Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers

Martin F. Arlitt
Carey L. Williamson
Department of Computer Science
University of Saskatchewan
57 Campus Drive
Saskatoon, SK, CANADA S7N 5A9
{mfa126,carey}@cs.usask.ca

September 11, 1996

Abstract

Given the continued growth of the World-Wide Web, performance of Web servers is becoming increasingly important. File caching can be used to reduce the time that it takes a Web server to respond to client requests, by storing the most popular files in the main memory of the Web server, and by reducing the volume of data that must be transferred between secondary storage and the Web server.

In this paper, we use trace-driven simulation to evaluate the effects of various replacement, threshold, and partitioning policies on the performance of a Web server. The workload traces for the simulations come from Web server access logs, from six different Internet Web servers. The traces represent three different orders of magnitude in server activity and two different orders of magnitude in time duration.

The results from our simulation study show that frequency-based caching strategies, using a variation of the LFU (Least Frequently Used) replacement policy, perform the best for the Web server workload traces considered. Thresholding policies and cache partitioning policies for Internet Web servers do not appear to be effective.

1 Introduction

The tremendous growth in the popularity of the World-Wide Web (WWW, W3 or "the Web") has sparked a number of research efforts aimed at improving the scalability and performance of the World-Wide Web [1, 2, 4, 6, 9, 15, 20]. The range of performance enhancements considered can usually be classified as either platform-oriented, protocol-oriented, or caching-oriented.

Platform-oriented enhancements refer to the obvious approaches of simply upgrading the Web server platform, or at the very least making sure that the Web server is properly configured [22]. The end result should be a Web server with a CPU powerful enough to process all of the client requests, and `httpd` software that can accept multiple simultaneous requests, in an efficient manner [22].

Protocol-oriented approaches focus on the HTTP and TCP protocols used for exchanging documents between Web clients and Web servers. These approaches try to reduce the latency for retrieving Web documents, either by streamlining the protocols used, improving the implementation of the protocols in terms of operating system overhead, or redesigning the protocols to operate on a "session basis" to better hide the round trip latencies inherent in per-document TCP connection setup and teardown [16, 17, 19].

Finally, caching-oriented approaches focus on how classical file caching strategies can be used to improve Web server performance. By caching documents at the Web server (e.g., in main memory, or on local disk), at the client (e.g., in main memory, or on local disk), or within the network (e.g., at a proxy server or network cache, in memory or on disk), access times can be reduced for repeated accesses to popular Web documents, by multiple users from multiple sites. Many approaches have been proposed and analyzed for Web document caching, including the Harvest cache [5], push caching [10], and proxy caches [1, 14, 20].

This paper focuses solely on the caching-oriented approach. Trace-driven simulation is used as the vehicle for evaluating the relative performance impacts of different document caching strategies for Internet Web servers, similar to work done by other researchers [1, 4, 6, 9, 13, 15, 20].

This work was motivated by the results of our workload characterization study of Internet Web Servers [2]. First, our design of Web server document caching strategies is guided, in part, by several of the Web server workload invariants identified in [2]. These invariants provide intuitive explanations for some of the problems identified in traditional document caching strategies, and suggest improved strategies. Second, our performance study uses Web server access logs from six different sites on the Internet. The traces from these six sites represent servers with very different workloads (ranging from an average of 776 requests per day to an average of 355,787 requests per day), and time durations ranging from one week to one year (these traces are characterized in [2]). Our analysis and evaluation of document caching strategies are based on the desire to provide consistently strong performance across all of the workload traces considered. Third, our study uses *both* cache hit rate and data volume as performance metrics, since (as first argued in [2]) optimizing a cache for hit rate only may not provide a complete solution to the Web server performance problem.

Our search for better Web server caching strategies is in fact driven by the following six observations (taken from Arlitt and Williamson [2]), each of which has obvious implications on the performance of a Web server cache:

Observation 1: *There is an extremely high concentration of Web server requests to a relatively small set of highly popular documents.* Typically, 10% of the documents account for well over 90% of the requests. This observation suggests that document caching for Internet Web servers can be extremely effective.

Observation 2: *There is a distinct difference between designing a Web server cache to maximize cache hit rate versus designing a Web server cache to maximize savings in bytes transferred (i.e., data volume).* Optimizing one metric does not necessarily optimize the other. The choice of which metric is more important depends on whether the Web server is CPU-bound (i.e., the Web server is cycle-limited for handling requests), or I/O-bound (i.e., disk bandwidth or network bandwidth is the performance bottleneck).

Observation 3: *The document access patterns found in Web server access logs show very little temporal locality.* This behaviour, which may result from client-side caching [8], or from the sheer volume and diversity of Web documents and Web users, means that traditional recency-based policies (e.g., LRU) may not perform well for Internet Web servers.

Observation 4: *Many Web server documents are accessed only once, ever.* Often, up to 33% of the accessed Web documents fall into this category, even in very long (e.g., one year) Web server traces. There is clearly no benefit to caching such "one-timers"', and in fact doing so (i.e., caching them) may adversely affect cache performance[1]. Intuition suggests that frequency-based policies (e.g., LFU) may provide better discrimination against these "one-timers".

Observation 5: *The distribution of document sizes on Web servers is heavy-tailed.* This observation means that while most of the accessed documents are small, there are several extremely large documents that, even if accessed very rarely, may dominate the total data volume transferred by the server. Furthermore, caching such documents may displace a large number of other documents from the cache. Whether it is worthwhile to cache such large documents or not is an issue to be investigated.

Observation 6: *Different document types (e.g., HTML, image, audio, video, postscript) have very different size characteristics, and very different access frequencies.* This observation suggests that cache partitioning, based on document type, may be a worthwhile strategy for Internet Web servers. With this approach, different document types can be cached separately (and perhaps differently), without interfering with other document types.

The purpose of this paper, then, is to evaluate Web server document caching strategies in light of these observations, and to assess their relative impact on server performance. In particular, the three specific research questions addressed in this paper are:

1. What type of cache replacement policy is most effective for Internet Web servers: frequency-based, recency-based, or size-based?
2. Are size-based thresholding policies, which restrict the size of documents allowed into a cache, worthwhile for Internet Web servers?
3. Is cache partitioning based on document type a useful approach to cache management?

Our results, in a nutshell, demonstrate the clear superiority of frequency-based approaches (particularly, the LFU*-Aging policy defined in Section 3.1.2) over recency-based and size-based approaches. Furthermore, the results demonstrate the ill-advised nature of size-based thresholding and cache partitioning.

The remainder of this paper is organized as follows. Section 2 provides background on Web servers, the Web server caching architecture assumed in our study, and the workloads used in our trace-driven simulations. Section 3 describes the experimental design for our simulation study, including the factors, levels, and metrics used. Section 4 presents the results of our study, and Section 5 our conclusions.

2 Background

This section provides a brief overview of Web server operation, our Web server simulation model, and the workload traces used for our trace-driven simulations.

[1] The advent of *Web Crawlers* may change "one-timers" to "N-timers", for some small integer N. However, caching any rarely requested document may degrade cache performance.

2.1 Web Overview

The World Wide Web is based on the client-server model. A client accesses documents on the Web via a Web browser. The browser sends a request to a Web server, which responds with the requested document. Although the information may be stored in almost any location throughout the world, the Web provides the user with the illusion that the data is stored locally.

