Computer Networks and ISDN Systems 28 (1996) 893–905

COMPUTER NETWORKS and ISDN SYSTEMS

Main memory caching of Web documents

Evangelos P. Markatos [1]

Computer Architecture and VLSI Systems Group [2], *Institute of Computer Science (ICS)* [3], *Foundation for Research and Technology – Hellas (FORTH), P.O. Box 1385, Heraklio, Crete, GR-711-10 Greece*

Abstract

An increasing amount of information is currently becoming available through World Wide Web servers. Document requests to popular Web servers arrive every few tens of milliseconds at peak rate. To reduce the overhead imposed by frequent document requests, we propose the notion of caching a World Wide Web server's documents in its main memory (which we call Main Memory Web Caching). We show that even a small amount of main memory (512 Kbytes) that is used as a document cache, is enough to hold more than 60% of the documents requested.

We also show that traditional file system cache management methods are inappropriate for managing Main Memory Web caches, and may result in poor performance. Based on trace-driven simulations of several server traces we quantify our claims, and propose a new cache management that dynamically adjusts itself to the clients' request pattern and cache size. We show that our policy is robust over a variety of parameters and results is better overall performance.

Keywords: WWW caching; File caching; Prefetching

1. Introduction

Results suggest that World Wide Web (hereafter called Web) servers receive an everincreasing amount of requests for documents. Some servers receive up to 25 requests per second at peak times [6]. As more users start using the Web, the number of requests each server is going to see is bound to increase. To be able to respond at these request rates, a Web server should avoid all unnecessary overheads. One of the sources of Web server overhead is related to disk accesses. When a client requests a document from a Web server, the server performs one or more file system calls to open, read, and close the requested file. Depending on the file system cache policy of the server, these system calls may result in one or more disk accesses. To make matters worse, if the document does not reside in a disk local to the server, it has to be fetched through a network file system (e.g. NFS) which makes the server overhead of getting the document even higher.

File system caches may reduce the number of disk accesses by caching the most popular file blocks in the server's main memory. Unfortunately, as we show in this paper, traditional methods for caching like those used in file systems do not perform well for Web traffic for the following reasons:

- *Caching granularity is different*: Traditional file system caching policies tend to cache individual blocks, while Web caches should cache whole files, because either the whole file is requested or

[1] Email: markatos@csi.forth.gr, http://www.ics.forth.gr/~markatos

[2] http://www.ics.forth.gr/proj/arch-vlsi

[3] http://www.ics.forth.gr/

0169-7552/96/$15.00 © 1996 Published by Elsevier Science B.V. All rights reserved
PII S0169-7552(96)00035-9

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270

nothing at all. Moreover, Web Caches that decide to cache a file, may request the whole file to be fetched with one disk access. File systems, on the other hand, need to fetch the file one (or a few) blocks at-a-time.

- *Operating System Intervention*: Even when the requested document is in the file cache, the Web server still has to pay the overhead of a few file system calls to open and read the file.
- *Web documents are typically accessed in read-only mode*: Most Web requests read a document without changing its contents. On the other hand, file systems need to deal both with read-only and read-write traffic, which may force them to choose conservative implementations. For example, file systems may copy data blocks several times as they cross protection domain boundaries; this overhead can be clearly avoided for read-only data.

In this paper we propose and explore *Main Memory Caching* of frequently requested Web documents *on the Web server's memory*. That is, if a document is frequently accessed, then it is kept in the Main Memory Cache, inside the Web server's address space. The server accesses its own cache directly, without any help from the file system. When a client requests a document that resides in the server's main memory cache, the document is directly read from the server's cache and sent to the client without making any file system calls to access the document. We believe that main memory caching of Web documents is beneficial today, and will be even more attractive in the near future for the following reasons:

- *The time to receive a document is likely to be dominated by the server's latency*: Although the time to receive a document in traditional networks has been dominated by the transfer time due to low-bandwidth interconnections, the situation has started to change: A significant percentage of a server's requests originate from clients that reside in the same LAN with the server. These clients can be connected to the server via a fast network like an FDDI, or even an ATM. In these cases, the time to transfer a document over the high-bandwidth interconnection medium is low, and the latency that the clients experience is mainly due to the server's latency.
- This argument holds (although to a lesser extent) for remote clients as well. Servers with Internet connections of 20 Mbps are not uncommon [6], while experimental 155 Mbps ATM WANs that connect research, academic, and supercomputing institutions are starting to appear. Also, in these environments the total time to receive a document is likely to be dominated by the server's overhead and not the document's transfer time.

Our work complements traditional Web Caching research. Web Caching systems attempt to reduce a server's load by caching documents close to the clients that request them. Our Main Memory Web Caching system reduces a server's load by keeping the most frequently accessed documents in the server's main memory, thus avoiding disk accesses as much as possible.

2. Experiments

2.1. The experimental environment

2.1.1. The Traces

To evaluate the performance benefits of main memory Web caching we use trace-driven simulations. We have gathered server traces from several Web servers from a variety of environments that include Universities, Research Institutions, and Supercomputing Centers both from Europe and the States. All traces total more than one million requests. Specifically the traces are from:

- *Crete*: Traces from the Computer Science Department of the University of Crete, Crete, Greece. These traces represent infrequent traffic, mostly localized within the department.
- *Parallab*: A Supercomputing Center associated with the University of Bergen, Norway. These traces represent heavy traffic, especially from all over Norway.
- *NCSA*: The traces of January 8th 1995 of the NCSA server, USA These are the traces of one of the busiest servers in the world.
- *FORTH*: Traces from the Institute of Computer Science of FORTH, heavily accessed from within Greece. These traces include a lot of traffic from all over Greece.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 2 of 13

- *Rochester*: Traces from the Computer Science Department of the University of Rochester, NY, USA.

