

CACHING CHARACTERISTICS OF INTERNET AND INTRANET WEB PROXY TRACES

Arthur Goldberg, Ilya Pevzner, Robert Buff

Computer Science Department
Courant Institute of Mathematical Science
New York University

{artg, pevzner, buff}@cs.nyu.edu

www.cs.nyu.edu/artg, www.cs.nyu.edu/phd_students/{pevzner, buff}

This paper studies the caching characteristics of HTTP requests and responses that pass through production Web proxies. We evaluate caching opportunities and problems. Traces with 5.9 million entries from a large Internet Service Provider (ISP) and 2.0 million entries from an Intranet firewall are studied. We find maximum cache hit rate opportunities of about 40% for an ISP and 70% for an Intranet. Cache size needs and document residence times are also examined.

1. Introduction

Caching proxies [LUOT94] play an important role in IP networks, enhancing security, conserving bandwidth and potentially improving performance by reducing response time (e.g. [ABRA95] and [HAMI98]). Designing and deploying proxies is challenging work because proxies are subject to complex inputs which can be summarized as a request stream and associated responses. In this paper we characterize realistic proxy inputs to help people who design proxies and people who architect and operate networks with proxies.

2. Proxy Data Sources

We have analyzed proxy traces collected in a large ISP and a large intranet. We now discuss these environments.

ISP

Prodigy Internet, the world's 5th largest ISP with approximately 450,000 users, maintains a United States network of eight proxy servers. When a user logs onto Prodigy their browser is configured to use proxy.prodigy.net as a proxy. A custom DNS server resolves the domain name proxy.prodigy.net to the IP address of one of the eight proxies. A user remains connected to the same proxy for the duration of their session. The resolution alternates among the proxies in a round-robin fashion. Thus, each proxy services users from the entire United States.

A typical proxy server supports about 500 unique clients (averaged over a 5-minute period) with an average load of 30 requests per second during peak hours, or approximately 10^6 requests per day. Between the fall of 1996 and now (summer 1998) the proxies have been implemented with Netscape 2.5 on IBM RS/6000 systems running AIX 4.1 and equipped

with 256 MB of RAM and three 4 GB disks. Each proxy has been configured with a cache of 5.5 GB spread over the 3 disks. Other relevant cache configuration parameters [NETS97] have been set as follows:

- $max\text{-}unchecked^1 = 21600$ seconds (6 hours)
- $lm\text{-}factor^2 = 0.1$
- $term\text{-}percent^3 = 80\%$.

We analyzed traces from the proxy proxy3.ykt.prodigy.net located in Yorktown, New York. We examined traces collected between June 8 and 13, 1998. This log has very few entries for Tuesday, June 9, suggesting that the proxy server was down or being maintained on that day. The log consisted of 5.9 million entries or about 983300 entries per day. The traces were in Netscape *Extended-2* log format [NETS97] with a record for each request-response pair containing the following fields, among others:

¹ $max\text{-}unchecked$ is the maximum time allowed between consecutive up-to-date checks.

² $lm\text{-}factor$ is used to estimate the duration for which the document will remain unchanged, which is known as the expiration time. The estimated expiration time is given by the time elapsed since the last modification multiplied by $lm\text{-}factor$. Note that this only guesses how long a document might remain up-to-date.

³ If the client interrupts a response when a document has been only partly retrieved from the server then the proxy attempts to complete the retrieval if at least $term\text{-}percent$ of the document has already been retrieved. Otherwise, the proxy closes the server connection and removes the partial file.

- the HTTP request line as it came into the proxy
- the content-length sent to the client by the proxy, in bytes
- the client's finish status
- the cache's finish status⁴
- the HTTP response code sent by the server

Intranet

The intranet trace comes from the Union Bank of Switzerland (UBS), an investment bank with approximately 8,000 employees in New York City and Switzerland. UBS runs an HTTP proxy in the firewall that connects their intranet to the Internet. They operate a network of two Squid 1.1.21 proxies running on Sun Ultra 1 servers with 1 G RAM and 7 4 GB disks. Each proxy is configured with disk cache size of 12 GB and memory cache size of 750 MB. The 12 GB disk cache is built from three 4G disks.

For UBS, we analyzed traces from uma.ny.ubs.com located at 299 Park Ave. in New York City. We examined traces collected between March 12 and 16, 1998. During that time, the trace length averaged 500 thousand records per day. Trace entries were in the Extended Log Format [PHIL96], which includes all HTTP request and response headers transferred through the proxy and the following fields:

- date of request
- time of request
- time to service request, in milliseconds
- squid action [NLAN96]
- HTTP response status [FIEL97]
- number of bytes transferred
- request method [FIEL97]
- requested URI [FIEL97]

3. Analyzing Caching Properties

We have begun our analysis by asking, "Given these traces what is the best a cache can do? How high a hit rate it can achieve? What would be the smallest cache that can achieve that hit rate?" Answers to these questions will set standards by which real caches can be judged. To answer these questions we begin by imagining a cache with infinite space. In Section 5 we examine finite sized caches.

