request hit rate by maximizing $\sum_{i=1}^{n} r_i z_i$ instead of $\sum_{i=1}^{n} r_i b_i z_i$, and with r_i/b_i decreasingly ordered instead of r_i.

As far as the computational overhead is concerned, the static caching policy requires to keep reference counters in order to perform estimations of access frequencies of the documents. When cache is to be reconfigured, a computational cost of $O(n \log n)$ is needed to sort the documents according to their access frequencies. These costs are similar to those of most dynamic policies. However, as we mentioned previously, under the static caching policy, the sorting, as well as the cache replacement, occur much less frequently than under dynamic policies. Thus, the static caching policy is more efficient in this regard.

3. EXPERIMENTS

3.1. Experimental Environment

For this study, we use the access files of two different servers (with two different behavior and sizes): the W3C[†] server, which is fairly frequently visited (almost 400000 requests a day in mid 1995), and uses 2 workstations (www12 and www13) with DNS round robin, and a less frequently visited server, the INRIA[‡] web server (40000 requests/day in Nov. 1996, where almost 20% of the requests come from the local network). Both use the CERN httpd[§] software.

	W3C (www12)	INRIA
Total requests:	303396	41959
Number of requested documents:	5270	6841
Bytes	2.3GB	250MB
Total size of requested documents	72MB	105MB
CGI requests:	<1%	2%

Table 1. *Summary of workload traffic statistics (24h duration)*

For the W3C Web server, the ten most popular documents represent almost 30% of the total traffic in bytes transmitted by the server.

[†]http://www.w3.org
[‡]http://www.inria.fr
[§]http://www.w3.org/pub/WWW/Daemon/

hits	size	URL
8962	6236	/
10082	93	/hypertext/WWW/Icons/WWW/new_red.gif
10119	227	/hypertext/WWW/Icons/WWW/star.km.gif
10146	1407	/hypertext/WWW/Icons/WWW/WWWlogo.gif
14791	137	/hypertext/WWW/Icons/WWW/star11t.gif
15402	83	/hypertext/WWW/Icons/WWW/new.gif
15474	2709	/hypertext/WWW/Icons/WWW/ListOfServers.gif
17300	17453	/hypertext/DataSources/WWW/Servers.html
22661	1437	/hypertext/WWW/Icons/WWW/Virtual_Library.gif
23852	15392	/hypertext/DataSources/bySubject/Overview.html

Table 2. *10 most accessed documents from W3C's Web server (06/07/95)*

3.2. Trace-Driven Simulation

We have developed a simulator which models the behavior of a cache manager for a Web server. We use the approach of trace-driven simulation: the request arrivals of the simulator are taken from a log file. When a new request arrives in the simulator, it checks the contents of the cache to see whether the corresponding documents is already in it (according to its URL). If yes, the cache is left unchanged and the hit counter of the document is increased by one. Otherwise, the reference of the file corresponding to the request is fetched into the cache, provided the cache is not full. In the case where the cache is full, some documents are removed from the cache according to a predefined removal policy, until there is sufficient room for the new file. We also use a threshold: documents with a size greater than 50% of the cache size are never put in the cache.

We analyze two measures of efficiency of the removal policies:

- The request hit rate: percentage of requested documents that are in the cache.
- The byte hit rate: percentage of bytes transferred from the cache.

The byte hit rate seems to be more relevant to the performance of the Web server (and therefore, as stated in section 2.3, the static cache is filled in order to maximise $\sum_{i=1}^{n} r_i z_i$ by decreasingly ordering the r_i).

The following cache removal policies have been investigated:

- LRU (Least Recently Used). This policy removes the files that have not been used for the longest period of time.
- LFU (Least Frequently Used). The documents being referenced less frequently are removed first. In our simulation we use a simple approximation which maintains a reference counter on the number of accesses to each document in the cache. The documents having the smallest number of references are removed.
- SIZE. Biggest files are removed first. This policy tends to be good for the request hit rate (removing large documents leaves a lot of space for smaller documents). However, it can have a higher byte hit rate (although it has been shown to outperform LFU and LRU on proxy caches[8])
- LFU*-aging.[3] It is a more complex removal policy and seems to be quite efficient in terms of request hit rate.
- Static policy: files are never removed from the cache. The documents are placed in the cache according to the previous day's (or hours, weeks ...) statistics.