We believe that it is very important to use traces from a variety of sources. Since different traces have different characteristics and because the server they are gathered from, provides different documents to its clients. For example, some sophisticated servers that contain lots of images, movies and sounds contain mostly large files. On the other hand, servers connected with slower lines to the outside world provide mostly (highly interconnected) short text files. Our experimental results suggest that the average size of the requested documents plays a significant role in the performance of caching policies and in the choice of the best caching policy.

2.1.2. The caching policies

These traces were fed into a trace-driven simulator that simulates a main memory cache of a given size. The server caches documents according to the following policies:

- ALL: Cache all documents. The cache replacement policy is LRU (Last Recently Used). The unit of caching and replacement is a document.
- THRESHOLD: Cache all documents up to a certain (threshold) size only. The threshold is a fixed run-time parameter specified in the simulations. The cache replacement policy is LRU.
- ADAPTIVE: Similar to the THRESHOLD policy, with the exception that the threshold is not fixed, but it is calculated dynamically by the algorithm given in Section 2.4. The cache replacement policy is a variation of the LRU: When the cache runs out of space, the documents that are larger than the (dynamically adjustable) current threshold are evicted on an LRU basis. When they are all evicted from the cache, the rest of the documents are evicted on an LRU basis until there is enough space in the cache.

2.1.3. The performance metrics

The ultimate goal of each Web server is to be able to serve client requests at the maximum rate. This rate becomes higher when most of the requested documents are found in the Main Memory Cache. Thus, the Cache Hit Rate is a fundamental factor in the server's performance.

There are two variations of the cache hit rate:

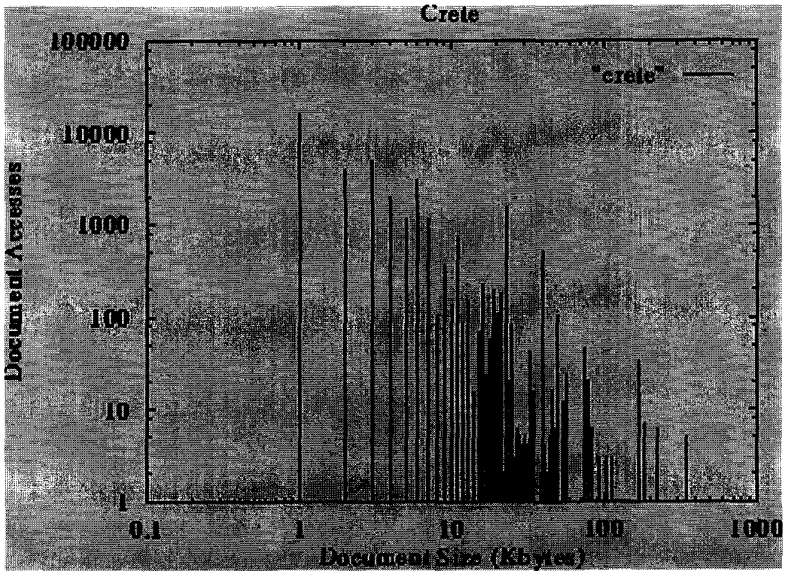


Fig. 1. Number of accesses to documents as a function of their size.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 3 of 13

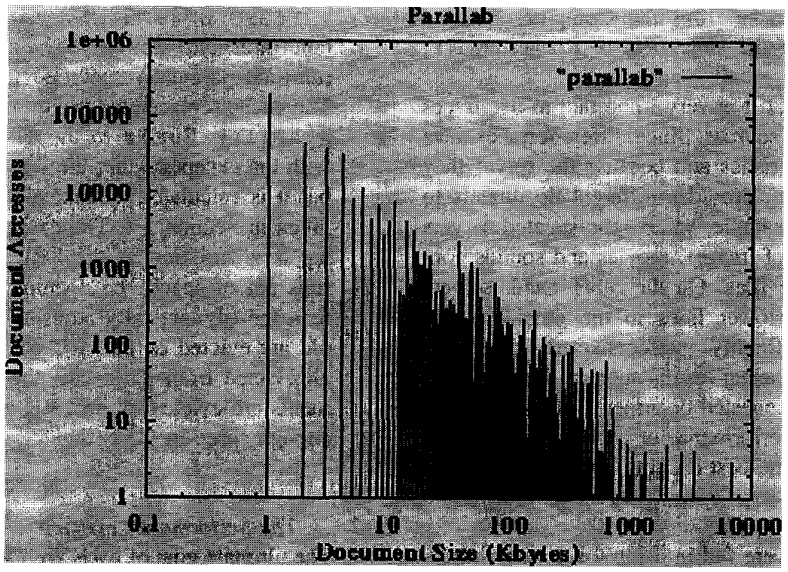


Fig. 2. Number of accesses to documents as a function of their size.

- *Document Hit Rate*: This is the ratio of the number of documents found in the cache to the number of documents requested.
- *Byte Hit Rate*: This is the ratio of the number of bytes brought from the cache to the number of bytes requested by the clients.