A Web server can respond to requests from multiple Web clients. Communication is always in the form of request-response pairs, and is always initiated by the client. Web clients and servers communicate using the HyperText Transfer Protocol (HTTP). HTTP runs on top of TCP, a reliable bidirectional byte stream protocol at the transport layer.

The Web server itself operates as follows. The server listens on a designated port (usually port 80) for a request from a Web client to establish a TCP connection. Once a TCP connection has been opened and the client has made its request, the server must respond to that request. The response includes a status code to inform the client if the request succeeded. If the request was successful, then the response includes the requested document. If the request was unsuccessful, a reason for the failure is returned to the client [17, 22]. Once the Web server has sent its response and terminated the TCP connection with the client, the server repeats the cycle and begins listening for its next request.

The role of document caching for Internet Web servers is simply to shorten the response time for Web clients. By having a document cached (e.g., in memory, rather than on local disk, or on local disk rather than on a remote disk at the file server) a document can be retrieved and returned to the client sooner. Returning a document from the cache can also save on the number of I/O requests (and thus bytes transferred) either to the local disk or to the file server.

2.2 Web Server Simulation Model

For this performance study, a very simple Web server is assumed, with a single-level file cache located within the address space of the `httpd` software [15]. From this location, the file cache is able to see all requests to the Web server. A file cache in this position does not reduce the number of requests that the server must process, nor does it reduce the volume of traffic on the network link between the Web server and the client. Instead, this file cache can improve the performance of the server by reducing the time it takes the server to respond to requests, since memory access time is typically an order of magnitude faster than disk access time.

Our Web server simulation model operates as follows. When a request is received by the Web server, it looks for the requested file in its file cache. A *cache hit* occurs if a copy of the requested document is found in the file cache. If the document is not found in the file cache (a *cache miss*), the document must be retrieved from the local disk or the file system where the document is permanently stored (i.e., secondary storage).

On each cache hit, the file cache reduces by one the number of requests sent to secondary storage. Since the cache is located within the address space of the Web server, each cache hit also reduces by one the number of interactions with the operating system [15]. Furthermore, each cache hit results in a reduction of the number of bytes transferred to the server from secondary storage (i.e., a reduction in the data volume). Since the Web server can retrieve a document from its file cache much quicker than it can retrieve a copy from secondary storage, the time needed to process the request is reduced. This reduction in response time assumes that the transfer of documents from secondary storage represents the performance bottleneck.

On each cache miss, the Web server must retrieve a copy of the requested document from secondary storage. When a copy of the document is returned to the Web server, the document may be added to the Web server's file cache (depending on the replacement, threshold, and partitioning policies that are in place at the file cache). At this point the server can complete its response to the client that made the original request.

For simplicity, our simulation model ignores the issues of *cache consistency* (i.e., making sure that the cache has the most up-to-date version of a document, compared to the master copy version at the server, which may change at any time), *cache security* (i.e., users specifically designating private documents as non-cacheable, so that unauthorized copies do not end up in client, server, or network caches), and *dynamic files* (i.e., Web server requests that ask for the remote execution of a particular file, which then results in different contents returned for each such request). These issues would unduly complicate our simulation model. Furthermore, the workload characteristics reported in [2] suggest that such requests are relatively rate (e.g., file modification events typically affect less than 2% of requests, and dynamic requests account for less than 3% of server requests).

The model also assumes file-level caching. File-level caching differs from block-level caching in one small but important way: a file cache holds a variable number of files (which can have different sizes), while a block cache contains a fixed number of equal-sized data blocks. On a cache miss, a file cache removes zero or more files from the cache, until there is sufficient space to add the new file. This differs from block-level caching, which replaces at most one data block at a time on a cache miss. In a file cache, only complete documents are cached; when a file is added to the cache, the whole file is added, and when a file is removed from the cache, the entire file is removed. A file cache never keeps only part of a file, unlike a block cache.

2.3 Workload Traces

The data used in the trace-driven simulations come from Web server *access logs*. Web servers are typically configured to produce such a log, which records information about each client request, including the time of the request, the name of the document requested, the response code for the request, and the number of bytes transferred, if any.

In this research, logs from six different Web servers were used: a lab level Web server at the University of Waterloo (Shoshin Research Lab in the Department of Computer Science); a department-level Web server at the University of Calgary (Department of Computer Science); a campus-wide server at the University of Saskatchewan; the Web server at NASA's Kennedy Space Center; the Web server from ClarkNet, a commercial Internet provider in the Baltimore-Washington D.C. region; and the Web server at the National Center for Supercomputing Applications (NCSA) in Urbana-Champaign, Illinois. These access logs were previously used in a workload characterization study of Internet Web servers [2]. A summary of the original logs is given in Table 1.

Table 1: Summary of Access Log Characteristics (Raw Data)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Access Log Duration	8 months	1 year	7 months	2 months	2 weeks	1 week
Access Log Start Date	Jan 1/95	Oct 24/94	Jun 1/95	Jul 1/95	Aug 28/95	Aug 28/95
Access Log Size (MB)	17.9	49.9	222.6	355.8	327.5	267.7
Total Requests	188,636	726,739	2,408,625	3,461,612	3,328,587	2,490,512
Avg Requests/Day	776	2,059	11,255	56,748	237,756	355,787
Bytes Transferred (MB)	2,071	7,577	12,343	62,489	27,647	28,268
Avg Bytes/Day (MB)	8.5	21.5	57.7	1,024.4	1,974.8	4,038.3

Only a portion of the original data is needed for the file cache simulations, resulting in a dramatic reduction in the size of the traces. First, since the *Successful* responses are responsible for all of the

Table 2: Summary of Trace Characteristics (All Data Sets)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Duration	8 months	1 year	7 months	2 months	2 weeks	1 week
Start Date	Jan 1/95	Oct 24/94	Jun 1/95	Jul 1/95	Aug 28/95	Aug 28/95
Size (MB)	1.7	5.8	22.8	30.9	31.8	22.8
Requests	163,112	567,795	2,165,415	3,087,043	2,940,712	2,289,510

documents transferred by the server, only these responses (approximately 88% of all responses [2]) are used in the cache simulation. Further reductions are possible, as only two parts of each *Successful* response are needed. A normal access log has the form: `hostname - - [dd/mmm/yyyy:hh:mm:ss tz] request status bytes]`. The only information needed in this performance study is the name of the requested document (`request`) and the size of the requested document (`bytes`). To further reduce the size of the trace (and to improve the performance of the simulator), each trace entry consists of only three integers: one to uniquely identify the file, the second to indicate the size of the file, and the third to indicate the file type (this last integer is needed for the partitioning policy). Table 2 provides a summary of the resulting trace files used in this performance study.

3 Experimental Design

This section describes the design of the Web server file cache performance study. The discussion begins with the factors and levels used for the simulations, and then presents the performance metrics used in the study.

3.1 Factors and Levels

There are four main factors in our trace-driven simulation experiments: cache size, cache replacement policy, cache threshold policy, and cache partitioning policy. This section details the levels used for each of these factors.