In order to compare the efficiency in a fair way, we run the simulation with log files for several consecutive days. For each day, we have run the simulator and have looked at the hit rate and weighted hit rate during this day. For the static policy, we have filled up the cache with the most popular documents of the previous day. For the others policies, the cache is initialized with the documents that were in the cache at the end of the previous simulation. We have done this both with the W3C and the INRIA traces.

4. SIMULATION RESULTS

4.1. Comparison with Other Caching Policies

The simulation results on the W3C Web server are illustrated in Figures 1, 2. The variation in time of the efficiency of the caching (removal) policies will be discussed later.

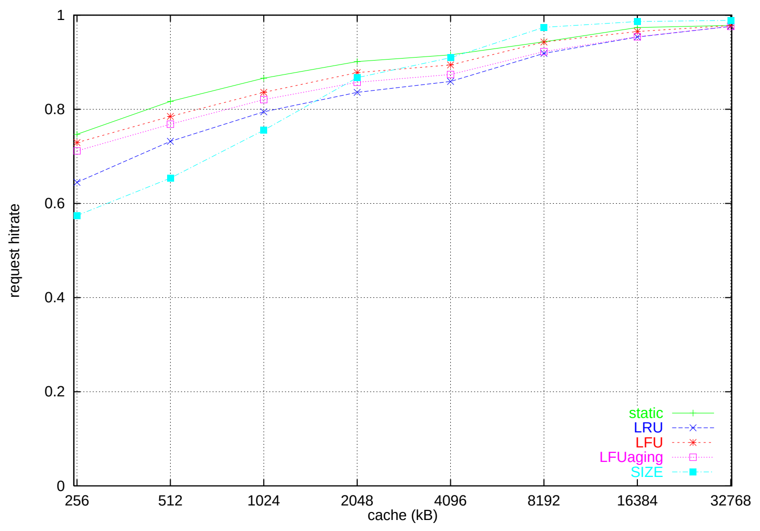

Figure 1. *W3C server - request hit rate*

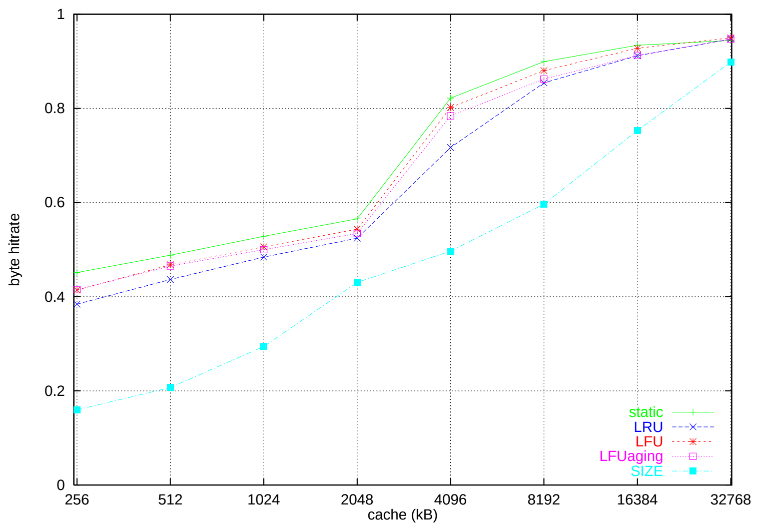

Figure 2. *W3C server - byte hit rate*

As already observed[2] and mentioned previously, very high hit rates (both for requests and bytes) are obtained with a small cache size. With only 256KB, the request hit rate ranges from 55% to 75%, and the byte hit rate is around 40% (except for SIZE, which is only 15%). With a cache of 8MB, the request hit rate is more than 90% with all the policies, and the byte hit rate around 85%, except for the SIZE policy (60%).

It is clear from the comparison, that static caching outperforms all of the others in terms of byte hit rate for any trace and any cache size. It is also better than all of the removal policies (except SIZE) in terms of reducing the request hit rate.

The simulation results confirm our remarks about the behavior of the SIZE policy. It is good for the request hit rate (for cache size > 8MB), but it performs poorly for the byte hit rate (60%, whereas the others have byte hit rate greater than 80%).