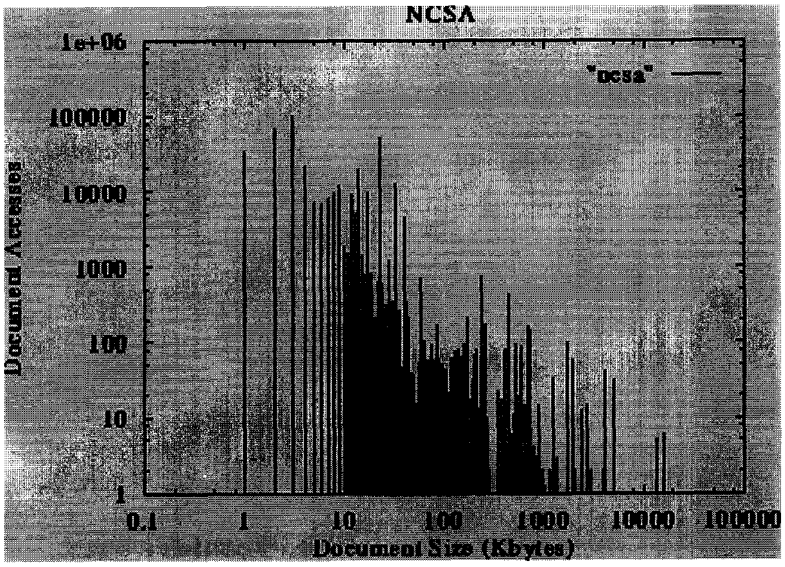


Fig. 3. Number of accesses to documents as a function of their size.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 4 of 13

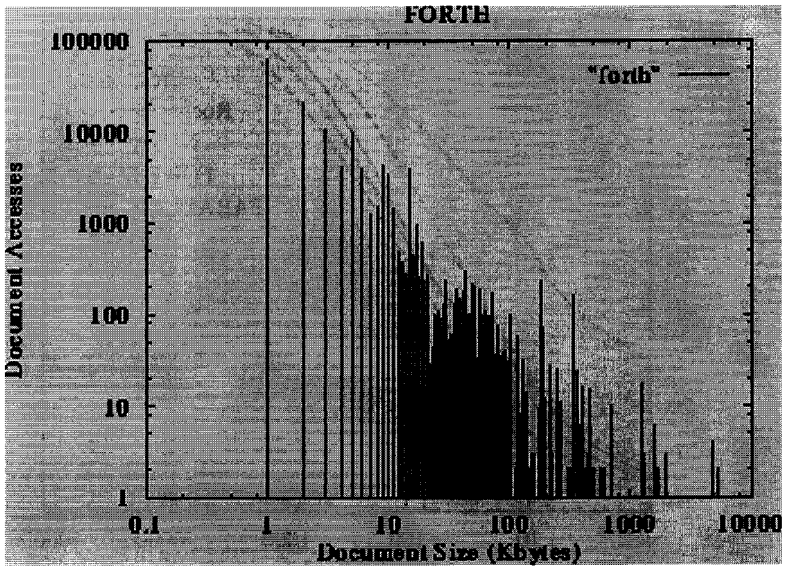


Fig. 4. Number of accesses to documents as a function of their size.

The Document Hit Rate seems to be the most important performance metric, since the higher it is the more client requests will see a low latency. On the other hand, when the Document Hit Rate does not vary significantly, the Byte Hit Rate should be given serious consideration because it represents the number of bytes serviced from the cache, and thus do not have to be read from the disk.

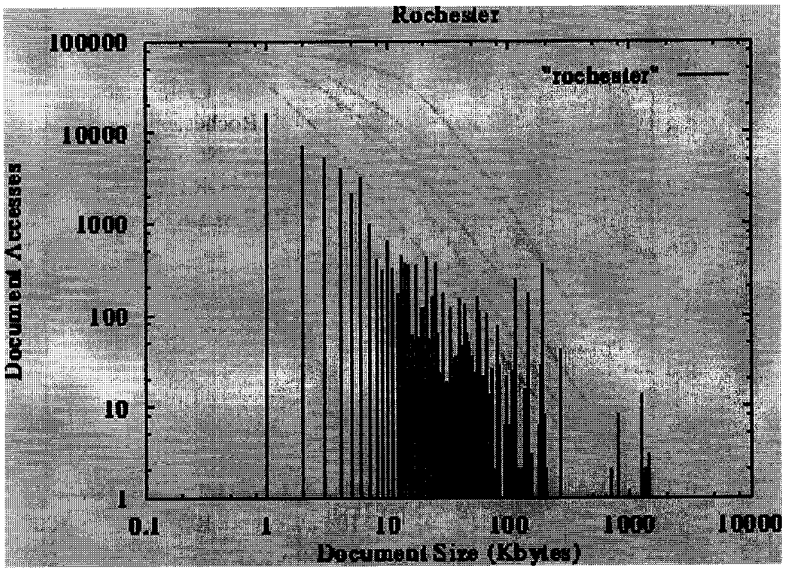


Fig. 5. Number of accesses to documents as a function of their size.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 5 of 13