3.1.1 Cache Sizes

The first factor in this study is the size of the file cache. Since entire files are being cached, rather than data blocks, the size of the file cache is given as a number of megabytes (MB), rather than as a number of blocks. In total, 12 different levels are used. They are: 1, 2, 4, 8, 16, 32, 64, 96, 128, 256, 512, and 1024 MB.

There are two reasons for choosing this set of levels. First, this set of levels clearly spans the relevant range of interest for RAM cache size on today's Web servers (e.g., 8 MB to 96 MB). Second, the largest file cache size studied (1024 MB) is large enough to cache the entire document set on any of the Web servers under study. The results for the 1024 MB file cache can therefore be interpreted as the results for an infinite-sized file cache, or a file cache that never removes a document once the document has been added to the cache. Using a cache of this size indicates the maximum achievable cache hit rate, and the minimum achievable data volume. The performance of smaller cache sizes can then be compared to the performance of the infinite cache. Such a comparison can be used to justify how large a file cache should be, and to determine at what point increasing the cache size does not result in significant performance improvement.

3.1.2 Cache Replacement Policies

The second factor in this study is the cache replacement policy. This policy determines which document (or documents) should be removed from the file cache in order to create sufficient space for the most recently requested document (documents are replaced only when a cache miss occurs, and there is insufficient free space in the file cache to add the new document).

Four common replacement policies are examined in this study: Least Recently Used (LRU), Least Frequently Used (LFU), Least Frequently Used with Aging (LFU-Aging), and Frequency Based Replacement (FBR). In addition, two new approaches are considered: size-based replacement, as proposed by Williams *et al.* [20], and a modified version of LFU, proposed by Arlitt [3].

The LRU policy removes from the cache the file that has not been used for the longest period of time. LRU works well in reference streams that exhibit the property of *temporal locality* (this property suggests that files that have been referenced in the recent past will likely be referenced again in the near future). The LRU policy is frequently used in computer systems (e.g., virtual memory caches, file system caches, instruction/data caches).

The LFU replacement policy maintains a reference count for each file in the cache. On each cache hit, the reference count of the requested file is incremented. On each cache miss (if there is insufficient space to add the requested file) the file with the lowest reference count is selected for replacement. In the case of a tie, the least recently used file with the lowest reference count is replaced.

The LFU-Aging policy is similar to the LFU policy in that the least frequently used document is replaced when space is required for a new document. However, LFU-Aging attempts to deal with a problem of LFU. With LFU, some documents can build up extremely high reference counts so that they are rarely (if ever) replaced, even if these documents are never requested again. The LFU-Aging policy attempts to ensure that this condition does not occur by limiting and aging (i.e., occasionally reducing) reference counts [18, 21].

The FBR policy makes a replacement decision using a combination of reference frequency and age [18, 21]. FBR divides the cache into three sections: a new section, a middle section, and an old section. When a file is initially added to the cache, it is placed at the start of the New Section (most recently used, or MRU). As the file ages (on an LRU basis), it will be removed from the New Section and added to the Middle Section. If the file continues to age (i.e., it remains in the cache but is not re-referenced) it will eventually be removed from the Middle Section and added to the Old Section. Only files in the Old Section are candidates for removal from the cache by the replacement strategy. On every cache hit, the referenced file is always moved to the start of the New Section (MRU). If the file was found in either the Middle or Old Sections, its reference count is incremented. If the file was already in the New Section, its reference count is not incremented [18, 21]. On a cache miss that requires a cache replacement decision, the least frequently used file in the Old Section is replaced (on a tie the least recently used file with the lowest reference count is chosen).The FBR policy also uses an aging policy.

In a study of Web proxy cache performance, Williams *et al.* find that a size-based replacement policy works better than traditional approaches like LRU or LFU [20]. In general, the goal is to remove large files from the cache, in order to make room for more small files. The primary replacement decision can be based on either the size (i.e., the exact size) or $\lfloor log_2(Size) \rfloor$ (i.e., the approximate size) of the file that is to be added to the cache. Any other policy (such as LRU or LFU) can be used to break a tie.

One common characteristic of Web server workloads identified by Arlitt and Williamson was the presence of a significant number of "one-timers" [2]. Obviously, caching these documents would only degrade the performance of the file cache. Therefore, an attempt was made to minimize the performance impact of these "one-timers'. This was done by modifying the LFU replacement algorithm. The modified algorithm is denoted LFU*, to distinguish it from the standard LFU algorithm. LFU* is similar to the LFU policy in that a reference count is maintained for each file in the cache. The LFU* policy differs from the LFU policy in that on a cache miss, the newly requested document is not always added to the cache. More specifically, only files with a reference count of one in the cache are candidates for replacement. In other words, the LFU* policy is really a "restricted" version of LFU that only allows files with a current reference count of one[2] to be removed from the cache. A newly referenced document enters the cache only if there is enough replaceable space for it. Otherwise, the document does *not* enter the cache, and no replaceable documents are removed.

Like LFU, LFU* can also suffer from files building up high reference counts. In fact, once all files in the cache have reference counts of two or more, no new files are ever added. This is not a problem if the cached documents remain popular, but would be disastrous if the set of popular documents changed. To alleviate this problem, the aging policy that is used by the traditional LFU policy was employed, so that the reference counts, over time, can be reduced back to one, making more documents candidates for replacement. This second new policy is denoted as LFU*-Aging. For the purpose of comparison, both LFU-Aging and LFU*-Aging are tested with the same aging parameters [3].

3.1.3 Threshold Policy

The third factor used in this study is a threshold policy that places constraints on the sizes of documents allowed in the cache. A requested document is added to the file cache if and only if the size of the requested document is within the acceptable range. One characteristic of the data sets under study is that most client requests are for small documents, while a significant portion of the data volume was caused by requests for larger documents [2]. Therefore, it might be possible to improve the cache hit rate by caching a large number of small documents (i.e., put an upper limit on the size of document that may be added to the cache), or to reduce the data volume by caching only larger documents (i.e., place a lower limit on the size of document that may be added to the cache).

In total, four levels are examined for this factor. The first level caches all files (no threshold policy). The second level places an upper limit on the size of documents that may be added to the cache, while the third level uses a lower limit on document size. The fourth level used for this factor implements both an upper and a lower bound on the size of file that may be added to the cache.

3.1.4 Partitioning Policy

A partitioning policy is the fourth factor in this study. This policy divides the file cache into a number of partitions based on document types. This approach is considered to determine if preventing files of one type from replacing another type in the cache has any effect on performance. This approach was considered as most requests ($\geq 90\%$) are for either HTML or image documents [2, 6, 12, 13].

Two levels are tested in this study. The first level uses only one partition (the cache is shared by all document types). The second level uses three partitions: one for HTML documents, one for image documents and one for all other document types.

3.2 Performance Metrics

Two performance metrics are used to evaluate each file cache configuration: cache hit rate, and the data volume. The cache hit rate is the fraction of requests to the server that are satisfied by searching the file cache for a copy of the requested document. The data volume refers to the fraction of the requested

[2] Over time, aging can make more and more documents fit into this "replaceable" category.

Table 3: Length of Warmup Period Chosen (All Data Sets)

Item	Waterloo	Calgary	Saskatchewan	NASA	ClarkNet	NCSA
Requests	50,000	250,000	500,000	1,000,000	900,000	900,000
% of Trace	30.7	44.0	23.1	32.4	30.6	39.3

bytes that must be transferred between the Web server and secondary storage, relative to the number of bytes transferred when no cache is used. The cache configuration that achieves the highest cache hit rate and the lowest data volume is considered to be the file cache with the best overall performance.

