Removal Policies in Network Caches for World-Wide Web Documents

Stephen Williams, Marc Abrams (Computer Science, Virginia Tech, Blacksburg, VA 24061-0106)

Charles R. Standridge (Industrial Eng., FAMU-FSU College of Eng., Tallahassee, FL 32310)

Ghaleb Abdulla, Edward A. Fox (Computer Science, Virginia Tech, Blacksburg, VA 24061-0106)

{williams,abrams}@cs.vt.edu, stand@eng.fsu.edu, {abdulla,fox}@cs.vt.edu

Abstract

World-Wide Web proxy servers that cache documents can potentially reduce three quantities: the number of requests that reach popular servers, the volume of network traffic resulting from document requests, and the latency that an end-user experiences in retrieving a document. This paper examines the first two using the measures of cache hit rate and weighted hit rate (or fraction of client-requested bytes returned by the proxy). A client request for an uncached document may cause the removal of one or more cached documents. Variable document sizes and types allow a rich variety of policies to select a document for removal, in contrast to policies for CPU caches or demand paging, that manage homogeneous objects. We present a taxonomy of removal policies. Through trace-driven simulation, we determine the maximum possible hit rate and weighted hit rate that a cache could ever achieve, and the removal policy that maximizes hit rate and weighted hit rate. The experiments use five traces of 37 to 185 days of client URL requests. Surprisingly, the criteria used by several proxy-server removal policies (LRU, Hyper-G, and a proposal by Pitkow and Recker) are among the worst performing criteria in our simulation; instead, replacing documents based on size maximizes hit rate in each of the studied workloads.

1 Introduction

It is ironic that what is arguably the most popular application of the Internet, namely the World-Wide Web (WWW or "Web"), is inherently unscalable. The protocol underlying the Web, HTTP [7], is designed for a setting in which users access a large body of documents, each with a low volume of readership. However the Web today is characterized by high volume of access to popular Web pages. Thus identical copies of many documents pass through the same network links. This has several costs: network administrators see growing utilization that requires bandwidth upgrades, Web site administrators see growing server utilization that requires upgrading or replacing servers, and end users see longer latency for document requests to be satisfied.

These problems can be alleviated by widespread migration of copies of popular documents from servers to points closer to users. Migration can follow a distribution model, in which servers control where document copies are stored, or a cache model, in which copies automatically migrate in response to user requests. Distribution is popular with commercial users that want to protect material with copyrights, and will only trust certain sites to keep copies of such material. However the cache model is the most popular method to date: caching at a server (e.g., [10]), caching at a client (e.g., caches built into Web browsers), and caching in the network itself through so-called proxy servers (e.g., [11]). This paper examines the last of these.

In the following, a *document* is any item retrieved by a Universal Resource Locator (URL), such as a dynamically created page or an audio file. We call a document on a Web server an *original*, and a document on a cache a *copy*. Consider a client configured to use proxy P that requests document D from Web server S. The client sends an HTTP GET request message for URL *http://S/D* to P. Three cases are now possible: (1) P has a copy of D that it estimates is consistent with the original, (2) P has a copy that is considered inconsistent, and (3) P does not have a copy. (Various algorithms not considered here are used to estimate consistency.) In case (1), P serves the local copy; this is a *hit*. In (2), P sends an HTTP conditional GET message to S containing the `Last-Modified` time of its copy; if the original was modified after that time, S replies to P with the new version (a *miss*). Otherwise, P serves the local copy; this is also a *hit*. In (3), P either forwards the GET message to another proxy server (as in [12]) or to S (also a *miss*).

Proxy caches can dramatically reduce network load. For example, in one of the five Web traffic workload traces used in this paper, representing remote client requests for documents on a network backbone in the Computer Science Department at Virginia Tech, 88% of the bytes transferred in a 37 day measurement period were audio. This surprising result arose because a student created what has been recognized in the WWW as the authoritative Web site for a popular British recording artist. Unfortunately, every request worldwide to listen to a song on this Web site consumes network resources from the client through the Internet to the university, then over the campus backbone, and finally through the department backbone to the server. In contrast, a campus-wide cache located at the campus's connection to the Internet would eliminate up to 89.2% of the bytes sent in HTTP traffic in the department backbone.

However, proxy caches are not a panacea. First, caching only works with staticly created and infrequently changing documents. Dynamically created documents, which are becoming more popular with commercial content providers, currently, cannot be cached. Second, HTTP 1.0 does not provide a reliable method to identify whether a Web document is cacheable (e.g., whether or not it is script generated). Third, there is no accepted method in the Web of keeping cached copies consistent. Finally, copyright laws could legislate caching proxies out of existence, without proper accounting methods. Some possible approaches to caching dynamic documents are listed in the Conclusions (§5).

Caching proxies can reduce three different criteria: the number of requests that reach servers, the volume of network traffic resulting from document requests, and the latency that an end-user experiences in retrieving a document. We consider the first two in this paper. Arlitt and Williamson [5] also consider the first two, and observe that "optimizing one criterion does not necessarily optimize the other. The choice between the two depends on which resource is the bottleneck: CPU cycles at the server, or network bandwidth."

The performance measure we use for the criteria of number of requests reaching a server is *hit rate* (HR), or the number of requests satisfied by the cache. For the criteria of network volume, several measures are possible. With proxy caching, an equal number of bytes will be sent to the client no matter how caching is done; the only question is how far those bytes have to travel (e.g., from the proxy or from the server). Possible measures are the number of bytes not sent by the server, the number of packets not sent by the server, and the reduction in the distance that the packets traveled (e.g., hop count). In this paper we consider the first measure — the number of bytes not sent — using the complementary measure of the fraction of client-requested bytes returned by the proxy, *weighted hit rate* (WHR). The workload traces we use have insufficient information to determine packets not sent or reduction in distance. Finally, for the criteria of end-user latency, a measure such as transfer time avoided is appropriate. However our traces have insufficient information on timing of requests and responses to determine transfer time. We can only say that if HR and WHR are high, and the proxy is not saturated by requests, than the user will experience a reduction in latency.

We study cache performance by trace-driven simulation of a proxy cache. Considered in the first experiment is the maximum values of HR and WHR for a given workload, which is obtained by simulating an infinite size cache. This represents the inherent "concentration" (as defined in [5]) in URL references by clients, arising when multiple clients request the same URL, or "temporal locality" by a single client that requests the same URL multiple times.

Real caches have finite size, and thus a request for a non-cached document may cause the removal of one or more cached documents. The second experiment considers the effect of removal policies on HR, WHR, and cache size. (We use the term "removal" policy rather than "replacement" because a policy might be run periodically to remove documents until free space reaches a threshold [see §1.3]. Strictly speaking, the policy is just removing cached documents.)

Unlike CPU caches or virtual memory, which cache objects of identical size, documents in a proxy cache may have widely varying sizes — from text files of a few bytes to videos of many megabytes. Also document *type*, such as audio or image, may be considered by a removal policy, in contrast to CPU caches, which treat all data as homogeneous. On the other hand, proxy caches are simpler in that there are never "dirty" documents to write back.

We could directly compare removal policies by simulating various removal policies, such as the first-in, first-out (FIFO) [16]; least recently used (LRU) [16]; LRU-MIN [1], a variant of LRU that considers document size; least frequently used (LFU) [16]; the Hyper-G [3] server's policy; and the policy proposed in [13], denoted Pitkow/Recker. But the policies are ad hoc in the sense that little consensus exists among the policies on what factors to consider in deciding which document to remove, or their relative importance: document size, time the document entered the cache, time of last access, number of references, and so on.

Rather than directly compare such policies, the methodology in this paper is to develop a taxonomy of removal policies that views them as sorting problems that vary in the sorting key used. Policies in the literature represent just a few of the many possibilities, so our study yields insight into policies that no one has yet proposed.

Another issue explored is the effectiveness of second level caches. Widespread deployment of caching proxies within a large organization will lead to some caches handling misses from other caches. In experiment three we find the theoretical maximum HR and WHR of a second level cache.

The fourth and final experiment considers a question raised in [10]: Video and audio files often represent a small fraction of requests in a workload, but due to their large file sizes may represent a majority of bytes transferred. Thus large video and audio files could displace many smaller documents of other types in a cache. Should a cache be partitioned by media type? The fourth experiment considers this question for the workload described earlier with a high volume of audio bytes transferred.

1.1 Definition of a Hit in Simulation

Earlier we defined a hit as a URL reference that goes to a cache that has a copy of the requested document that is consistent with the Web server named in the URL. Because three of the five studied workloads are derived from common-format proxy log files, there is no data on file modification times. Instead, the simulation estimates whether a cached copy is consistent in the following manner.