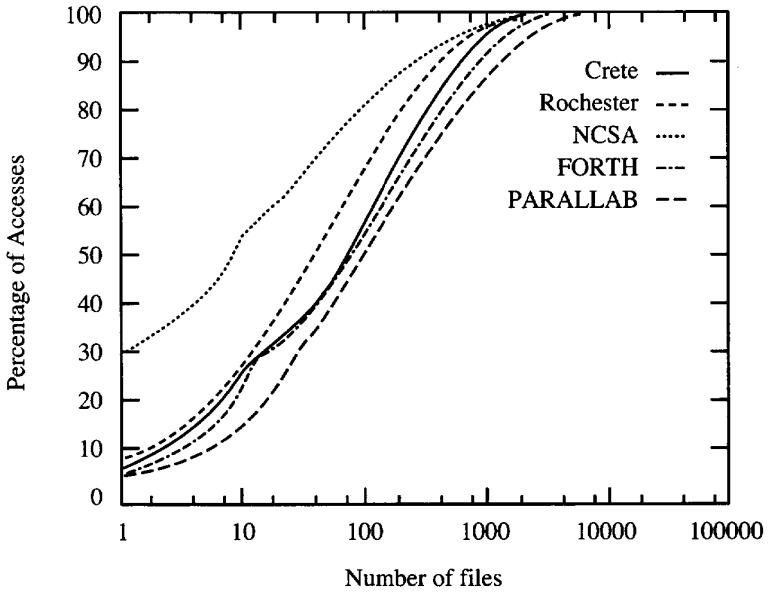


Fig. 6. Number of documents responsible for a given percentage of Web references.

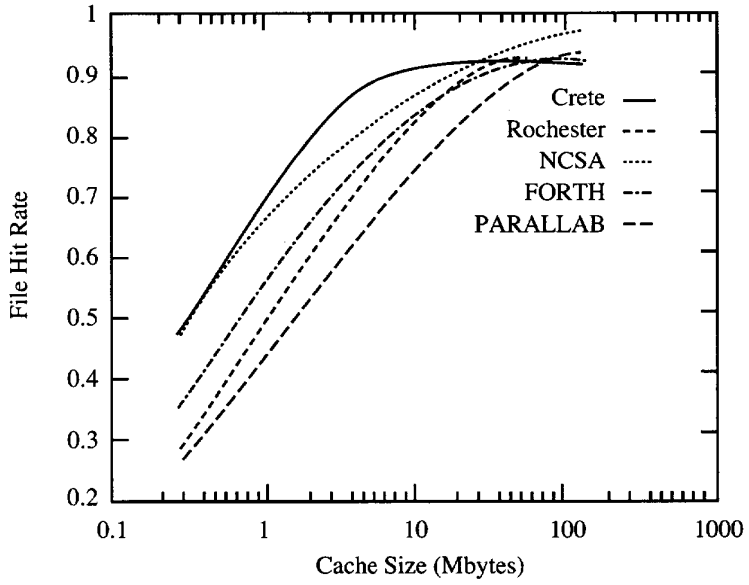


Fig. 7. Document Hit Rate as a function of the Cache Size.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 6 of 13

2.1.4. Trace characteristics

The document requests as a function of their size are shown in Figs. 1–5. The x-axis represents the size of the document, while the y-axis represents the number of references to documents of a given size. For example, Fig. 1 suggests that more than 100 000 accesses request documents of size 1 Kbyte or less. The traces suggest that small documents (a few Kbytes long) tend to be accessed much more frequently than larger documents. This preference to small-sized documents holds for all different traces studied, and has been observed by other researchers as well [2].

To illustrate the preference to small documents even further, we plotted the cumulative document requests for the various servers as a function of the number of files responsible for this request. Fig. 6 shows this cumulative document request for each server studied. We see that for all traces, a small number of files are responsible for a significance percentage of the Web requests. For example as little as 10 files are responsible for 50% of the accesses in the NCSA trace. Similar observations hold (to a lesser extent) for the other traces as well. As little as 100 files are responsible for 35–50% of the traffic in other traces. This trend is particularly encouraging for our work, since it implies that caching a small number of files is enough to service a significant percentage of the requests.

2.2. The effect of cache size

In this first set of experiments we want to explore the effect that the cache size has on the performance of caching policies. We are interested in answering the following questions:

- Is it possible to get hit rates as large as 80%? or even 90%?
- How large a cache is needed to get these hit rates?

Fig. 7 plots the Document Hit Rate as a function of the cache size, for the ALL policy that caches all documents and replaces them from the cache on an LRU basis. Our results suggest that the size of the cache needed to achieve a given hit rate depends on the Web server. We see that a cache as small as 4 Mbytes results in 80% hit rate for the NCSA server, but only in 63% hit rate for the Parallab server. Even more, to achieve a 90% document hit rate, 16 Mbytes of cache are enough for almost all