A third possible metric, response time (i.e., latency), is not included in this study, as the data sets do not contain the necessary information. However, the Web server that achieves the highest hit rate and the lowest data volume should also have the shortest response time (assuming that communication with secondary storage, and not the server's CPU, is the performance bottleneck).

3.3 Other Issues

Two remaining issues deserve mention: simulation validation and simulation warmup period. Our simulation was initially validated using very short (e.g., up to 100 requests) Web server workload traces, for which correct cache statistics could be computed by hand. Small cache sizes were also used so that cache replacement decisions could be exercised. More recently, we have also validated our simulator against that[3] used by Williams *et al.* [20]. Note that calculating (100% - Data Volume) gives the Weighted Hit Rate metric, as described in [20].

One final issue is the length of the simulation warmup period, used to reach steady state. Only the steady-state performance is of interest, since during the initial (or transient) state, many cache misses (so called "cold misses") occur, simply because the cache is empty. However, identifying where the transient state ends and the steady-state begins is a difficult problem [11].

For our workload traces, steady state was reached, on average, after 33% of the requests in the logs. All remaining experiments thus use the warmup periods[4] indicated in Table 3. During the warmup period, the file caches behave normally, with files being added and removed as needed, according to the replacement, threshold and partitioning policies that are in place at the cache. However, statistics concerning the performance of the file cache are not collected until the cache has entered into the (predetermined) steady-state.

3.4 Evaluation Technique

In order to observe the full range of interactions between the different factors, a full-factorial design is required. However, due to the large number of factors and levels being studied, this can result in a very time and resource consuming process. Thus, to reduce the overall size of the study, several levels for some of the factors are eliminated at an early point in the study.

The main factors affected by the decisions were aging parameters, cache replacement policies and cache threshold and partitioning parameters. During the initial stages of the study, values for the aging parameters were tested for each trace. The values determined during these tests were then used for the remainder of the experiments.[5] During the comparison of the replacement policies, it was clear that the

[3]To be precise, we have validated our results against the corrected (August 1996) version of their (initially incorrect) simulator.

[4]See [3] for information on how these values were obtained.

[5]This process is described in [3].

LFU*-Aging policy always outperformed the LFU-Aging policy, so for all of the remaining experiments, LFU-Aging was excluded. During the evaluation of the threshold and partitioning policies, only a small number of values were tested for each level. While these tests may not uncover the full potential of these policies, these few values are sufficient to identify some of the benefits and drawbacks of these policies. Throughout all of the experiments, all 12 different cache sizes were simulated.

3.5 Summary

This section has presented the experimental design for the performance study of a file cache on a Web server. This study focuses on the effects of cache size, replacement policy, threshold policy and partitioning policy on the performance of a file cache. The next section presents the results from our performance study.

4 Simulation Results

This section presents the simulation results from the file cache performance study. The results are presented in five steps. Section 4.1 presents the simulation results for frequency-based replacement policies (only), both with and without aging policies. Section 4.2 presents the performance results for the size-based replacement policies (only). Section 4.3 then presents results comparing frequency-based, size-based, and recency-based policies. Section 4.4 describes the performance results of a file cache that uses a size threshold policy. Section 4.5 discusses the results obtained when a partitioning policy is used. Finally, Section 4.6 summarizes the results of the performance study.

4.1 Comparison of Frequency-Based Policies

In Section 3.1.2 a new replacement algorithm, LFU*, was described. In this section, the performance of LFU* and LFU*-Aging (the LFU* policy with document aging) is compared to the performance of the traditional LFU and LFU-Aging policies.

Figure 1 shows the performance of the LFU, LFU-Aging, LFU*, and LFU*-Aging replacement policies, for the Saskatchewan data set. The general trends that were seen across all six data sets can be seen in this figure [3]. In terms of cache hit rate, the LFU*-Aging policy performs the best, while in most cases, LFU* performs the worst. The LFU and LFU-Aging policies perform similarly, usually better than LFU* but not as well as LFU*-Aging.

The poor performance of LFU* can be attributed to the fact that once it fills up with a set of popular documents, those documents are never removed. The LFU* cache performs well as long as this set of documents remains popular. As the popularity of the cached documents diminishes, so too does the performance of the LFU* cache. This effect is most pronounced in workloads with long time durations [3]. By using the aging policy, LFU*-Aging is able to make some changes to its (cached) document set, to avoid this performance degradation. At the same time, the LFU*-Aging policy appears to avoid some of the performance impact of the "one-timers", which LFU and LFU-Aging suffer from. Since LFU* performs so poorly, it is excluded from all remaining experiments.

It is interesting to note that initially adding the "replaceability" restriction onto LFU makes LFU* significantly worse than LFU, but then adding aging to LFU* then makes LFU*-Aging *better* than LFU-Aging, even though adding aging alone to LFU has minimal effect. In other words, it is precisely the *combination* of these two modifications to LFU, motivated in part by our workload observations in Section 1, that seems to provide better protection against one-timers, improving cache hit rate performance.

In terms of reducing data volume (i.e., the bottom graph in Figure 1), the LFU-Aging and LFU*-Aging replacement policies perform about the same [3]. In the range of interest (e.g., 8 MB to 96 MB), the data volume is reduced by at least an order of magnitude.

4.2 Comparison of Size-Based Policies

Recent work at Virginia Tech has found that size-based replacement policies outperform traditional policies like LRU and LFU for World-Wide Web proxy caches [1, 20]. The size-based policies attempt to remove large files from the file cache, in order to keep a large number of smaller files in the cache. In the case of a tie, a secondary algorithm, such as LRU, may be used.

For this study, six size-based policies were evaluated. On a cache miss, when there is insufficient space to add the new file, the initial replacement decision is made based either on the Size or $\lfloor log_2(Size) \rfloor$ of the incoming file, relative to that of files already in the cache. All ties are broken using one of the following policies: LRU, LFU, or Random. From this design, we have the following six replacement policies: Size-LRU (i.e., the initial choice is based on the size of the incoming file, and ties are broken in an LRU fashion), Size-LFU, Size-Random, Log-LRU, Log-LFU, Log-Random.