If a URL appears twice in a trace file, but with different sizes, then the named document *must* have changed between the two references, and hence the cached document copy is inconsistent with the Web server. Therefore during simulation, a cache *hit* is a match in *both* URL and size. The case of the size of a given URL changing occurs infrequently in our traces – the fraction of non-dynamically generated URLs in each trace used here that occurred earlier in the trace but with a different size ranges from 0.5% to 4.1%.

However, a document could be modified between successive occurrences of a URL in a trace, even though its size remains the same. Examples of this include a minor update to a text document that does not change its length, and a replacement inline GIF image in a document whose characteristics are identical (e.g., dimension, color frequency, color depth, etc). In such cases our simulation will mistakenly regard the cached copy as consistent. The mistake will not occur often, because all but the most trivial changes to text files and almost any change to a non-text file that is compressed will change the document length. To assess the likelihood of the mistake, we examined two workloads (denoted later as BR and BL) for which the trace contained the HTTP `Last-Modified` field and found that the file was modified in only 1.3% of references in which the file size remained the same. Thus our simulation would incorrectly calculate a hit when a real cache would miss in 1.3% of the references for two of the five workloads used here.

1.2 Removal Algorithms Studied

As stated above, our methodology for comparing removal policies is a taxonomy defined in terms of a *sorting procedure*.

A removal policy is viewed as having two phases. First, it sorts documents in the cache according to one or more *keys* (e.g., primary key, secondary key, etc.). Then it removes zero or more documents from the head of the sorted list until a *criteria* is satisfied. In this paper, the *criteria* is that the amount of free cache space equals or exceeds the incoming document size. (Alternatives are considered in §1.3.) The keys studied here are listed in Table 1.

Consider a cache of size 42.5kB. Table 2 exemplifies removal policies viewed as sorting algorithms. The upper table contains a sample trace for URLs, denoted $A-H$. After time 15, the cache is 100% full. Suppose a previously unseen URL with size 1.5kB, denoted I, is referenced just after time 15. Which document(s) will be removed to accommodate I?

The middle table lists the values of each key from Table 1 just after time 15. The lower table shows the sorted list that would result from several combinations of primary and secondary keys. Multiple files listed between two vertical lines in 2 indicates instances where the primary key value is equal; the numbers are then ordered by the secondary key. Asterisks denote which documents are removed. For example, LRU (equivalent to primary key of ATIME) will first remove document B, freeing up 1.2kB of cache space, but this is insufficient for document I of size 1.5kB. LRU then removes E to free 8kB more, and I can now be loaded.

Certain combinations of the sorting keys in Table 1 correspond to removal policies from the literature, as shown in Table 3. FIFO is equivalent to sorting documents by increasing values of the time each document entered the cache (e.g., ETIME), so that the removal algorithm removes the document with the smallest cache entry time (e.g., the first of the set to enter the cache). LRU is equivalent to sorting by last access time (ATIME). LFU is equivalent to sorting by number of references (NREF). The Hyper-G policy starts by using LFU (i.e., by using NREFS as the primary key), then breaks ties by using LRU (i.e., last access time as the secondary key), and finally size as the tertiary key. (In reality, before considering these three keys, Hyper-G uses a fourth binary key indicating if the document is a Hyper-G document, but our traces contain no Hyper-G documents.)

Two policies cannot be represented exactly in Table 3. First, the Pitkow/Recker algorithm uses a different primary key depending on whether or not all cached documents have been accessed in the current day (e.g., DAY(ATIME) is the current day). But our study will explore both primary keys used by the Pitkow/Recker policy. Also, the Pitkow/Recker policy is also run at the end of the day; this is considered in §1.3. The other policy that cannot be represented exactly is LRU-MIN. LRU-MIN first tests whether there are any documents equal or larger in size than the incoming document; if there is, choose one of them by LRU. Otherwise, consider all documents larger than half the size of the incoming document; if there is, choose one of them by LRU. If not, repeat using one quarter the document size, and so on. LRU-MIN is like using $\lfloor \log_2(\text{SIZE}) \rfloor$ as the primary key (to obtain factors 1/2, 1/4, ...) and ATIME as the secondary key in the sense that large files tend to be removed from the cache first, followed by LRU to select among similar sized files. However, they are not identical because in $\lfloor \log_2(\text{SIZE}) \rfloor$ the factors are not based on the incoming file size; $\lfloor \log_2(\text{SIZE}) \rfloor$ will tend to discard larger files more often than LRU-MIN. Therefore results on the combination of $\lfloor \log_2(\text{SIZE}) \rfloor$ and ATIME later in the paper examine the value of combining size and LRU, not the specific performance of LRU-MIN.

Table 1 contains six sorting keys: SIZE, $\lfloor \log_2(\text{SIZE}) \rfloor$, ETIME, ATIME, DAY(ATIME) and NREF. Our methodology is an experiment design with three factors — primary key, secondary key, and workload — and two response variables — hit rate and weighted hit rate. An equal primary and secondary key is useless. We additionally use random replacement as a secondary key. We always use random as a tertiary key, because we expect that a tie on both the primary and the secondary key is very rare. This gives 36 combinations of primary and secondary keys, and thus 36 policies. By simulating each key combination, we identify the most effective key combinations for our workloads.

1.3 Unexplored Removal Policies

This paper does not explore the implications of two policy issues for removal policies: when to replace and how many documents to remove. The question of when to run the removal policy has been addressed in the following ways in the literature:

On-demand: Run policy when the size of the requested document exceeds the free room in a cache.

Periodically: Run policy every T time units, for some T.

Both on-demand and periodically: Run policy at the end of each day and on-demand (Pitkow/Recker [13]).

The question of how many documents to remove from the cache has been answered in two ways in policies proposed so far. For on-demand, one policy is to stop when the free cache area equals or exceeds the requested document size. For periodic, one policy is to replace documents until a certain threshold (Pitkow and Recker's *comfort level*) is reached.

We did not study these choices because our simulation model is designed to compute document hit rates, but not timings of accesses and removals. However, another reason for not simulating the decisions is that there is no clear advantage of periodic removal.

The argument for using periodic removal is that if a cache is nearly 100% full, then running the removal policy only on-demand will invoke the removal policy on nearly *all* document requests. If removal is time consuming, it might create a significant overhead. This overhead would be reduced by removing periodically until the free space reaches a threshold. However, periodic removal also reduces hit rate (because documents are removed earlier than required and more are removed than is required for the minimal space criteria).

Two arguments suggest that the overhead of simply using on-demand replacement will not be significant. First, the class of removal policies in §1.2 maintains a sorted list. If the list is kept sorted as the proxy operates, then the removal policy merely removes the head of the list for removal, which should be a fast and constant time operation. Second, a proxy server keeps read-only documents. Thus there is no overhead for "writing-back" a document, as there is in a virtual memory system upon removal of a page that was modified since being loaded.

Table 1: Set of sorting keys that underlie the cache removal policies studied.

Key	*Definition*	*Sort Order*
SIZE	size of a cached document (in bytes)	largest file removed first
$\lfloor \log_2(\text{SIZE}) \rfloor$	floor of the log (base 2) of SIZE	one of the largest files removed first
ETIME	time document entered the cache	oldest access removed first (FIFO)
ATIME	time of last document access (recency [13])	least recently used files removed first (LRU)
DAY(ATIME)	day of last document access	files last accessed the most days ago removed first
NREF	number of document references	least referenced files removed first (LFU)

Table 2: Example of removal policy for 42.5kB cache. Top: Sample trace. Middle: Key values at time 15+. Bottom: Resultant sorted lists, with documents (indicated by asterisks) selected for removal to make room for new 1.5kB document.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
URL	*A*	*B*	*C*	*B*	*B*	*A*	*D*	*E*	*C*	*D*	*F*	*G*	*A*	*D*	*H*
Size(kB)	1.9	1.2	9	1.2	1.2	1.9	15	8	9	15	0.3	1.9	1.9	15	5.2

URL	A	B	C	D	E	F	G	H
SIZE	1.9	1.2	9.0	15.0	8.0	0.3	1.9	5.2
$\lfloor \log_2(\text{SIZE}) \rfloor$	10	10	13	13	13	8	10	12
ETIME	1	2	3	7	8	11	12	15
ATIME	13	5	9	14	8	11	12	15
NREF	3	3	2	3	1	1	1	1

Primary Key	*Secondary Key*	*Sorted list of URLs at time 15+*							
SIZE	ATIME	*D**	*C*	*E*	*H*	*G*	*A*	*B*	*F*
$\lfloor \log_2(\text{SIZE}) \rfloor$	ATIME	*E**	*C*	*D*	*H*	*B*	*G*	*A*	*F*
ETIME	n/a	*A**	*B*	*C*	*D*	*E*	*F*	*G*	*H*
ATIME	n/a	*B**	*E**	*C*	*F*	*G*	*A*	*D*	*H*
NREF	ETIME	*E**	*F*	*G*	*H*	*C*	*A*	*B*	*D*

2 Workloads Used in This Study

We can characterize traffic traces used in Web studies in the literature as server-based, client-based, or proxy-based. A server-based study examines logs from individual Web servers [5, 10, 13]. A client-based study examines logs collected by instrumenting a set of Web clients [9]. Server-based traffic usually contains a much higher request rate than client-based traffic, but server-based traffic is more homogeneous because it names a single server. We use a proxy-based study, using a trace of the URL requests that reach a certain point in a network, representing either a proxy log or a tcpdump log. Proxy-based traffic can be similar to server-based traffic if it monitors traffic from clients throughout the Internet to a few servers (e.g., workload BR below) or it can be similar to client-based traffic if it monitors traffic from clients within a network to servers anywhere in the internet (e.g., workloads U, C, G, and BL below).