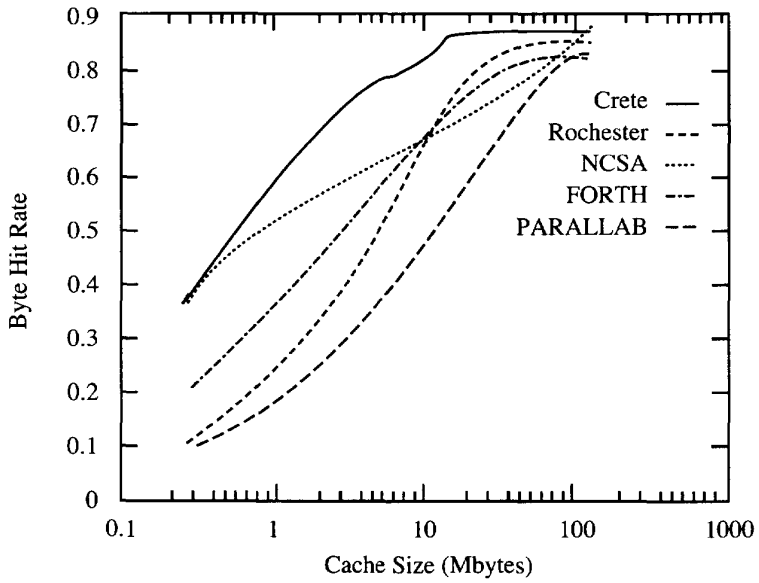


Fig. 8. Byte Hit Rate as a function of the Cache Size.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 7 of 13

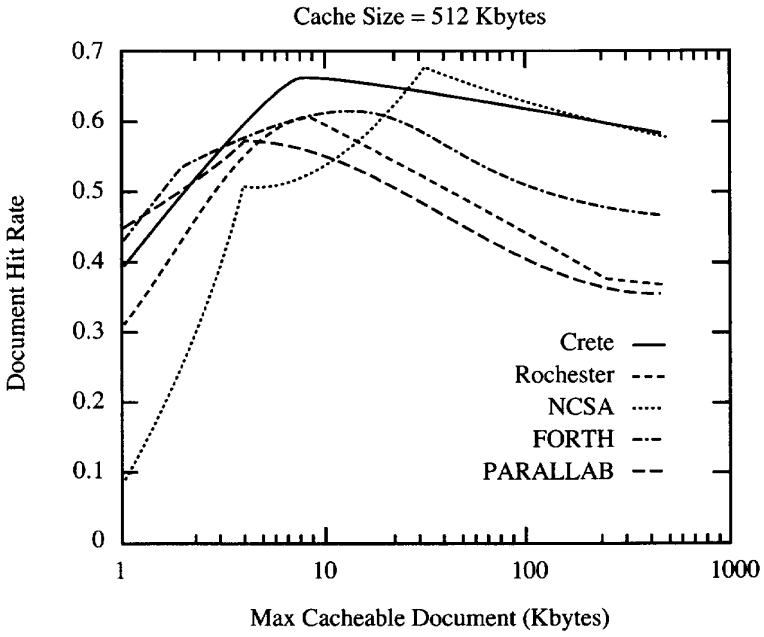


Fig. 9. Document Hit Rate as a function of the maximum size of cacheable documents.

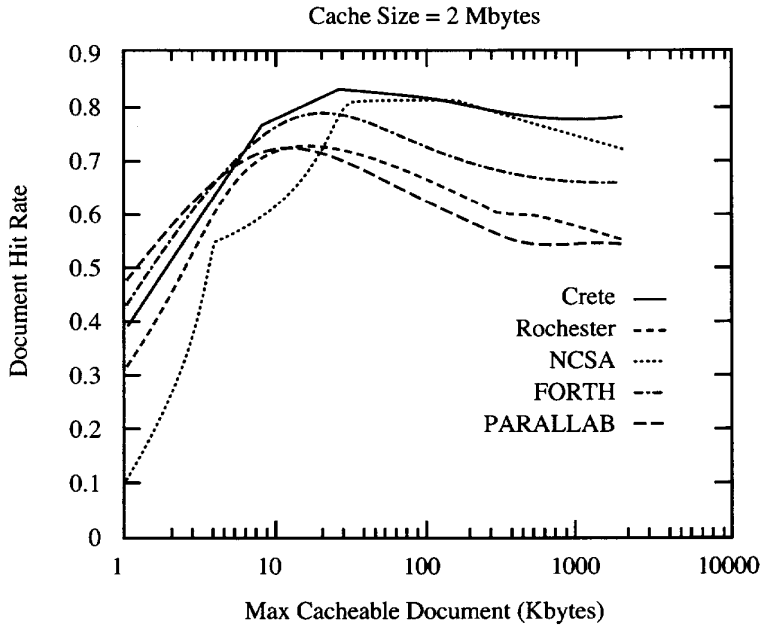


Fig. 10. Document Hit Rate as a function of the maximum size of cacheable documents.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 8 of 13

servers except the Parallab, which needs 64 Mbytes to achieve the same levels of performance.

We can see the reason why in Fig. 6 where it can be easily seen that NCSA has very good locality of reference, while Parallab has poor locality. For example, Fig. 6 shows that the ten most frequently accessed files in NCSA are responsible for almost 50% of the accesses, while the ten most frequently accessed files in Parallab are responsible for about 10% of the accesses. Thus, we expect that in order to achieve similar hit rates with NCSA, the Parallab server needs to cache more files, which will (probably) require a larger cache.

We reach similar conclusions by looking at Fig. 8, which displays the byte hit rate as a function of the cache size. We should note, however, that for the same cache size and the same server, the Document Hit Rate is higher than the Byte Hit Rate. The reason is simple: in order to achieve really high byte hit rates, large documents should be cached, which require really large caches. Caching large documents in small caches will evict from the cache lots of more frequently used small documents and will probably be brought back into the cache soon after they are evicted.