We wrote code to analyze traces. The analysis code was tested by comparing results with Netscape's analysis tool, analyzing traces backwards and verifying expected symmetries, and by obtaining the same results with two separate implementations.

Given a sequence of HTTP request/response pairs obtained from a proxy trace, the trace analysis program computes the following characteristics:

- Hit Ratio (HR) = (number of documents served from the cache) / (total number of documents served by the proxy)
- Fraction of Bytes Transferred saved by caching (BT) = (total number of data bytes served with documents in the cache) / (total number of bytes served by the proxy)

Assuming the cache's size is unlimited, we compute these values in several ways. To characterize the cacheability of traces we define HR and BT under the assumptions that 1) all documents are cacheable, or 2) documents that meet the HTTP 1.1 specification requirements or the Netscape requirements are cacheable (see table 1).

The first two columns of Table 1 lists the names we use to denote the curves in Figures 1 and 2. The third column defines the set of documents, and the last column describes the set of documents in detail.

To characterize document expiration, the analysis program also computes the ratio of the number of cache refreshes to the number of cache hits, which we call RR. A refresh occurs when a request finds an expired response in the cache so the proxy re-fetches the document from the origin server.

Pseudo-code for computing HR, BT and RR [PEVZ98] as well as Tcl source code for the analysis program [PEVZ98a] are available.

⁴ The cache's status marks each entry as non-cacheable, cache hit, cache miss, or cache refresh.

Analysis name		Documents not cached	Description of set of documents
Hit Rate	Fraction of bytes saved		
HR(ALL)	BT(ALL)	None	The curves HR(ALL) and BT(ALL) show the maximum potential hit ratio and bytes saved, respectively, based on only the URLs of requests in the proxy trace.
HR(-RFC)	BT(-RFC)	Non-cacheable	For the Prodigy traces, where the HTTP headers were not available, the cache status trace field [NETS97] were used to determine cacheability. For UBS, the HTTP/1.1 Specification was used to determine cacheability. See the implementation at [PEVZ98].
HR(-ACTUAL)	BT(-ACTUAL)	Actual misses	The curves HR(-ACTUAL) and BT(-ACTUAL) show the actual hit ratio and bytes saved as indicated by the cache status 'hit' in the trace. Actual misses reflect the finite cache size.

Table 1

4. Caching Characteristics

Figures 1 and 2 show the analysis of the Prodigy (only the first 5 million entries were analyzed) and UBS traces, respectively. They show trace characteristics against the trace length, given by entry number. This analysis assumes the cache starts empty and keeps the most recent copy of every cacheable response. The curves show cumulative ratios, for example, HR(-RFC) of 42% at trace entry 5 million for Prodigy means that assuming an infinite cache, 2.1 million of the 5 million requests would have been served from the cache. At the same point in the trace, the actual fraction of bytes delivered by the proxy from the cache was only 13%.

In both Figures 1 and 2 the hit ratios continue to climb, albeit gradually, even at the end of the trace. This indicates that documents that are as old as the length of the trace (6 days for Prodigy and 5 days for UBS) were hit. Analyzing longer traces might find higher potential hit rates.

The fraction of bytes transferred saved can drop dramatically when a big document is first loaded into the cache and increase significantly when a big document is hit. These changes are seen in the Prodigy BT graph at about 250,000 and 1,000,000 entries respectively.

We do not know why the hit and refresh ratios are unusually high for the first approximately 50,000 entries in the UBS traces in Fig. 2.

To conserve bandwidth and, in many cases, to improve response time, proxy designers and operators often attempt to increase HR(-ACTUAL) and BT(-ACTUAL) by varying such parameters as the cache replacement algorithm, the cache size and the default expiration time.

Server designers and operators, on the other hand, should attempt to maximize HR(-RFC) and BT(-RFC) by minimizing the fraction that cannot be cached.

Plots in Figures 1 and 2 show the opportunity for improvement.

In Fig. 1, analyzing the Prodigy traces, the difference between HR(-RFC) and HR(-ACTUAL) is at most 6% of the requests (at the end of the trace). Therefore, given Prodigy's request and response streams, changing the proxy configuration could only slightly improve the hit rate at most 6%. As we will show in Section 5, an optimal cache replacement algorithm would need only 2 GB to cache *all* documents which would be hit, and Prodigy used 5.5 GB. Thus, it is not surprising that Prodigy's proxy served almost as many documents as possible from the cache.

By contrast, Fig. 2 shows that the performance of UBS proxy might be improved significantly⁵. Although the observed hit ratio is greater for UBS than it is for Prodigy, the proxy does not appear to perform as well as it could since the gap between HR(-RFC) and HR(-ACTUAL) is 27%.