Five workloads are used in this study, all collected at Virginia Tech. All of the workloads except BR represent the usage patterns of 33 faculty/staff members, 369 undergraduates, and 90 graduate students in the Computer Science Department. They used about 185 Ethernet-connected computers and X-terminals in the department plus additional computers in homes and dormitories, connected by SLIP over modems or Ethernet. There are typically 12 HTTP daemons running within the department. The Web is used to make materials available to students from 20 undergraduate and 10 graduate courses, such as syllabi, course notes, assignments and tests.

Undergrad (U): About 30 workstations in an undergraduate computer science lab for the 185 day period from April to October 1995, containing 188,674 accesses requiring transmission of 2.26GB of static web documents. This workload may be representative of a group of clients working in close confines (i.e., within speaking distance).

Classroom (C): Twenty-six workstations in a classroom on which each student runs a Web browser during four class sessions on multimedia per week, in spring 1995, containing 13,127 accesses requiring transmission of 152.6MB of static web documents. This workload may be representative of clients in an instructional setting, which tend to make requests when asked to do so by an instructor.

Graduate (G): A popular time-shared client used by graduate computer science students, representing at least 25 users, containing 45,400 accesses requiring transmission of 555.2MB of static web pages for most of the spring 1995 semester. This workload may be representative of clients in one department dispersed throughout a building in separate or in common work areas.

Table 3: Removal policies from the literature, defined by equivalent sorting procedures.

	Key 1		*Key 2*		*Key 3*	
Policy Name	*Value*	*Removal Order*	*Value*	*Order*	*Value*	*Order*
FIFO	ETIME	Remove smallest	n/a	n/a	n/a	n/a
LRU	ATIME	Remove smallest	n/a	n/a	n/a	n/a
LFU	NREFS	Remove smallest	n/a	n/a	n/a	n/a
Hyper-G	NREF	Remove smallest	ATIME	smallest	SIZE	largest
Pitkow/Recker	If DAY(ATIME)≠today then Key1=DAY(ATIME), Remove smallest else Key1=SIZE, Remove largest		n/a	n/a	n/a	n/a

Remote Client Backbone Accesses (BR): Every URL request appearing on the Ethernet backbone of domain .cs.vt.edu with a client outside that domain naming a Web server inside that domain for a 37 day period in September and October 1995, representing 227,210 requests requiring transmission of 9.38GB of static Web pages. This workload may be representative of a few servers on one large departmental LAN serving documents to world-wide clients.

Local Client Backbone Accesses (BL): Every URL request appearing on the Computer Science Department backbone with a client from in the department, naming any server in the world, for a 37 day period in September and October 1995, representing 91,188 accesses requiring transmission of 641.8MB of static Web pages. The requests are for servers both within and outside the .cs.vt.edu domain.

Workloads G and C are identical to those used in [1], while workload U in [1] is a subset of workload U used here.

There are a few caveats in the interpretation of our data based on these workloads. Only workload U represents exactly the requests that a single proxy would cache, because it is a trace from an operational caching proxy acting as a firewall and because no Web servers ran on the network to which the clients were connected. Workload C is representative of a proxy that is positioned within a classroom to serve student machines in the classroom. It would be reasonable for a proxy to cache all requests, even on-campus requests, because there are no servers in the room, and even on-campus requests go to a different campus building in a different sub-domain. Thus results for workload C are probably representative of a real proxy's performance.

However results for workloads BR, BL, and G are upper bounds for what real proxies would experience, because a real proxy would probably not cache requests from clients in .cs.vt.edu to servers in .cs.vt.edu. In particular, workload BR is representative of a cache that is positioned at the point of connection of the Virginia Tech campus to the Internet. Such a cache is useful because it avoids consuming bandwidth on the campus backbone and within the .cs.vt.edu sub-domain. However, 29% of the client requests in the workload are from clients inside the vt.edu domain, which would not be cached by a proxy in this position. In workloads BL and G, a cache would be positioned at the connection from the .cs.vt.edu network to the campus (.vt.edu) backbone. However in BL and G a majority of cache hits are to local servers, and hence would not be cached.

2.1 Workload Collection Procedure

Workloads U, C, and G were collected using a CERN proxy server. Mosaic clients in workload G and Netscape clients in workload C were configured to point to a proxy server running within domain .cs.vt.edu. Workload C client machines were regularly checked to ensure the proxy pointers were in place, whereas workload G clients were unable to modify the proxy pointers. Workload U clients, running on Unix workstations within a lab, were required to point to a CERN proxy server running on a firewall.

Workloads BR and BL were collected at the same time as follows: We ran *tcpdump* on our department backbone to record the prefix of the data field of all packets that list TCP port 80 as either endpoint of a connection. This trace is then passed through a filter (available from http://www.cs.vt.edu/~chitra/www.html) that decodes the HTTP packet headers and generates a log file of all non-aborted document requests in the "common log format" used by NCSA and CERN servers, augmented by additional fields representing header fields not present in common format logs. (The use of the common log format allows our traces to be post-processed by various programs developed for log file analysis.) This collection method is transparent, simple to use because it requires no modification of user preferences or source code of Web clients, and comprehensive in that it collects all HTTP packets that appear on our backbone.

2.2 Workload Characteristics

Simulation results depend on the traces used as input; hence this section characterizes the traced workloads.

Workloads U, G, and C have average request rates under 2000 per day for spring and fall, although the request rate in U soared to about 5000 per day at the beginning of fall semester. The combined BL and BR workloads represent 3000 to 6000 requests per day.

Table 4 shows the distribution of file types arising in each workload, listed by fraction of references and by fraction of bytes transferred. File types are grouped by filename extension (e.g., files ending in *.gif, .jpg, .jpeg*, etc. are considered "graphics"). Those files whose filename extension do not fit into one of the other categories form category "unknown."

The most frequently requested document type was graphics, followed by text (including HTML) in all workloads but one (namely C), where text was first and graphics second. In contrast, there was much more diversity in file type distribution by number of bytes transferred: in workloads U and BL graphics and text accounted for most bytes transferred (78% and 76%, respectively), but graphics and video account for 74% of the bytes transferred in workload C, while

Table 4: File type distributions for workloads used, expressed as percentage of references and bytes transferred.

	U		G		C		BR		BL	
File type	%Refs	%Bytes	%Refs	%Bytes	%Refs	%Bytes	%Refs	%Bytes	%Refs	%Bytes
Graphics	53.00	47.43	51.45	35.39	40.78	35.42	61.66	8.09	51.13	46.26
Text/html	41.46	31.05	45.23	26.56	56.06	19.63	34.11	4.01	43.38	29.30
Audio	0.09	3.15	0.07	1.47	0.21	2.93	2.57	87.78	0.25	17.91
Video	0.19	18.29	0.35	25.77	0.34	39.15	0.00	0.04	0.04	3.58
CGI	0.13	0.08	0.15	0.12	0.12	0.03	0.22	0.00	0.95	0.05
Unknown	5.12	28.23	2.76	10.58	2.49	2.84	1.44	0.07	4.25	2.89

the byte traffic to workload G was more evenly distributed with graphics, text, and video accounting for 88% of the bytes requested. Meanwhile audio and graphics represented 96% of the bytes transferred in workload BR.

The workloads illustrate a phenomena noted elsewhere (e.g., [10]) that a media type may account for a small fraction of references but a large fraction of bytes transferred. For example, video is less than 1% of references in workloads G and C, but 26% and 39% of the bytes transferred, respectively; audio is almost 3% of workload BR, but 88% of bytes transferred (Table 4).

To better understand the distribution of requests, we consider how many servers and unique URLs are requested in the backbone local client traffic (BL). Fig. 1 shows the distribution of requests to the 2543 unique servers referenced in the trace; 10 or fewer requests went to 1666 servers, while 100 or more requests went to only 84 servers (13 of the top 20 servers were outside the "vt.edu" domain and would benefit from caching). Fig. 1 suggests that the number of requests to each server in workload BL follows a Zipf distribution. (In comparison, the *requested URL* in traces studied elsewhere [4, 9] also follows a Zipf distribution.)