2.3. Caching "small" documents

One remedy to the problems introduced by caching large documents is *not* to cache large documents at all. Thus, if a document is larger than a threshold value, the document is not cached. Although it may sound simple, it is really a complicated trade-off to decide which documents are large and should not be cached. If the threshold is chosen too small, then the cache may be underutilized. If the threshold is chosen too large, then the documents may end up thrashing for the same cache space, and no document will stay in the cache long enough to achieve a decent hit rate. To understand the performance implications of this threshold, we simulated the THRESHOLD caching policy that caches documents whose size is less than or equal to the threshold value. The performance of caching in the various Web servers as a function of the caching threshold is shown in Figs. 9–11 for cache sizes of 0.5 Mbytes, 2 Mbytes and 8 Mbytes respectively.

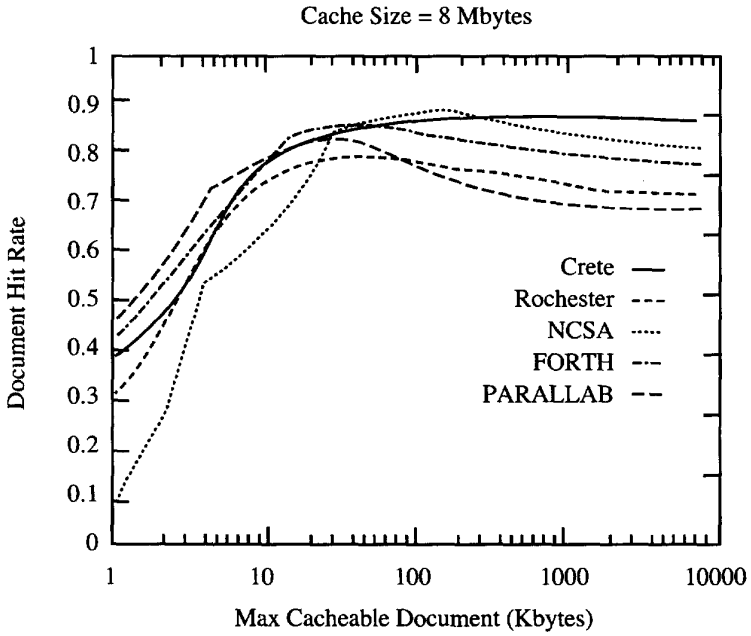


Fig. 11. Document Hit Rate as a function of the maximum size of cacheable documents.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 9 of 13

The first thing we observe is that the performance increases with the threshold value up to a maximum and then it starts to decrease, which is in accordance with our intuition: increasing a small threshold makes better use of the cache by bringing in more documents, and thus improves performance. But if the threshold is increased beyond a certain point, too many documents will compete for the same cache space, and they will end up pushing each other out of the cache. Effectively, few documents will manage to stay in the cache long enough to achieve a decent hit rate. By looking closely at Fig. 9, we see that the optimal threshold and the achievable performance for each Web server is different. For example, the optimal threshold for NCSA is 32 Kbytes, while the optimal threshold for Parallab is 4 Kbytes. This is because NCSA tends to serve larger documents than Parallab. The average size of the requested documents from Parallab is 7.5 Kbytes, while the average size from the requested documents from NCSA is 17 Kbytes. By comparing Fig. 9 with Figs. 10 and 11 we see that the achievable performance and the optimal threshold increases with the available cache size. For example, If we increase the cache size from 512 Kbytes to 8 Mbytes, the optimal threshold for NCSA moves from 32 Kbytes to 128 Kbytes, and for Parallab moves from 4 Kbytes to 32 Kbytes.

Our results suggest that the optimal threshold depends both on the documents requested, and on the cache size. Choosing a threshold that will result in good performance is a delicate procedure. Naively chosen values of the threshold may result is poor performance. For example, Fig. 9 suggests that in the Rochester trace, the optimal threshold results in 62% Document Hit Rate, while the policy that uses no threshold at all results in only 36% Document Hit

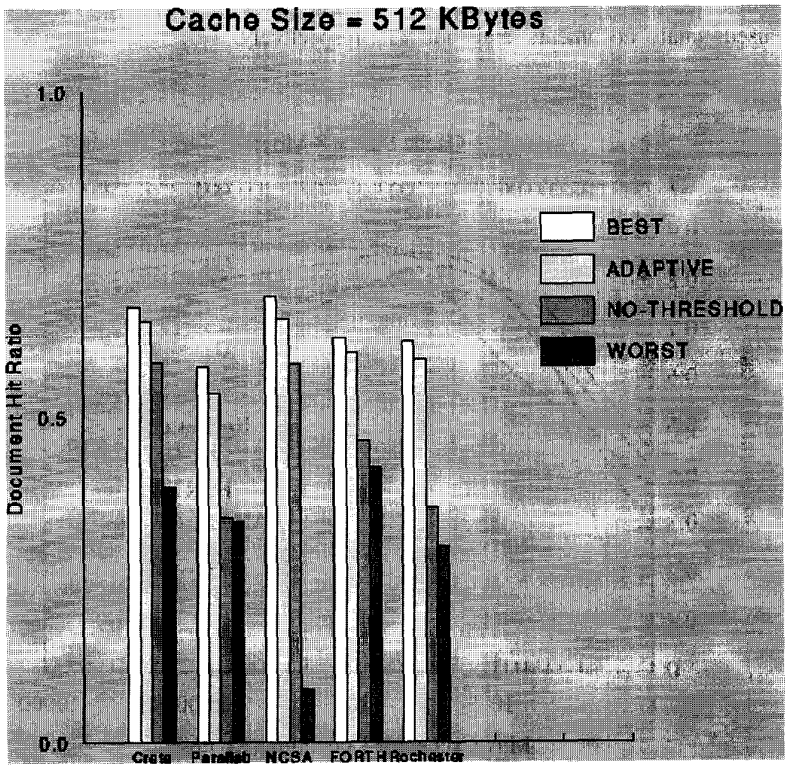


Fig. 12. Performance of the adaptive policy.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 10 of 13

Rate. It seems that for any threshold we choose there is a server and a cache size that perform very poorly for this threshold.