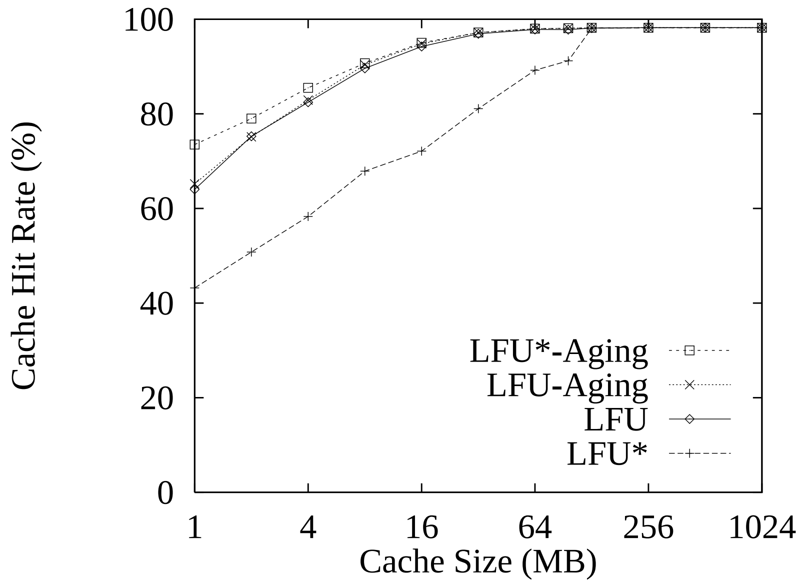
Figure 2 shows the results of the size-based replacement policy experiments for the NCSA data set. As can be seen, there is little performance difference between the size-based policies. Across all six data sets, the performance results are quite similar to the results for the NCSA data: the $\lfloor log_2(Size) \rfloor$ policies always slightly outperformed the Size policies (both in cache hit rate and data volume), while Log-LFU always outperformed Log-LRU, which in turn always outperformed Log-Random. Across all six data sets, the three Size policies performed almost identically. This occurs because there are far fewer ties than with the $\lfloor log_2(Size) \rfloor$ size approach.

4.3 Comparison of All Cache Replacement Policies

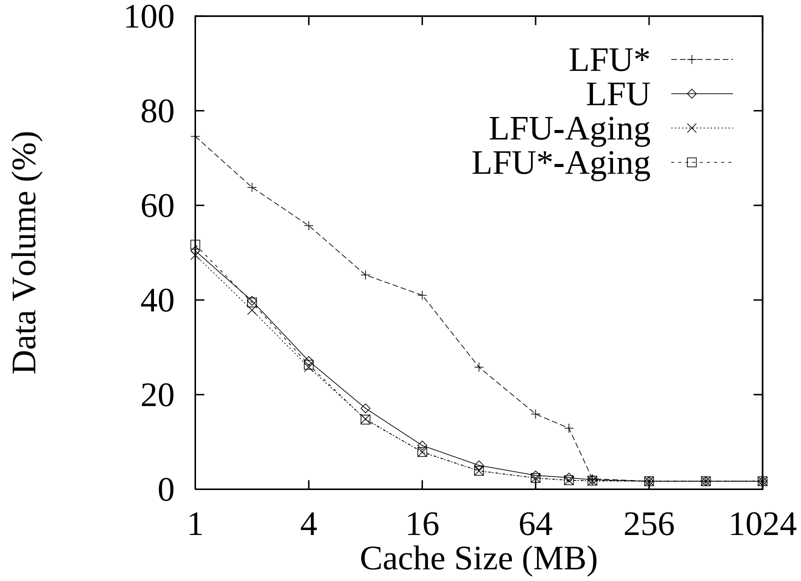
The next experiment was to compare the performance of all the cache replacement policies (i.e., frequency-based, size-based, and recency-based). The replacement policy that achieves the highest cache hit rate and the lowest data volume is considered to be the best policy. In order to make the graphs more legible, a number of the replacement algorithms are not considered in this experiment. LFU*, which always performs worse than LFU*-Aging, has been removed. Since the Log-LFU policy always performed the best among the six size-based policies studied, it alone is used to represent the size-based policies. Although LFU-Aging did not perform as well as LFU*-Aging (see Section 4.1), it is retained in this section so that the performance of the standard LFU policy (with aging) can be compared with the LRU, FBR, and Size policies.

Figure 3 shows the results of the replacement policy experiments using the NCSA data set. As is to be expected, the cache hit rate increases for all policies as the cache size increases, and the data volume correspondingly decreases.

Performance differences between the policies are most evident on small caches (when good replacement decisions are important) and negligible on large cache sizes (when very few replacements occur). The relative order is quite consistent across the range of cache sizes considered. In general, the modified LFU algorithm with aging (LFU*-Aging) provides the best performance, in terms of both cache hit rate and data volume. The size-based policy achieves a very high hit rate, but performs very poorly in terms of data volume (in all the data sets, the size policy had the highest data volume). These results are somewhat intuitive, as the policy attempts to keep a lot of small files in the cache (which should improve the hit rate) by removing the larger files first (however, the larger files are responsible for a significant portion of the data volume). LRU, traditionally one of the "best" caching policies in computer systems

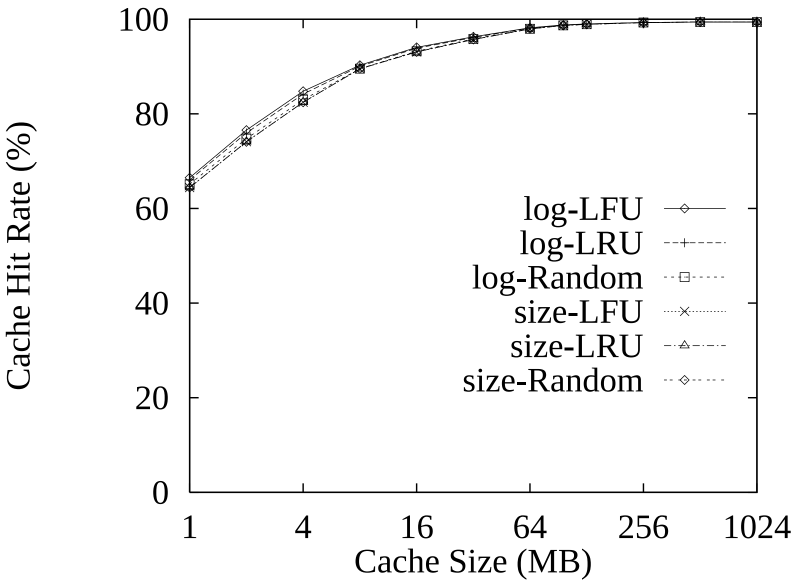


(a)

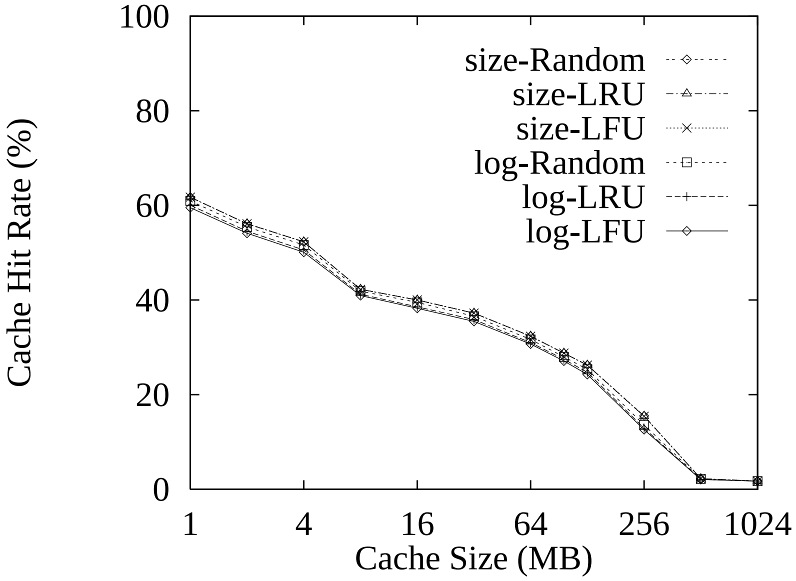


(b)

Figure 1: Comparison of LFU, LFU-Aging, LFU*, and LFU*-Aging Replacement Policies: (a) Cache Hit Rate (b) Data Volume (Saskatchewan Data)



(a)



(b)

Figure 2: Comparison of Size-Based Replacement Policies: (a) Cache Hit Rate (b) Data Volume (NCSA Data)

performance studies, performs quite poorly. Across all data sets, LRU achieved the lowest cache hit rate, and was next to last in reducing the data volume. The FBR policy, which combines the notions of LRU (in the New Section) and LFU (in the Old Section) generally provides performance between that of LFU-Aging and LRU (better than LRU, but not as good as LFU-Aging or LFU*-Aging). These results indicate that the extra implementation overhead of FBR caching is not likely to be worthwhile for Web server caching. LFU*-Aging is a simpler policy to implement, and consistently provides better performance.