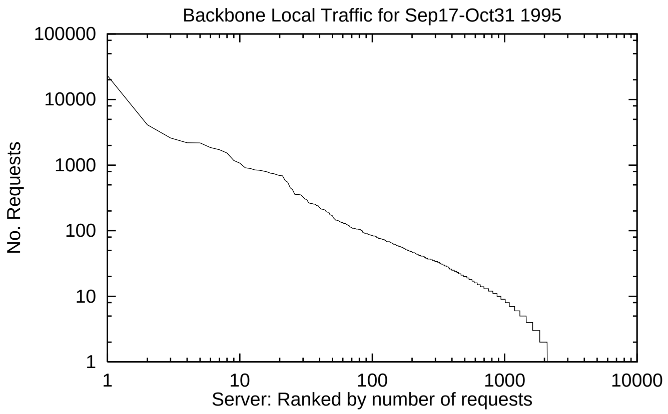

Figure 1: Distribution of requests for particular servers for workload BL.

Many clients in Fig. 1 are requesting URLs from a few servers, suggesting a high degree of concentration in workload BL. An alternate view is the distribution of number of bytes transferred (Fig. 2), which represents the volume of traffic returned by servers. For the period September 17 through October 31, approximately 290 URLs of 36,771 unique URLs referenced returned 50% of the total requested bytes.

The concentration of requests to and bytes transferred from a few servers means that this workload contributes substantially to the packet arrival rate at certain Web servers. Therefore, Figs. 1 and 2 suggest that widespread proxy caching in networks with workloads like ours could dramatically reduce the load on popular Web servers.

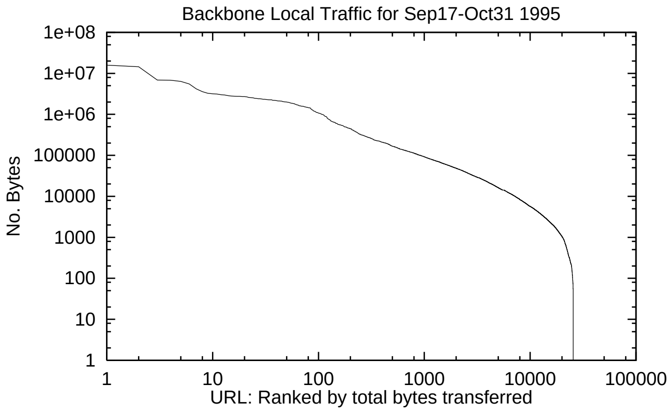

Figure 2: Distribution of bytes transferred for each URL for workload BL.

3 Experiment Objectives and Design

3.1 Objectives

Our experiments assess the following:

1. What is the maximum possible WHR and HR that a cache could ever achieve, no matter what removal policy is used?
2. How large must a cache be for each workload so that no document is ever removed?
3. What removal policy maximizes weighted hit rate for each workload over the collection period?
4. How effective is a second level cache?
5. Given a fixed amount of disk space for a cache, should all document types be cached together, or should the cache be partitioned by document type? (A partitioned cache would avoid the situation where one very large file displaces all other files in the cache [10], possibly increasing overall hit rate.)

3.2 Experiment Design

Table 5 defines four experiments. Experiment 1 addresses objectives 1 and 2 in §3.1. To compute the maximum possible weighted hit rate, we simulate each workload with an infinite size cache. The cache size at the end of simulation is then the size needed for no document replacements to occur, denoted *MaxNeeded*, and addressing objective 2. Experiment 2 addresses objective 3 through simulation of each workload with all combinations of keys listed in Table 1 as primary and secondary keys with two cache sizes, either 10% or 50% of MaxNeeded. Experiment 3 addresses objective 4, by simulating a two level cache, whose first-level size is either 10% or 50% of MaxNeeded, and using an infinite size second level cache to derive the maximum possible second level hit rate. Finally, experiment 4 investigates objective 5 on a cache whose size is 10% of MaxNeeded, using one workload, BR. BR is the studied workload because 88% of the bytes transferred are audio. In the experiment, the cache is divided into two partitions, with one partition exclusively for audio and the other for non-audio document types. The partition size is varied so that the audio partition size is either 1/4, 1/2, or 3/4 of the whole cache size, with the non-audio partition equal to the remainder. In all experiments, three response variables are measured: hit rate, weighted hit rate, and the maximum cache size needed during the simulation.

All experiments are initiated with an empty cache and run for the full duration of the workload. The simulation reports WHR and HR for each day separately. There is great variation in daily hit rates; thus graphing the daily rates produces curves with so much fluctuation that it is difficult to see trends and compare curves. Therefore we apply a 7-day moving average to the daily hit rates before plotting. We chose a one week period because it was large enough to cover the typical "cycle" of activity without removing longer term trends in access patterns. In particular, each plotted point in a hit rate graph represents the average of the daily hit rates for that day and the six preceding days. No point is plotted for days zero to five, which are the days without six preceding days.

4 Experiment Results

4.1 Experiment 1: Maximum Possible Hit Rate

Figures 3 to 7 show WHR and HR for workloads U, G, C, BR, and BL. In each workload, HR ranges from 20% to just over 98%, and WHR ranges from 12% to 98%. HR is usually equal to or greater than WHR in workloads U, G, and C. In workload BR, HR and WHR reach the same value.

Some seasonal variations are present in the longest trace (workload U), in Fig. 3. The temporary drop in WHR and HR around day 50 is the week between the spring and summer semesters. Around day 155 the hit rates permanently decline; this is the start of the fall semester. New users and a dramatic increase in the rate of accesses are the most probable causes for the decline in hit rate.

In contrast to U, the observation periods for workloads G and C are entirely contained in the spring semester. The hit rates for workload G (Fig. 4) tend to increase over the semester to a maximum between 80% and 90% (the sudden drop from day 66 to day 71 is due to the "time-shared client" machine being down). We expected workload C (Fig. 5) to exhibit some of the highest rates, because in a classroom setting students often follow the teacher's instructions in opening URLs or following links. Instead, the graph shows hit rates fluctuate between extremes throughout the semester.

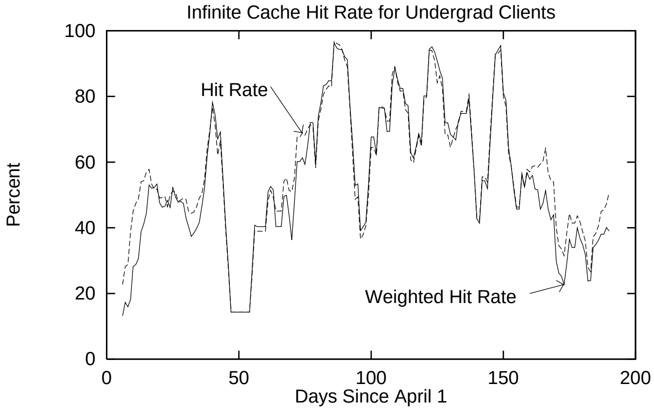

Figure 3: Maximum achievable hit-rate for workload U.

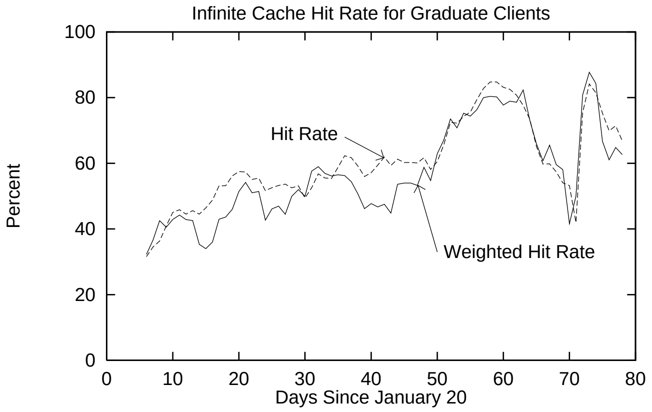

Figure 4: Maximum achievable hit-rate for workload G.

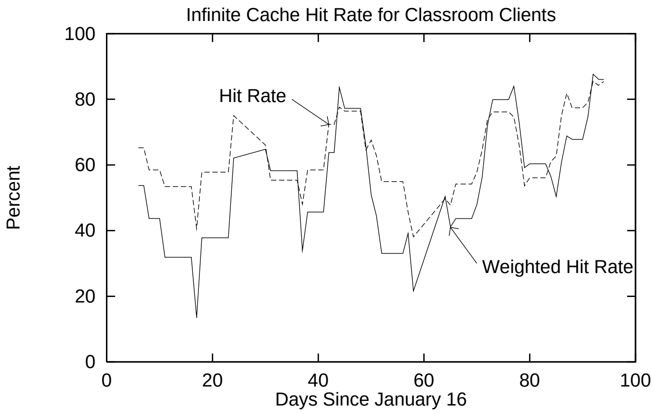

Figure 5: Maximum achievable hit-rate for workload C.

Table 5: Factor-level combinations for all experiments.