2.4. Adaptive threshold tuning

To eliminate the need for off-line careful threshold tuning, and to provide responsiveness to varying access patterns, we have developed an ADAPTIVE caching policy that estimates the best threshold at run-time without user intervention. The intuition behind the policy is simple:

- Start with an initial threshold.
- Periodically increase (decrease) the threshold. If the performance gets better, continue increasing (decreasing) it. Otherwise, start decreasing (increasing) the threshold.

In our implementation we used an initial value of 16 Kbytes (which is a reasonable start), and an increment (decrement) of 2 Kbytes. We updated the threshold value every 5000 references. The metric we use to estimate performance improvements is the Document Hit Rate. To make the policy stable, we reverse the direction of changing the threshold from increasing to decreasing (and vice versa) only if the decrease in performance is larger than 1%. We felt that performance decreases lower than 1% are usually due to statistical noise and are not worth changing the direction of the threshold calculation.

The performance of the ADAPTIVE policy is shown in Figs. 12 and 13 for cache sizes of 512 Kbytes and 8 Mbytes respectively. We compare the ADAPTIVE policy with the following policies:

- BEST: This is the best performance of the THRESHOLD policy observed for any threshold between 1 Kbyte and 2 Mbytes.
- WORST: This is the worst performance of the

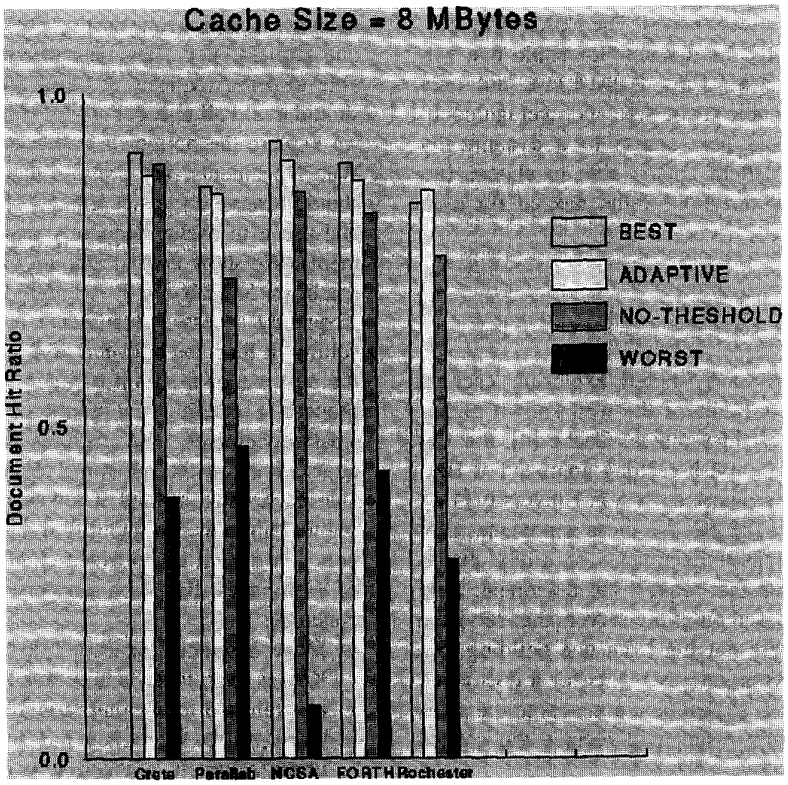


Fig. 13. Performance of the adaptive policy.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 11 of 13

THRESHOLD policy observed for any threshold between 1 Kbyte and 2 Mbytes.
- NO-THRESHOLD: This is our known ALL policy that caches all documents irrespective of their size.

We see that in all cases the ADAPTIVE policy is close to the BEST because it manages to approximate the best threshold value based on run-time performance measurements. The performance when NO-THRESHOLD is used is always inferior to the ADAPTIVE policy, sometimes substantially so (see Parallab in Fig. 12). The difference between ADAPTIVE and NO-THRESHOLD is much more pronounced when the cache size is small (512 Kbytes).

To our surprise we saw that in one case, ADAPTIVE was even better then the BEST policy (see Rochester in Fig. 13). This is because in one part of the traces a small threshold was appropriate, while for another part of the traces a larger threshold was appropriate. Thus no static threshold was able to deliver the best possible performance; a dynamically changing threshold was needed. The ADAPTIVE policy provided exactly the dynamic threshold that was needed for this case. Thus, ADAPTIVE not only adapts to the available cache size and average document requested, but also adapts to a dynamically changing access pattern and is able to choose (almost) the optimal threshold value needed for each case.