Among all the policies other than SIZE, LRU is the worst policy in all cases. LFU is quite similar to static, but slightly less efficient. For example, a 4MB cache with a static policy will read around 100MB/day less from disk than a cache with LFU, and almost 250MB/day less than LRU.

The simulation results on the INRIA Web server are illustrated in Figures 3 and 4.

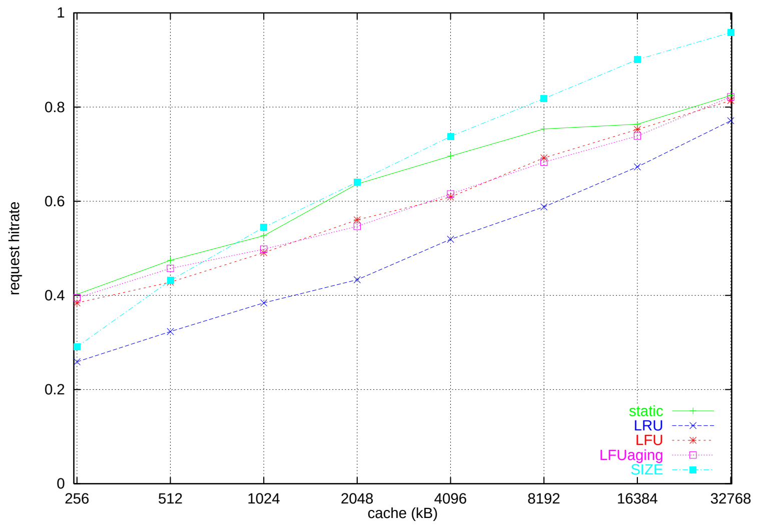

Figure 3. *INRIA server - request hit rate*

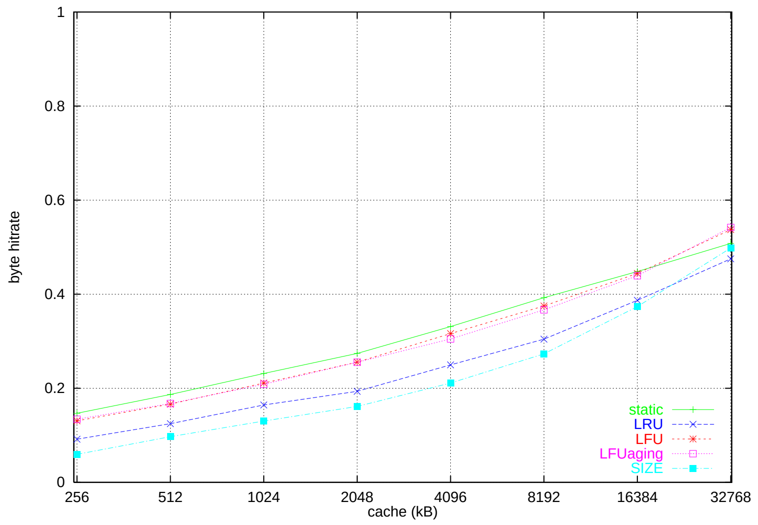

Figure 4. *INRIA server - byte hit rate*

One can see that observations made previously about the efficiency of the removal policies on W3C server still holds for the INRIA Web server. However, the miss rates are now higher (for the INRIA server) than in the previous case (the W3C server). With a cache of 8MB, the byte hit rate is around 50%, and the request hit rate around 70%. This phenomenon can be explained by the fact that the most required documents are more concentrated in the W3C Web server than in the INRIA Web server.

4.2. Empirical Sensitivity Analysis of Static Caching Policy

We have carried out simulations for ten consecutive days. The curves provided in Figures 1 and 2 as well as in Figures 3 and 4 are a snapshot of one day. The variation in time of the efficiency of the caching (removal) policies is illustrated in Figure 5. One can see that the qualitative comparison among the policies holds for all the days, except for day 7, which corresponds to a Monday, on the INRIA Web server. The "bad" performance of static caching can be explained by the fact that the estimates of the request rates used for the cache on Monday were computed based on the few requests of the previous Sunday where the server was very lightly loaded.