	Levels			
Factor	*Experiment 1*	*Experiment 2*	*Experiment 3*	*Experiment 4*
Cache levels	1	1	2	1
L1 cache size	infinite	10, 50% of MaxNeeded	10, 50% of MaxNeeded	10% of MaxNeeded
L2 cache size	n/a	n/a	infinite	n/a
Workloads	U,G,C,BR,BL	U,G,C,BR,BL	U,G,C,BR,BL	BR
Keys (Prim.,Sec.)	n/a	(All in Table 1, Random); ($\lfloor \log_2(\text{SIZE}) \rfloor$, All in Table 1 but $\lfloor \log_2(\text{SIZE}) \rfloor$)	(Best in Exp. 2, Random)	(Best in Exp. 2, Random)

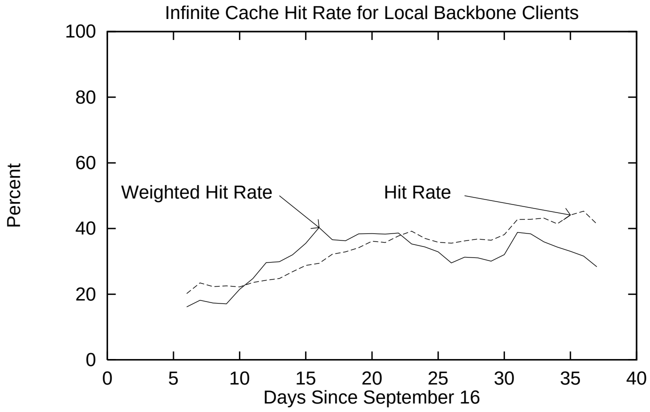

Figure 6: Maximum achievable hit-rate for workload BL.

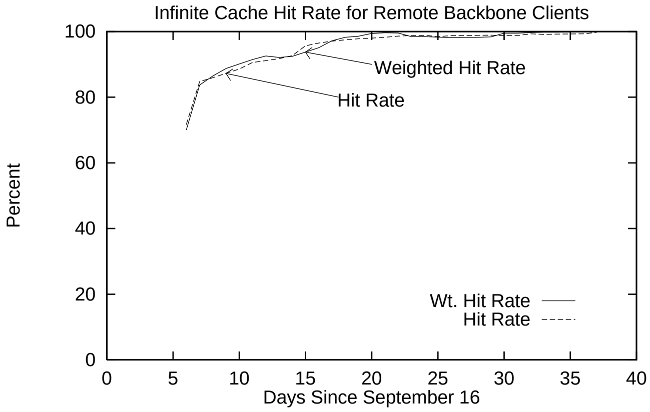

Figure 7: Maximum achievable hit-rate for workload BR.

(The horizontal lines in the curves of Fig. 5 arise because classes met on four days each week, and there were no URLs traced for the other three days each week. The class also, occasionally, took field trips, reducing the number of collection days further. Every plotted point is the average of hit rates for however many class days occurred in that day and the preceding six days.)

Workload BR achieves the highest hit rates by far — over 98% for most of the collection period. Two reasons account for the high rates. First, all URLs named one of a small set of Web servers in domain .cs.vt.edu, unlike other workloads, where URLs could name servers anywhere in the world. Second, as mentioned earlier, traffic to popular Web pages with audio files on a single server dominated the traffic.

For caches to have sufficient size to never replace a document, they must have the following sizes: 191 Mbytes for workload C, 388 Mbytes for G, 403 Mbytes for BL, 431 Mbytes for BR, and 631 Mbytes for U.

4.2 Experiment 2: Removal Policy Comparison

4.3 Primary Key Performance

The preceding experiments give the maximum theoretical hit rates, because the simulation uses infinite cache size and hence no removals occur. In Experiment 2 we simulate a finite cache size to investigate what removal policy performs best. Recall from Table 5 that we compare results for a cache that is half or one tenth of the size needed for no replacement. Rather than plotting WHR in our graphs, we will plot the ratio of WHR in Experiment 2 to the value obtained in Experiment 1 (the theoretical maximum). The resulting fraction reports how close to optimal a replacement policy performs: the closer a curve is to the line $y = 1$ the closer the policy is to the theoretical maximum WHR. To interpret the graphs, the reader may want to compare the corresponding Experiment 2 and Experiment 1 graphs, to determine whether the absolute HR and WHR that a replacement policy yields is high or low.

Consider first the case generating the most replacement, namely cache size equal to 10% of MaxNeeded. Figures 8 to 12 shows results for various primary keys with a secondary key of random. To simplify the graphs, keys $\lfloor \log_2(\text{SIZE}) \rfloor$ and DAY(ATIME) are not shown. With one exception (days 60-70 of workload G) $\lfloor \log_2(\text{SIZE}) \rfloor$ always yields equal or slightly higher WHR than SIZE. DAY(ATIME) is within about 5% of ATIME, except in workload BR when it is always worse than ATIME but never as bad as NREF.

The results for a cache whose size is 50% of MaxNeeded are not shown, because the curves follow the same trend as in the 10% graphs. However the magnitude of difference between the WHR curves is smaller at 50%. Also the results are similar with respect to HR, except that key NREFS yields a clearly lower hit rate than any other policy.

Each of the sorting keys plotted is discussed below.

SIZE: Remarkably, in four of the workloads (U, G, C, and BR), some replacement policy achieves a WHR that is over 90% of optimal most of the time, *even though the cache size is only 10% of MaxNeeded.* The second surprise is that *replacement based on either SIZE or* $\lfloor \log_2(\mathit{SIZE}) \rfloor$ *always outperforms any other replacement criteria.*

Why does SIZE work well? The histogram for workload

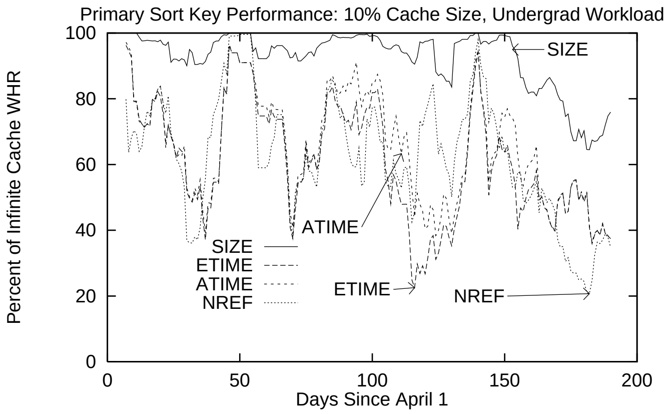

Figure 8: Ratio of WHR for various primary keys and cache size of 10% of MaxNeeded to WHR of Fig. 3 in workload U.

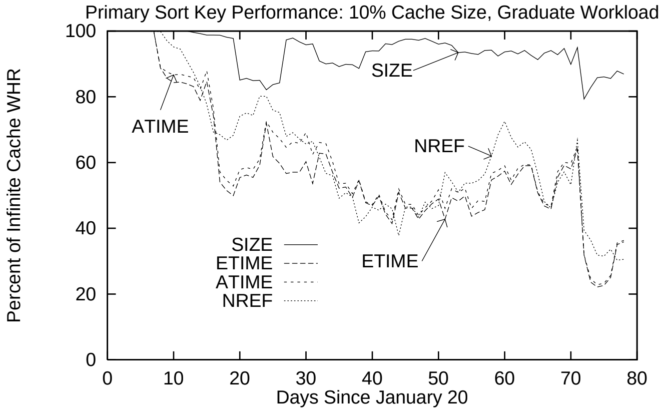

Figure 9: Ratio of WHR for various primary keys and cache size of 10% of MaxNeeded to WHR of Fig. 4 in workload G.

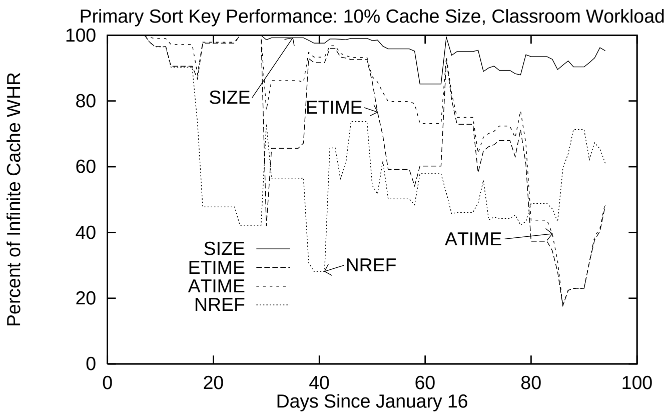

Figure 10: Ratio of WHR for various primary keys and cache size of 10% of MaxNeeded to WHR of Fig. 5 in workload C.