3. Related work

Several groups have started exploring caching issues on the Web. Some of the well known caches include the Harvest hierarchical Cache [4], the Lagoon system [3], the push-caching approach at Harvard [5], the caching work at Boston University [1,2], the Hensa Archive [8], etc. All these approaches deal with caching of remote documents on a disk that is close to the clients that request the document. Several of them are based on proxy servers that service a community of users ranging from one building to a whole country. When one user asks for a document, the proxy caches it, so that next accesses to the same document by nearby users will receive the cached copy.

Our research however, is complementary to previous caching approaches, and deals with a different form of caching: caching of *local* documents in a Web server's main memory, so that they can be sent faster to clients that request them.

Our approach is in several ways similar to file system caching [7]. However, Main Memory Caching of Web Documents differs from file system caching mainly because Web access patterns are different from file access patterns: Web clients access *entire* Web documents, in *read-only* mode, (usually) *only once*. On the other hand, file systems access *portions* of files, usually in *read-rite* mode, *repeatedly*.

For the above reasons we believe that caching policies used in file systems should not be used "per se" in Main Memory Web caches as well. Our performance results strengthen our point even more.

4. Conclusions

Recent Web servers receive an ever-increasing amount of Web requests that reach peaks of up to one request every few tens of milliseconds. To be able to serve requests at these rates, a Web server should avoid all unnecessary overheads. In this paper we explore the notion of Main Memory Caching of Web Documents. We propose that each Web server should reserve a (small) amount of its main memory and use it to cache frequently requested Web documents. We show that an amount of main memory as small as 512 Kbytes is enough to hold the most frequently accessed documents of Web servers, resulting in hit rates between 55% and 70%. To make the best use of this limited main memory cache, we propose a caching policy that prefers caching of small documents. The caching policy dynamically adjusts itself to the incoming requests and the available cache size so that it always achieves (almost) optimal hit rates. To quantify the performance of our policy, we used trace-driven simulation of a variety of Web traces. Based on our performance results we conclude:
- Most frequently accessed documents are a few Kbytes long; thus caching small documents should be preferred to caching large documents.
- Our ADAPTIVE policy has shown to be robust under a variety of Web server loads, and cache configurations. Our policy adapts to the available

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 12 of 13

cache size and the clients' patterns of requests so that to cache documents up to the size that optimally uses the main memory cache.

- No static caching policy is likely to provide good results under all Web server access patterns. Our traces suggest that some servers should cache documents up to 32 Kbytes, while other servers should cache documents up to 4 Kbytes. Thus, any policy that does not take into account the characteristics of the documents requested is bound not to perform well for all Web servers.

Acknowledgements

Part of this work has been supported by the ESPRIT/OMI project "ARCHES" (20693). We deeply appreciate this financial support. We also thank the University of Rochester, the University of Bergen, ICS-FORTH, NCSA, and the University of Crete, for providing us with traces of their Web servers. Catherine Chat and Bob Hopgood contributed countless important improvements to the appearance and contents of this paper. Catherine Chronaki, George Dramitinos, Sotiris Ioanidis, and Richard Uhlig provided many useful comments in earlier versions of this paper. We thank all of them.

References

[1] Azer Bestavros, Demand-based document dissemination to reduce traffic and balance load in distributed information systems, in: *Proc. 1995 7th IEEE Symp. on Parallel and Distributed Processing* (1995).

[2] Azer Bestavros, Bob Carter, Mark Crovella, Carlos Cunha, Abdelsalam Heddaya and Suliman Mirdad, Application-level document caching in the Internet, in: *Proc. SDNE'95: 2nd Internat. Workshop on Services in Distributed and Network Environments* (1995). http://cs-www.bu.edu/faculty/crovella/paper-archive/www-sdne95/paper.html

[3] P.M.E. De Bra and R.D.J. Post, Information retrieval in the World-Wide Web: Making client-based searching feasible, in: *Proc. 1st Internat. WWW Conf.* (1994).

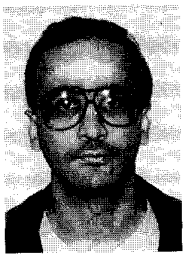
[4] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz and Kurt J. Worrell, a Hierarchical Internet object cache, Tech. Rept. 95-611, Computer Science Dept., University of Southern California, Los Angeles, CA (1995). http://excalibur.usc.edu/cache-html/cache.html

[5] J. Gwertzman and M. Seltzer, The case for geographical pushcaching, in: *Proc. 1995 Workshop on Hot Operating Systems* (1995).

[6] Jeffrey C. Mogul, Network behavior of a busy Web server and its clients, Tech. Rept., Research Report 95/5, DEC Western Research Laboratory (1995).

[7] M. Nelson, B. Welch and J. Ousterhout, Caching in the Sprite Network File System, *ACM Trans. Comput. Systems* 6 (1) (1988) 134–154.

[8] Neil Smith, The UK national World-Wide Web proxy cache at HENSA Unix, 1995. http://www.hensa.ac.uk/wwwcache/intro.html



Evangelos P. Markatos is an Assistant Professor at ICS-FORTH and at the University of Crete, Greece. He received his diploma in Computer Engineering from the University of Patras in 1988, and the M.S. and Ph.D. degrees from the University of Rochester, NY, USA, in 1990 and 1993 respectively. His interests include parallel and distributed systems, operating systems and computer architecture.

Microsoft - Exhibit 1021
Microsoft v. Sterling - IPR2025-00270
Page 13 of 13