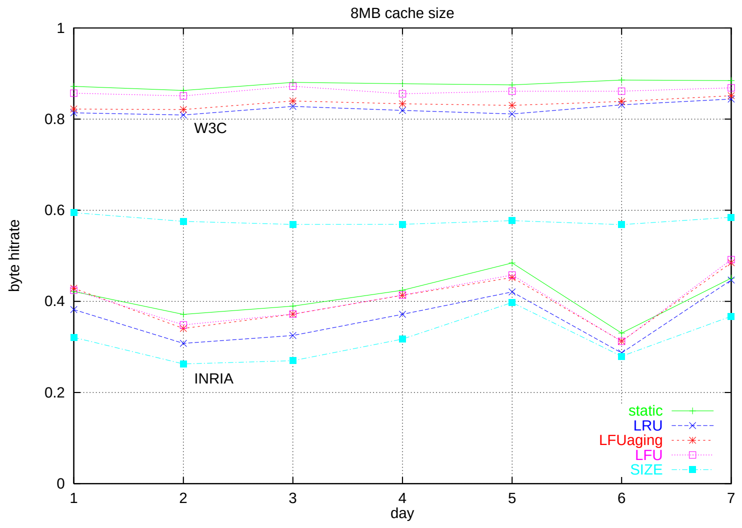

Figure 5. *Variation in time of the performance of the caching policies for W3C and INRIA servers - byte hit rate*

Since the request rates used in the implementation of the static policy are estimated from the statistics of previous requests, one expects that the performance of the static policy will vary over time, as we have already seen in Figure 5. We thus conducted additional simulations to see how it varies over time for different sizes of the cache. The results are reported in Figures 6 and 7, which seem to indicate that the forms of the curves are insensitive to the sizes of the cache.

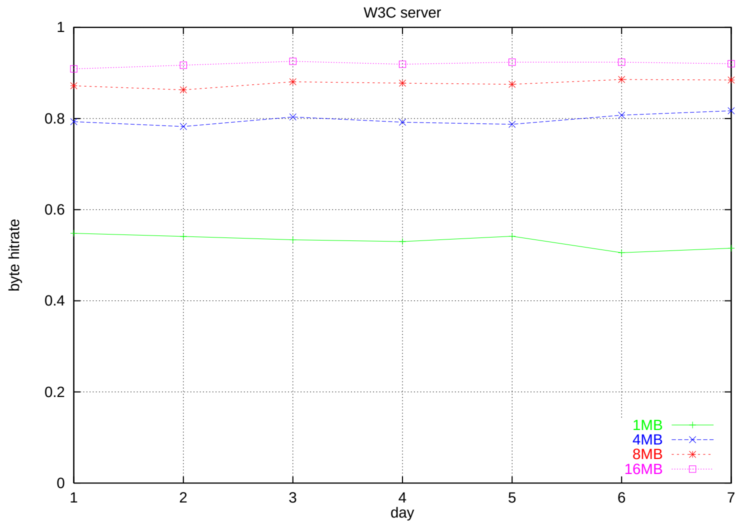

Figure 6. *Evolution of byte hit rate for static policy over a week for W3C server.*

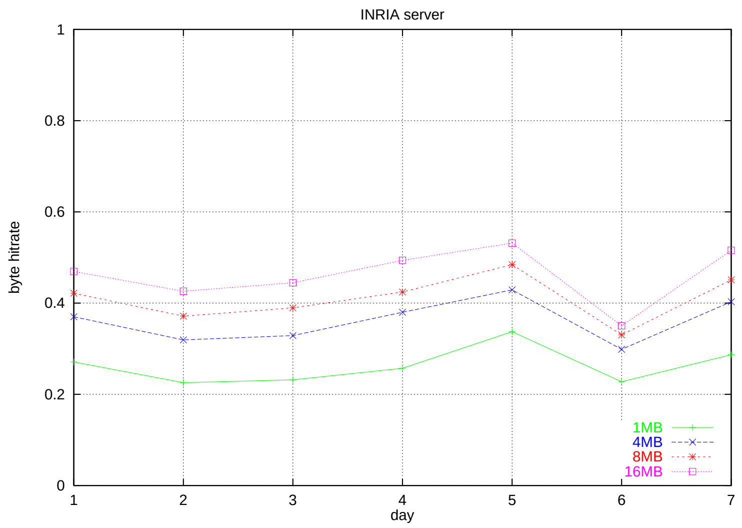

Figure 7. *Evolution of byte hit rate for static policy over a week for INRIA server.*

One can see, without surprise, that the static policy is very stable for the W3C server. For the INRIA server, however, it varies slightly. One reason for this is the small amount of data (i.e. number of request) used in the estimation. Indeed, in this case, the total number of requested documents is larger whereas the total number of requests is much smaller.