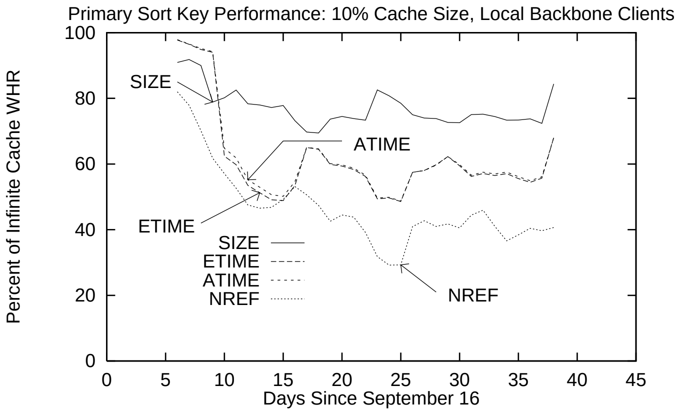

Figure 11: Ratio of WHR for various primary keys and cache size of 10% of MaxNeeded to WHR of Fig. 6 in workload BL.

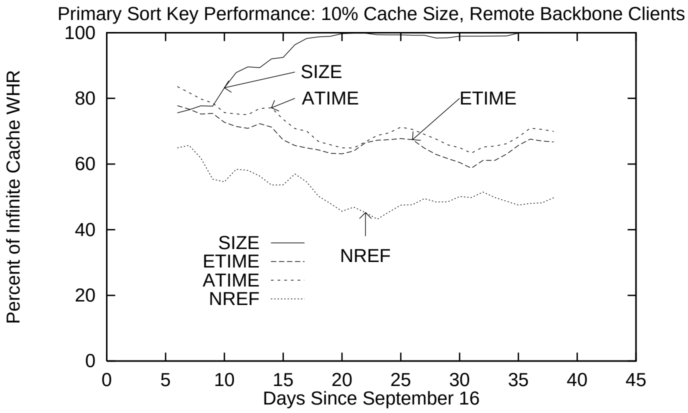

Figure 12: Ratio of WHR for various primary keys and cache size of 10% of MaxNeeded to WHR of Fig. 7 in workload BR.

BL in Fig. 13 shows that most requests in the trace are for small documents; generally similar results for workloads U and G are reported in [2, Figs. 1,2,4]. The fact that SIZE keeps small files in the cache makes HR high for two reasons: most references go to small files, and removal of a large file makes room for many small files, which increases the percentage of URLs in the cache that *can* be hit.

The fact that most requests are for small document sizes is not surprising for three reasons. First, users probably tend to avoid using large documents due to the time required to download them. Second, many popular Web pages are created by professional designers who keep graphics small to increase their usability. It is also apparent from the size distribution and file type distribution [2] that a many URL references are for embedded iconic images. Third, even in the case that users make frequent use of multi-page documents, it may be that users prefer to print the documents and repeatedly reference hard-copy rather than on-line copies.

The time between references is also important as it is typically the primary measure for determining which file to delete from a cache (e.g., in LRU). Figure 14 plots one point for each request in workload BL. There are a fairly large number of references to files in the 1-2MB range. However, the removal of one 2MB file would make room in the cache for 1000 2kB files, and 1000 2kB files referenced 1000 times each will yield a higher WHR than one 2MB file being referenced ten times.

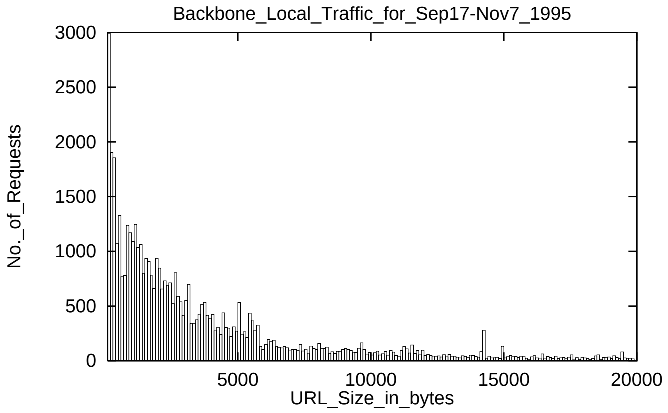

Figure 13: Distribution of document sizes in workload BL.

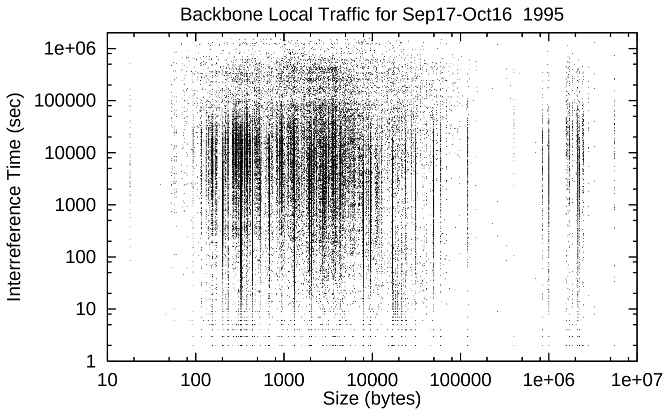

Figure 14: Scatter plot of size and time since last reference of each URL referenced two or more times in workload BL.

ATIME: The center of mass in Fig. 14 lies in a region with relatively small size (just over 1kB) but large interreference time (about 15,000 seconds or 4.1 hours). This implies that the ATIME key (and hence LRU) discards many files that will be referenced in the future. If ATIME was a good policy, then a trace should exhibit temporal locality: frequent references to the same document within a short interval of time. We could test for temporal locality by plotting a histogram of the interreference time of each URL referenced two or more times, and checking whether its mass is concentrated at short times with a tail at long times. Such a histogram could be derived from Fig. 14 by counting how many points lie on every horizontal line that could be drawn through the graph. Most points lie on lines roughly in the range $y = 1000$ to $y = 100000$. So the histogram would have a tail as interreference time goes to 0. Therefore, we expect ATIME to be a poor sorting key, and this turns out to be the case in Figures 8 to 12. This result is also consistent with [5], which observes that LRU is not a good *server* cache removal policy, and concludes that temporal locality does not exist in requests reaching a server because servers multiplex requests of many clients and because client-side caching removes temporal locality from the reference stream.

NREFS and ETIME: ETIME performs about the same as ATIME for all workloads. In workloads U and G NREFS performs about the same as ATIME and ETIME. However, in workload C key NREFS first performs significantly worse and later better than ETIME and ATIME. At the end of the course from which C was collected, students reviewed important Web pages to prepare for the final, so it appears that the number of references earlier in the semester correlated with importance of a page and hence performed better than ETIME or ATIME. In workloads BR and BL, NREFS performed significantly worse than ETIME and ATIME.

4.4 Secondary Key Performance

Next consider the choice of secondary key. As listed in Table 5, we consider the question with respect to one primary key: $\lfloor \log_2(\text{SIZE}) \rfloor$. (We consider one primary key because "size" keys outperform all others, so it is pointless to consider other keys. Furthermore, $\lfloor \log_2(\text{SIZE}) \rfloor$ yields more ties than SIZE, and hence would exercise the secondary key more than SIZE.)

How much better (or worse) does each possible secondary key do compared to a random secondary key? To answer this, we plot in Fig. 15 the ratio of WHR obtained for each possible secondary key to the WHR obtained with random as a secondary key for one workload, G. (The figures shows a cache size of 10% of MaxNeeded; using 50% yields a similar plot.) If the ratio is consistently larger than 1, then a non-random secondary key is useful; otherwise random is as good as any other secondary key. DAY(ATIME) yields the highest hit rate, followed by SIZE, ETIME, ATIME, and NREF. Also ATIME and NREF both occasionally yield worse hit rates than random selection. However, this graph is atypical, because for the other four workloads (not shown) no key consistently outperforms random. Therefore there is no conclusive evidence to use a non-random secondary key with $\lfloor \log_2(\text{SIZE}) \rfloor$ as a primary key.

We did examine the LFU policy by considering a primary key of NREF and all possible secondary keys, again with workload G. In contrast to Fig. 15, *all* secondary keys at some point during simulation performed worse than random. The best secondary key was $\lfloor \log_2(\text{SIZE}) \rfloor$. The best performing key from Fig. 15, DAY(ATIME), performed erratically – doing better as often as it did worse than random. The lesson is that using SIZE or $\lfloor \log_2(\text{SIZE}) \rfloor$ as a primary or secondary key is a good idea.