4.4 Effects of a Threshold Policy

The next set of experiments studied the effects of a threshold policy on the performance of a file cache. The threshold policy places a constraint on the size of documents allowed into the cache.

Four different levels were tested for the threshold policy. The levels are: (i) no restrictions on document size (i.e., no threshold policy; the results for this level are the same as those in Section 4.3); (ii) an upper limit (100,000 bytes) on document size; (iii) a lower limit (500 bytes) on document size; (iv) both an upper and a lower limit on file size (500 - 100,000 bytes).

The results for the threshold experiments are shown in Figure 4 for the NCSA data set, for a file cache using the LFU*-Aging replacement policy (results for the other data sets, as well as for the LRU and FBR replacement policies, are available in [3]). The key in each graph refers to the range of document sizes permitted in the cache for the given threshold policy.

For the upper threshold experiment, the limit on document size was set to 100,000 bytes. On average, 95.4% of documents on the six servers were less than this size. The results for this threshold policy are labelled 0-100K on each of the graphs. Across all six data sets, the use of the "upper limit only" policy resulted in an improvement in cache hit rate for small cache sizes, compared to a similar cache with the same replacement policy (see Figure 4(a)). Similar findings are reported in [15]. In all cases, using an upper limit threshold policy results in an increase in the data volume (see Figure 4(b)). This increase occurs because files larger than the maximum cacheable size are responsible for a significant portion of the data volume.

A single value (500 bytes) is tested as a lower limit for the threshold policy. Across the six servers, approximately 14.8% of the files were smaller than 500 bytes in size. The results for these experiments are shown in Figure 4 as well, and are labelled 500-INF on each of the graphs. As was expected, the cache hit rate worsened for all data sets, since many small, frequently requested files are now excluded from the cache. The intent of this threshold policy is to cache only large files, which are responsible for more of the data traffic. However, the results show that a decrease in the data volume is not achieved. As with the upper limit threshold policy, using a lower limit prevents the cache from reaching the maximum achievable hit rate or the minimum achievable data volume.

The last set of experiments involving the threshold policy places both an upper and a lower limit on the size of files that may be added to the cache. These experiments used an upper limit of 100,000 bytes and a lower limit of 500 bytes. The results are labelled as 500-100K on Figure 4. This policy achieves a slightly better hit rate than using only a lower limit, but does not achieve as high a hit rate as a policy that implements only an upper limit. However, using both an upper and a lower restriction on file size results in a significant increase in the data volume, which is clearly undesirable. Once again the threshold policy is unable to reach either the maximum achievable hit rate or the minimum achievable data volume, no matter how large the cache is.

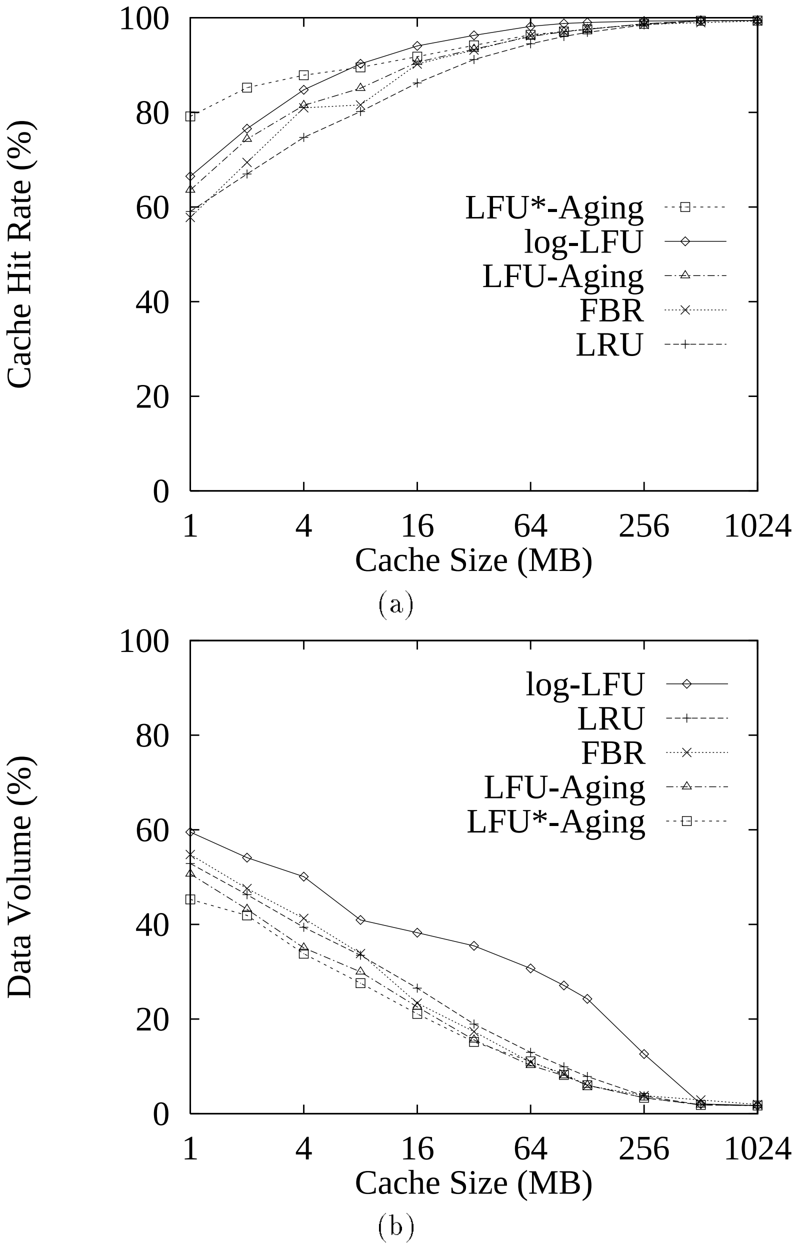


Figure 3: Comparison of Replacement Policies: (a) Cache Hit Rate (b) Data Volume (NCSA Data)