The observation that hit rate varies from day to day motivates our investigation to the behavior of the static policy on the INRIA server when we increase the number of days taken into account for the estimation of the request rates of the documents. The results are illustrated in Figure 8. It turns out that the hit rate of static policy is insensitive

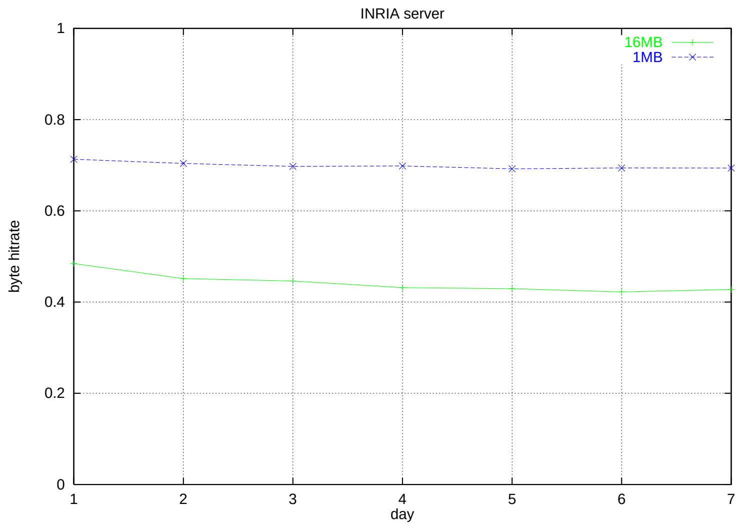

Figure 8. *Efficiency of the static policy on the INRIA server when the number of day to compute the statistics increases.*

to the number of days taken into account in the estimation. This implies that one-day workload is sufficient for the estimation of request rates in this case. However, for the Monday where we observed “bad” performance of static caching in Figure 5, when the estimates are computed for the last two or three days, there is significant improvement of the efficiency.

Last, we investigated the impact of the stationarity assumption we made on the data accesses while presenting theoretical justification of the static caching policy. Indeed, the hypothesis of stationarity is not always true, in which case the approximate solution, which is based on past access frequency, may significantly loose its efficiency in comparison with the ideal optimal solution. To examine this effect, we compare the approximate solution with the upper bound S_r which is implemented such that it uses the access frequency computed *a posteriori*.

With the W3C server traces, the approximate policy is still very close to the upper bound, so to the optimal solution, see Figure 9. For the INRIA server's traces, it is no longer the case, especially with large caches, as illustrated in Figure 10.