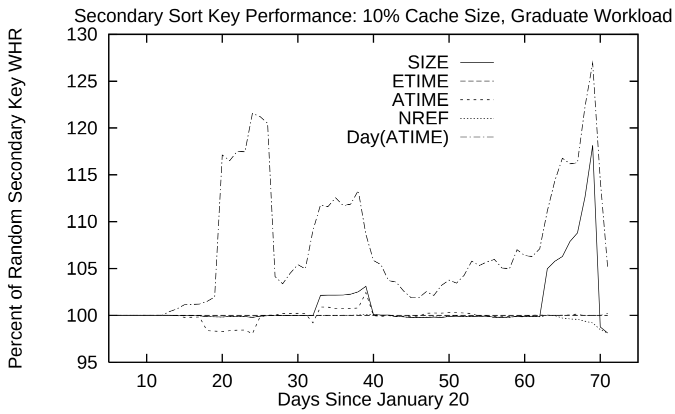

Figure 15: Ratio of WHR for various secondary keys and cache size of 10% of MaxNeeded to WHR with random secondary key (workload G, primary key $\lfloor \log_2(\text{SIZE}) \rfloor$).

4.5 Experiment 3: Effectiveness of Two Level Caching

Experiment 3 (from Table 5) uses the best policy from Experiment 2 (SIZE) for the primary key and random for the secondary key. The primary cache is set to one of two sizes, 10% or 50% of MaxNeeded, and the second level cache has infinite size. When a document request is a miss in the primary cache, the request is sent to the second level cache. If the second level cache has the document, it returns a copy of the document to the primary cache; otherwise the second level cache misses and the document is placed in both the second level and primary cache. This implies that when a primary cache removes a document, the document will always be in the second level cache; this represents a possible implementation strategy of a primary cache sending replaced documents to a larger second level cache.

Figures 16 to 18 show the three trends that emerged from the five workloads. In all three figures with a primary cache size of 50% the second level cache makes virtually no difference, because its HR and WHR are both less than 3%, except workload G where WHR jumps to 10% briefly. However, in a memory-starved primary cache (the 10% of MaxNeeded case), the second level cache reaches a maximum 9-35% HR, and a 15-55% WHR. Therefore the second level cache is playing an important role of an "extended memory" for a small primary cache. In particular, because SIZE is the primary key for the primary cache, larger documents will be displaced to the second level cache. This explains why WHR is larger than HR — primary cache misses that are hits in the secondary cache are for large files.

The only differences among the three graphs are whether the 10% WHR curves level out and to what extent the second level cache is utilized. In Fig. 16 the WHR levels off at between 10% and 15%. In contrast, Fig. 17 shows a workload whose "working set" of documents can fit in the primary cache of only 10% of MaxNeeded for over a month, after which the second level cache experiences a rapid growth in HR and WHR. Fig. 18 shows a WHR that fluctuates throughout the collection period. Each increase in the second level cache WHR correspond to sharp increases in the infinite cache hit rate that could not be handled by the limited 10% primary cache (see Fig. 4).

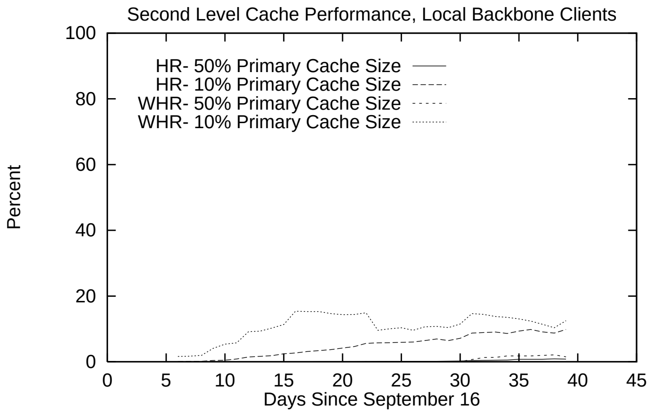

Figure 16: HR and WHR second level cache in workload BL with first level cache size of 10% and 50% of MaxNeeded.

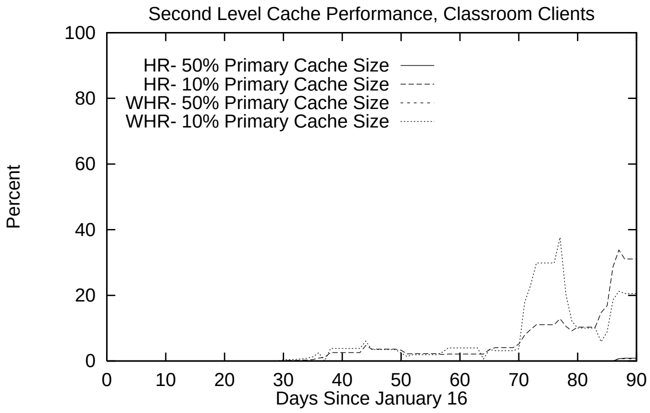

Figure 17: HR and WHR for second level cache in workload C with first level cache size of 10% and 50% of MaxNeeded.

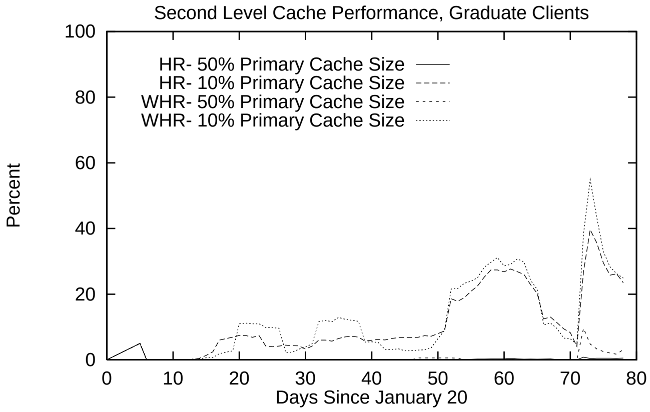

Figure 18: HR and WHR for second level cache in workload G with first level cache size of 10% and 50% of MaxNeeded.

4.6 Experiment 4: Effectiveness of Partitioned Caches

Recall from Table 4 that most of the bytes transferred in the BR workload are audio. Do clients that listen to music degrade the performance of clients using text and graphics? Could a partitioned cache with one portion dedicated to audio, and the other to non-audio documents increase the WHR experienced by either audio or non-audio documents?

In Experiment 4, a one-level cache with SIZE as the primary key and random as the secondary key was used with three partition sizes: dedicate 1/4, 1/2, or 3/4 of the cache to audio; the rest is dedicated to non-audio documents. Comparing Figures 19 (the audio partition) and 20 (non-audio) reveals that periods of heavy audio use overwhelm even a 3/4 audio partition with a 10% cache size, and there is very little difference between the 3/4 and 1/2 non-audio partition hit rate. Therefore, for this workload, splitting the cache into two partitions of equal size would increase the audio WHR substantially, while maintaining a high WHR on other file types. Note that in Figs. 19 and 20, the HR and WHR reported are those *within* the category (i.e., audio HR is the number of audio hits for all audio references).

5 Conclusions and Future Work

Removal policies in network caches for WWW documents that use size clearly outperform any other removal criteria. Consistently, in our simulations of all five workloads, primary keys SIZE and $\lfloor \log_2(\text{SIZE}) \rfloor$ achieve a higher hit rate and weighted hit rate than any other policy. This behavior is confirmed by histograms of how many references went to

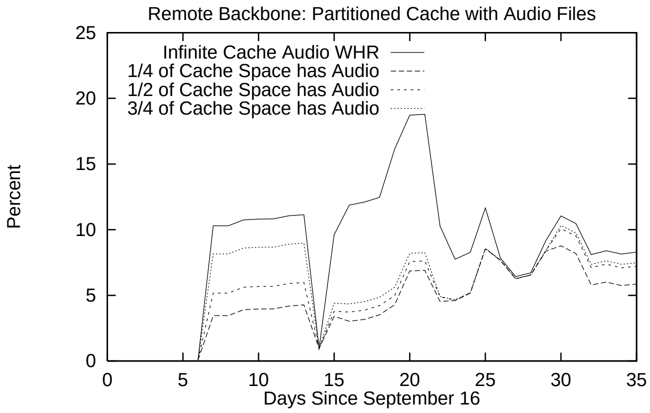

Figure 19: WHR for audio requests in workload BR for partitioned cache with total size of 10% of MaxNeeded.

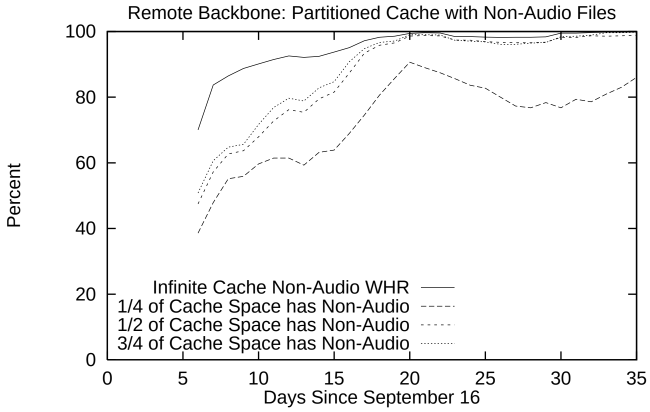

Figure 20: WHR for non-audio requests in workload BR for partitioned cache with total size of 10% of MaxNeeded.

files of different sizes, where the mass is concentrated in file sizes of under 1KB (Fig. 13).