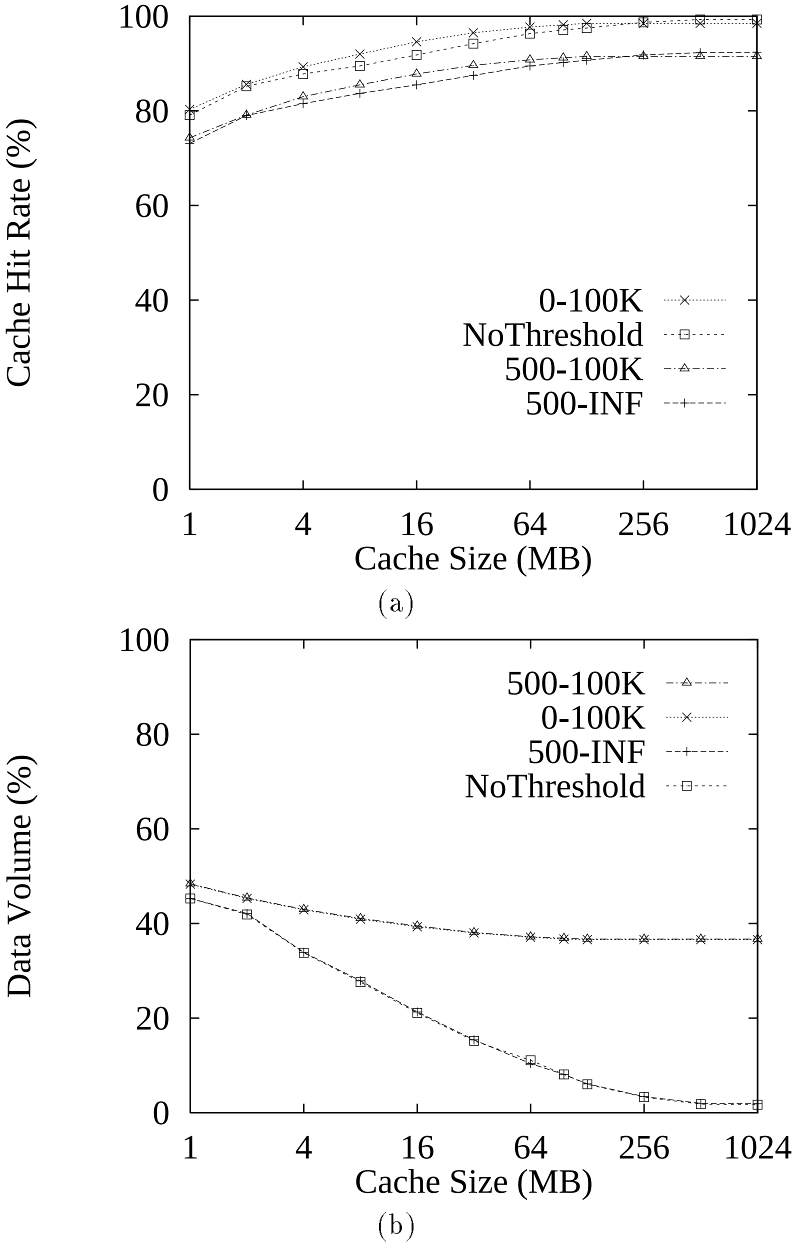


Figure 4: Comparison of Document Size Threshold Policies (a) Cache Hit Rate (b) Data Volume (NCSA Data)

4.5 Effects of a Partitioning Policy

The final experiment attempts to determine if a partitioning approach to file caching could improve the performance of a file cache, by protecting popular documents from being replaced by infrequently requested documents (of a different type).

The control case in this experiment is when no partitioning is used (i.e., the cache space can be shared by all file types). The results for this approach are the same as for those discussed in Section 4.3.

The second case under study uses three partitions: one for HTML documents only, one for Image documents only, and the third for any other document types that are requested. Figure 5 shows the results for the partitioning policy experiment using the NCSA data set and the LFU*-Aging replacement policy. For this experiment, the HTML partition was allocated 51% of the cache space, the Image partition 48%, and the remaining 1% to the Other partition. These partition sizes were chosen to match the request frequencies for each document type in the NCSA data set.

Figure 5(a) shows that by using this configuration, the partitioning policy attained approximately the same hit rate as the no-partitioning approach. However, the partitioned cache increases the data volume (see Figure 5(b)). Similar results were found in the other data sets [3].

In general, the partitioning approach was found to provide some small gains in cache hit rate, but usually increased the data volume. The increase in data volume occurs because a lot of the large files are not cached, because they are too large to fit into the appropriate partition. The main problem with the partitioning approach is the difficulty in selecting the number of partitions, and the sizes of those partitions; most choices that were tested resulted in huge decreases in performance [3]. Due to these problems, a cache that uses statically defined partitioning does not appear to be beneficial. An adaptive approach, similar to the adaptive threshold policy designed by Markatos [15], may have better results.

4.6 Summary of Results

This simulation study found that for a Web server, a file cache using the LFU*-Aging replacement policy performs better (i.e., has a higher cache hit rate and a lower data volume) than file caches using LFU, LRU, or FBR replacement policies. Size-based policies were found to perform about the same as LFU*-Aging in terms of cache hit rate, but had a much higher data volume. This study also found that a threshold policy that places an upper limit on document size can improve the hit rate of a file cache, but increases the data volume. Finally, this study found that in some cases it is possible to improve file cache performance with the use of a partitioning policy, but that it is difficult to determine the partition sizes to use. If the partitions are not set correctly, the performance of the file cache is negatively affected.

5 Summary and Conclusions

This paper has presented a performance study of a file cache for a Web server. Trace-driven simulations were used to study the effects of the replacement policy used by the file cache, as well as the effects of a threshold policy (that restricts which documents can be added to the cache based on the size of the document) and a partitioning policy (that maintains separate partitions within the cache for files of a particular type).

The simulation results show that the choice of replacement policy has the greatest impact on the performance of the file cache. Across the six data sets studied, the LFU*-Aging policy achieved the best performance using the metrics of cache hit rate and data volume. Size-based policies also achieved high hit rates, but did not reduce the data volume as effectively as the other policies studied. The results also