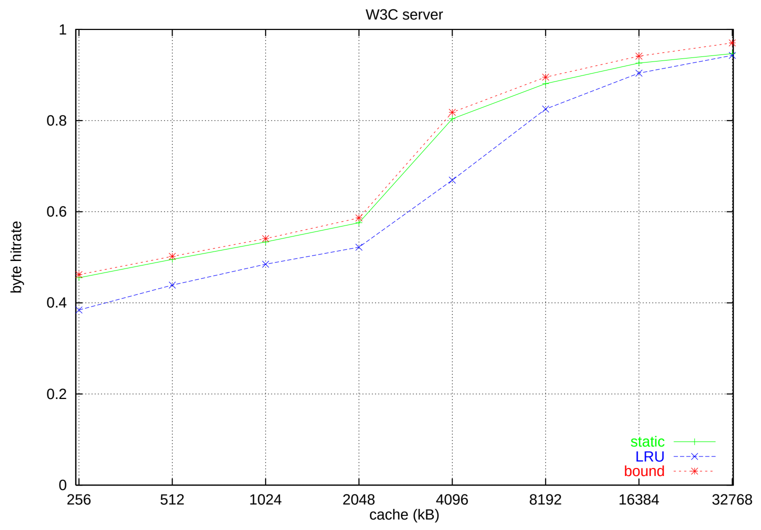

Figure 9. *W3C WWW server byte hit rate*

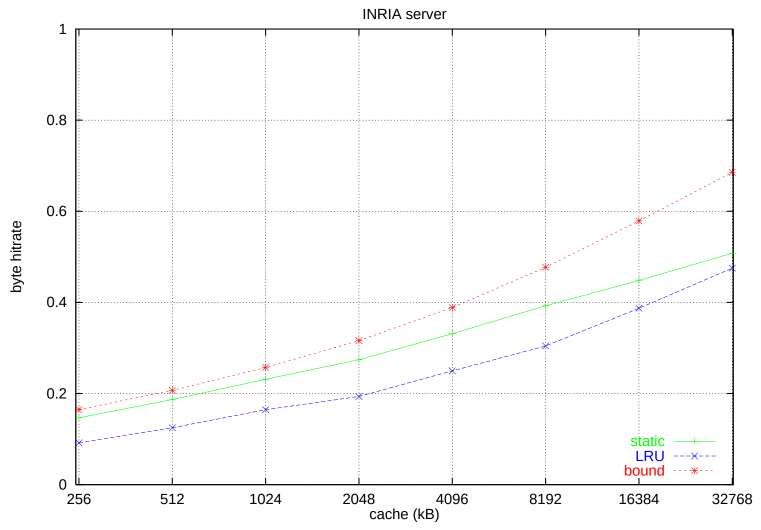

Figure 10. *INRIA's WWW server byte hit rate*

5. CONCLUSION

We have proposed and evaluated a static server caching policy. It incurs lower management overhead and exhibits a higher hit rate. Under some statistical assumptions we have shown that static caching gives the highest hit rate. We have also provided empirical comparison results obtained by trace-driven simulations. The simulation results also illustrated that the static caching is more efficient than those analyzed in the literature.

One intuitive explanation of the good performance of the static caching policy is that it avoids bringing documents into the cache that are accessed only once. It turns out that a nonnegligible proportion of documents are accessed only once. Thus, dynamic policies waste both space and time in bringing them into the cache.

One potential disadvantage of static caching is that, because it infrequently modifies the contents of the cache, it exhibits a long delay in reaction to important events where some document becomes all of a sudden very popular. In this case, the cache has to wait for the next update to bring in this document. Such a situation might occur at the creation of new pages. Thus, a possible remedy could be to put potentially highly required new pages in the cache as soon as possible. This results in a mixed caching policy where part of the cache is managed in a static manner while the other part is managed in a dynamic manner. Such an approach is currently under investigation.

Another direction of on-going research is the design of semi-static caching policies where the cache reconfiguration is invoked when the miss rate exceeds a threshold, or when the number of total requests exceeds a threshold.

REFERENCES

1. A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," Tech. Rep. 95-611, Computer Science Department, University of Southern California, Los Angeles, California, Mar. 1995.
2. E. Markatos, "Main memory caching of web documents," in *Proceedings of the 5th World Wide Web Conference '1996*, (Paris, France), May 1996.
3. M. Arlitt and C. Williamson, "Trace-driven simulation of document caching strategies for internet web servers," tech. rep., Dept. of Computer Science, University of Saskatchewan, Canada, 1996.
4. H.-W. Braun and K. Claffy, "Web traffic characterization: An assessment of the impact of caching documentts from ncsa's web server," *Computer Networks and ISDN Systems* **28**, pp. 37–51, 1995.
5. M. E. Crovella, A. Bestavros, and C. Cuhna, "Characteristics of www client-based traces," tr-95-010, Computer Science Dept., Boston University, 1995.
6. P. Cao and S. Irani, "Cost-aware www proxy caching algorithms," in *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pp. 193–206, Dec. 1997.
7. S. Williams, M. Abrams, C. Standridge, G. Abdulla, and E. Fox, "Caching proxies: Limitations and potential," in *Proceedings of the Fourth World Wide Web Conference '1995*, (Boston, MA), 1995.
8. 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.
9. 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.
10. J. F. C. Kingman, "The ergodic theory of subadditive stochastic processes," *J. Roy. Statist. Soc. Ser. B* , pp. 499–510, 1968.
11. Z. Liu, P. Nain, N. Niclausse, and D. Towsley, "Static caching of web servers." Available at http://www.inria.fr-/mistral/personnel/Nicolas.Niclausse/articles/mmcn98/.