Therefore the previously proposed LRU-MIN policy [1] (that uses a primary key like $\lfloor \log_2(\text{SIZE}) \rfloor$) is one of the best policies, but the computationally simpler rule of just using SIZE as a key also works well. The popular LRU policy, equivalent to using primary key ATIME, ranks next in performance, but with much lower hit and weighted hit rates. A primary key of NREF, which is used as the primary key in the Hyper-G policy, performs the *worst* of all primary keys in our study.

In fact, Hyper-G's policy uses NREF, ATIME, and SIZE as the primary, secondary, and tertiary keys. Based on our study we suggest *reversing* the ranking: use SIZE, then ATIME, then NREF.

Finally, the Pitkow/Recker policy, designed as a server cache policy but investigated here as a proxy cache policy, uses DAY(ATIME) as a primary key if there are documents older than one day in the cache, and otherwise using SIZE. While it is intuitively attractive to replace documents that have not been used for days, in our study, DAY(ATIME) is one of the worst performing primary keys (although it is the best performing secondary key in one workload). Replacing days-old files dramatically reduced HR and WHR in our study. Therefore the Pitkow/Recker policy would work better if it simply used SIZE alone as a key.

The use of SIZE as a primary key blends well with a two level cache hierarchy in which documents that the primary cache replaces are sent to a larger second level cache. The arrangement is natural, because SIZE as a primary key will always transmit the largest document from primary to second level cache. Coupling the results from Experiments 3 and 4, a primary key of SIZE in the primary cache keeps the primary cache weighted and unweighted hit rates high, while a second level cache for large documents that overflow primary caches significantly boosts the weighted hit rate of byte-intensive media types (see Fig. 19).

A number of open problems on proxy caching remain:

1. Certain sorting keys for removal algorithms have never been explored in caching proxy implementations or simulation studies to our knowledge, but have intuitive appeal. The first is *document type.* A sorting key that puts text documents at the front of the removal queue would insure low latency for text in Web pages, at the expense of latency for other document type. The second sorting key is *refetch latency.* To a user of international documents, the most obvious caching criteria is one that caches documents to minimize overall latency. A European user of North American documents would preferentially cache those documents over ones from other European servers to avoid using heavily utilized transatlantic network links. Therefore a means of estimating the latency for refetching documents in a cache could be used as a primary sorting key.

2. A second open problem is how caching can help dynamic documents. Today, caches cannot cache dynamic documents. On closer inspection, a cache is only useless for dynamic documents if the document content completely changes; otherwise a portion but not all of the cached copy remains valid. Therefore one solution we envision is an HTTP protocol change to allow caches to request the *differences* between the cached version and the latest version of a document. For example, in response to a conditional GET a server could send the "diff" of the current version and the version matching the `Last-Modified` date sent by the client; or a specific tag could allow a server to "fill-in" a previously cached static "query response form." Another approach to changing semi-static pages (i.e., pages that are HTML but replaced often) is to allow Web servers to preemptively update inconsistent document copies, at least for the most popular files. The issue is discussed in [6].

3. We observed a 15% to 55% WHR in a second level cache with a primary cache that is 10% of the size needed for no replacement. How would this hit rate change if a single second level cache handled misses from a set of primary caches? Whereas we observed concentration in each individual workload of the five we studied, how much commonality exists between the workloads if they share a single second level cache? An interesting future study would be simulation of a multi-level cache more complex than the single first and second level configuration used here.

4. A final open problem is to study the interaction of removal algorithms with algorithms that identify when cached copies may be inconsistent, such as expiration times or the time of last modification for documents. For example, the Harvest cache [8] tries to remove expired documents first.

Caching has a bright future because Web users do not aimlessly and randomly request Web pages, according to our workloads. In fact, there is a significant (and to us unexpected) amount of concentration exhibited by all of the collected workloads. One workload — off campus clients requesting documents from Web servers in our department — reaches a surprising weighted hit rate of 95% (averaged over all days in the trace). This is the effect of a popular single Web site, and the fact that a large group of clients is accessing a small group of servers. More telling are three other workloads (U, G, and C), which all have mean weighted and unweighted hit rates of around 50% (averaged over all days in a trace) allowing a cache to cut in half their bandwidth and server access demands. Another indication of locality is that the server accessed appears to follow a Zipf distribution. In the BL trace, most requests went to a small number of servers, and a small number of the URLs used accounted for most bytes transferred over the network. This suggests either that each user tends to access the same servers and URLs over and over, or that many users request the same set of servers or URLs at the same time.

6 Acknowledgements

We thank Jeffrey Mogul, Carey Williamson, and the referees for their many comments and suggestions on this paper; Laurie Zirkle for help in setting up our network monitors; and Leo Bicknell and Carl Harris for providing the workload U trace logs. This work was supported in part by the National Science Foundation through CISE Institutional Infrastucture (Education) grant CDA-9312611, CISE RIA grant NCR-9211342, and SUCCEED (Cooperative Agreement No. EID-9109853). SUCCEED is an NSF-funded coalition of eight schools and colleges working to enhance engineering education for the twenty-first century.

References

[1] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching proxies: Limitations and potentials. In *4th International World-wide Web Conference*, pages 119–133, Boston, Dec. 1995. <URL: http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html>.

[2] M. Abrams, S. Williams, G. Abdulla, S. Patel, R. Ribler, and E. A. Fox. Multimedia traffic analysis using Chitra95. In *Proc. ACM Multimedia '95*, pages 267–276, San Francisco, Nov. 1995. ACM.

[3] K. Andrews, F. Kappe, H. Maurer, and K. Schmaranz. On second generation hypermedia systems. In *Proc. ED-MEDIA 95, World Conference on Educational Multimedia and Hypermedia*, Graz, Austria, June 1995. <URL: http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/claffy/main.html>.

[4] M. F. Arlitt. A performance study of internet web servers. Master's thesis, Computer Sci. Dept., University of Saskatchewan, Saskatoon, Saskatchewan, May 1996.

[5] M. F. Arlitt and C. L. Williamson. Web server workload characterization: The search for invariants. In *Proc. SIGMETRICS*, Philadelphia, PA, Apr. 1996. ACM.

[6] T. Berners-Lee. Propagation, replication and caching. <URL: http://www.w3.org/hypertext/WWW/Propagation/Activity.html>, Mar. 1995. World Wide Web Consortium.

[7] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – HTTP 1.0. <URL: http://www.w3.org/pub/WWW/Protocols/HTTP/1.0/spec.html>, Feb. 1996.

[8] A. Chankhuthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrel. A hierarchical internet object cache. <URL: ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/HarvestCahce.ps.Z >.

[9] C. R. Cunha, A. Bestavros, and M. E. Crovella. Characteristics of www client-based traces. Technical Report TR-95-010, Computer Sci. Dept., Boston Univ., July 1995.

[10] T. T. Kwan, R. E. McGrath, and D. A. Reed. NCSA's world wide web server: Design and performance. *IEEE Computer*, 28(11):68–74, Nov. 1995.

[11] A. Luotonen and K. Altis. World-Wide Web proxies. *Computer Networks and ISDN Systems*, 27(2), 1994. <URL: http://www1.cern.ch/PapersWWW94/luotonen.ps>.

[12] R. Malpani, J. Lorch, and D. Berger. Making World Wide Web caching servers cooperate. In *4th International World-wide Web Conference*, pages 107–117, Boston, Dec. 1995.

[13] J. E. Pitkow and M. M. Recker. A simple yet robust caching algorithm based on dynamic access patterns. In *Proc. 2nd Int. WWW Conf.*, pages 1039–1046, Chicago, Oct. 1994.

[14] A. A. B. Pritsker. *Introduction to Simulation and SLAM II*. John Wiley, Halsted NY, third edition, 1987.

[15] L. Schruben. Simulation modeling with event graphs. *Commun. ACM*, 26:957–963, 1988.

[16] A. Silberschatz and P. B. Galvin. *Operating Systems Concepts*. Addison Wesley, Reading, MA, fourth edition, 1994.

Appendix: Proxy Cache Simulator

A discrete event world view simulation model, available from http://www.cs.vt.edu/~chitra/www.html, was developed using event graphs [15] and implemented using the SLAM II simulation language [14]. Output measures include summaries for time from request to end of file transfer, file transfer rate, cache hit rate, cache hit rate from input file, file size from input file, in-bound transmission rate, time in cache, time in cache, file size in cache at end of simulation file size for hits, file size for misses, number of files with the same primary key value on initial input to the cache, percent of URL ties on the primary key, and number of references to files in the cache. Traces can also be created for transmission time, hit rate, transmission rate, files in cache at end of simulation (time in cache), and files leaving the cache (time in cache).