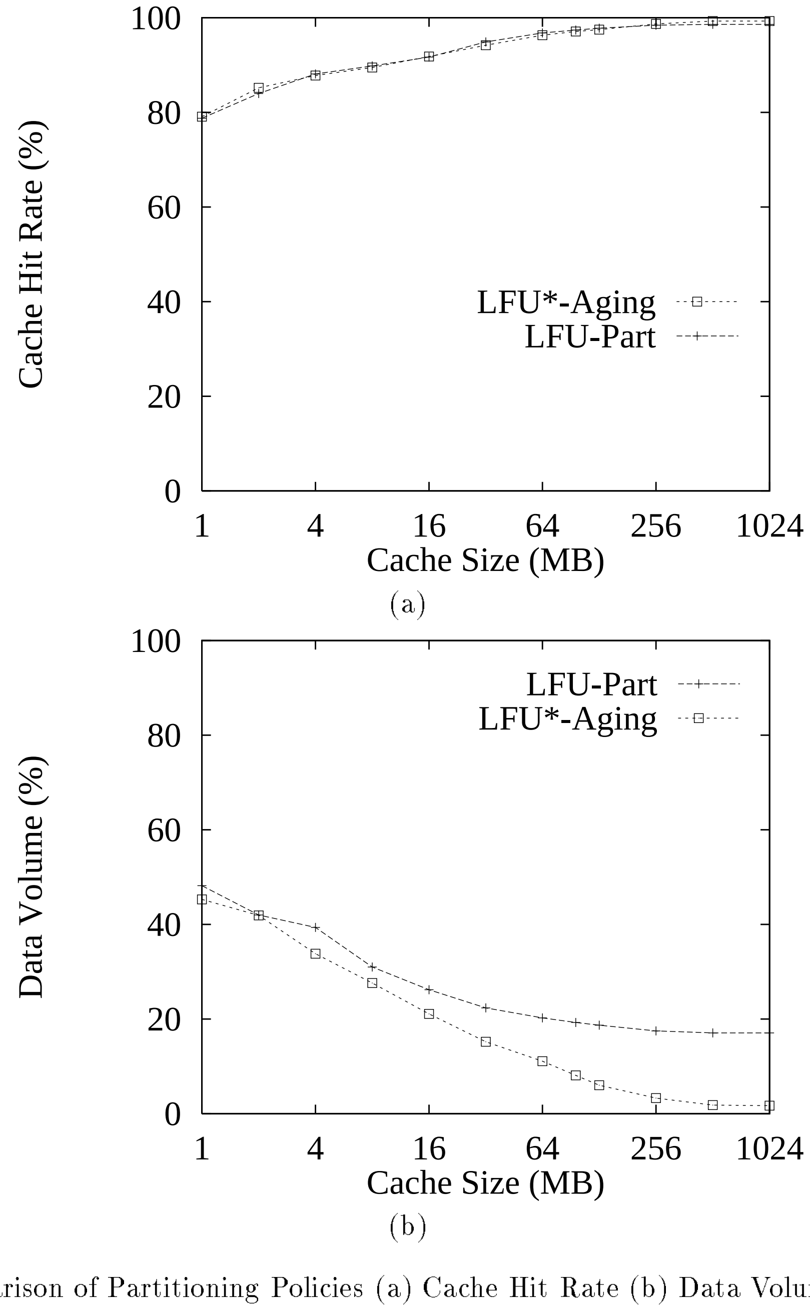


Figure 5: Comparison of Partitioning Policies (a) Cache Hit Rate (b) Data Volume (NCSA Data)

show that a threshold policy that implements an upper limit on file size can have some positive effects on file cache performance, while the partitioning policy does not appear to be as useful.

The results from this study show that a properly designed file cache can reduce data volume by at least an order of magnitude. The decision that has the largest performance impact is the choice of replacement policy. Recency-based replacement policies (e.g., LRU), which have often been the replacement policies of choice in most computer systems performance studies, are not the best choice for a file cache on a Web server. The success of simple frequency-based policies such as LFU-Aging and LFU*-Aging suggest that more complex policies such as FBR should not be chosen. Workload differences between a proxy server and a single Web server may explain why the size-based policies do not perform as well as the LFU*-Aging policy for Web servers. While some of the ideas considered (thresholding and partitioning) can improve file cache performance, this improvement is often minimal. Furthermore, these enhancements are difficult to implement, and can have a negative impact on performance.

The results also show that Web server performance could be increased by adding a file cache within the address space of the Web server, rather than relying on the caching mechanisms of the file system (which tends to use block caching and LRU replacement). A file cache can further increase the performance of the Web server by reducing interaction with the operating system.

Future work in this area will include implementing a file cache in a Web server, and addressing the practical matters of cache consistency, cache security, and dynamic files. Some companies (e.g., Spyglass) have already realized the performance potential of file caching, and have added it to their Web server software. A study of new policies may find further performance improvements. Multilevel caching might also be explored, to examine the benefits of a local disk cache for a Web server that already uses a memory file cache.

References

[1] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. Fox, "Caching Proxies: Limitations and Potentials", *Electronic Proceedings of the Forth World Wide Web Conference '95: The Web Revolution*, Boston, MA, December 11-14, 1995.

[2] M. Arlitt and C. Williamson, "Web Server Workload Characterization: The Search for Invariants", *Proceedings of the 1996 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Philadelphia, PA, pp. 126-137, May 23-26, 1996.

[3] M. Arlitt, "A Performance Study of Internet Web Servers", M.Sc. Thesis, Department of Computer Science, University of Saskatchewan, June 1996.

[4] A. Bestavros, R. Carter, M. Crovella, C. Cunha, A. Heddaya and S. Mirdad, "Application-Level Document Caching in the Internet", *Proceedings of the Second International Workshop on Services in Distributed and Networked Environments (SDNE '95)*, Whistler, BC, June 1995.

[5] C. Bowman, P. Danzig, D. Hardy, U. Manber, M. Schwartz and D. Wessels, "Harvest: A Scalable, Customizable Discovery and Access System", Technical Report CU-CS-732-95, University of Colorado, Department of Computer Science, Boulder, Colorado, March 1995.

[6] H. Braun and K. Claffy, "Web Traffic Characterization: An Assessment of the Impact of Caching Documents from NCSA's Web Server", *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, Chicago, IL, October 1994.

[7] P. Danzig, M. Schwartz and R. Hall, "A Case for Caching File Objects Inside Internetworks", *Proceedings of ACM SIGCOMM '93*, San Francisco, CA, pp. 239-248, September 1993.

[8] K. Froese and R. Bunt, "The Effect of Client Caching on File Server Workloads", *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, Wailea, HI, January 1996.

[9] S. Glassman, "A Caching Relay for the World Wide Web", *First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

[10] J. Gwertzman, "Autonomous Replication in Wide-Area Internetworks", B.A. Thesis, Center for Research in Computing Technology, Harvard University, Cambridge MA, April 1995.

[11] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, John Wiley & Sons, Inc., New York, NY, 1991.

[12] T. Kwan, R. McGrath, and D. Reed, "NCSA's World Wide Web Server: Design and Performance", *IEEE Computer*, Vol. 28, No. 11, pp. 68-74, November 1995.

[13] T. Kwan, R. McGrath, and D. Reed, "User Access Patterns to NCSA's Web Server", Technical Report UIUCDCS-R-95-1934, Department of Computer Science, University of Illinois, Urbana-Champaign, February 1995.

[14] A. Luotenen and K. Altis, "World-Wide Web Proxies", *First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

[15] E. Markatos, "Main Memory Caching of Web Documents", *Electronic Proceedings of the Fifth World Wide Web Conference '96*, Paris, France, May 6-10, 1996.

[16] National Center for Supercomputing Applications, "KeepAlive Directive", 1996.

[17] V. Padmanabhan and J. Mogul, "Improving HTTP Latency", *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, Chicago, IL, October 1994.

[18] J. Robinson and M. Devarakonda, "Data Cache Management Using Frequency-Based Replacement", *Proceedings of the 1990 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Boulder, CO, pp. 134-142, May 1990.

[19] S. Spero, "Progress on HTTP-NG", March 1995.

[20] S. Williams, M. Abrams, C. Standridge, G. Abdulla and E. Fox, "Removal Policies in Network Caches for World-Wide Web Documents", to appear in the *Proceedings of ACM SIGCOMM '96*, Stanford, CA, August 1996.

[21] D. Willick, D. Eager and R. Bunt, "Disk Cache Replacement Policies for Network Fileservers", *Proceedings of the 13th International Conference on Distributed Computing Systems*, Pittsburgh, PA, pp. 2-11, May 25-28, 1993.

[22] N. Yeager and R. McGrath, *Web Server Technology: The Advanced Guide for World Wide Web Information Providers*, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.