⁵ The large gap between HR(-RFC) and HR(-ACTUAL) partly occurs because Squid 1.21 never caches documents with cookies although the HTTP/1.1 specification ([FIEL97], Section 13) lets the origin server to specify whether such documents are cacheable. Our analysis program [PEVZ98] follows RFC 2068. According to [CACE98], the analysis of an ISP traces showed that over 30% of all responses contained cookies.

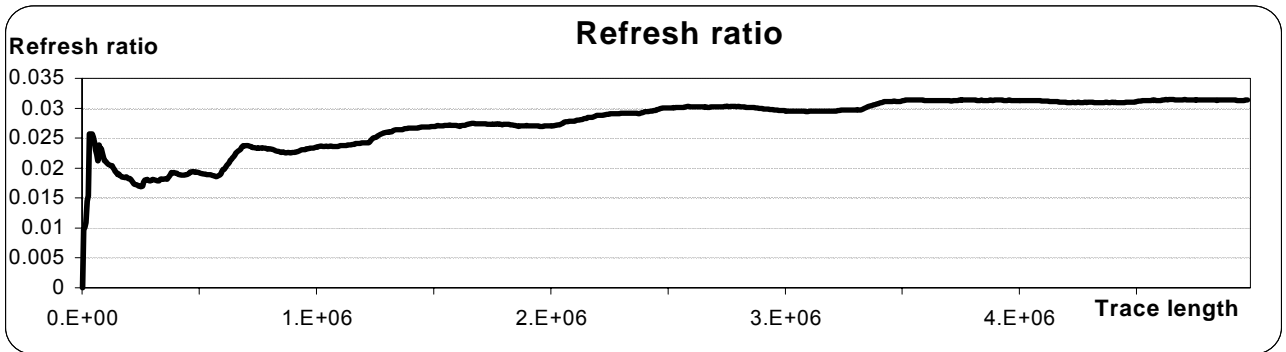
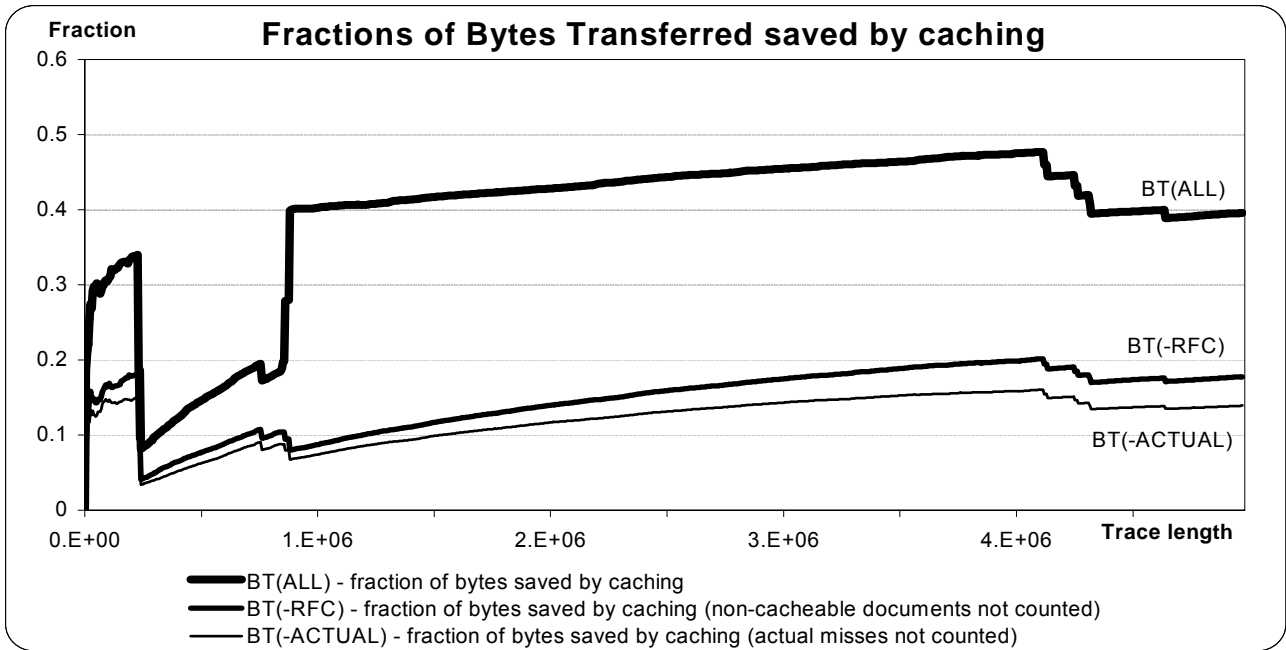
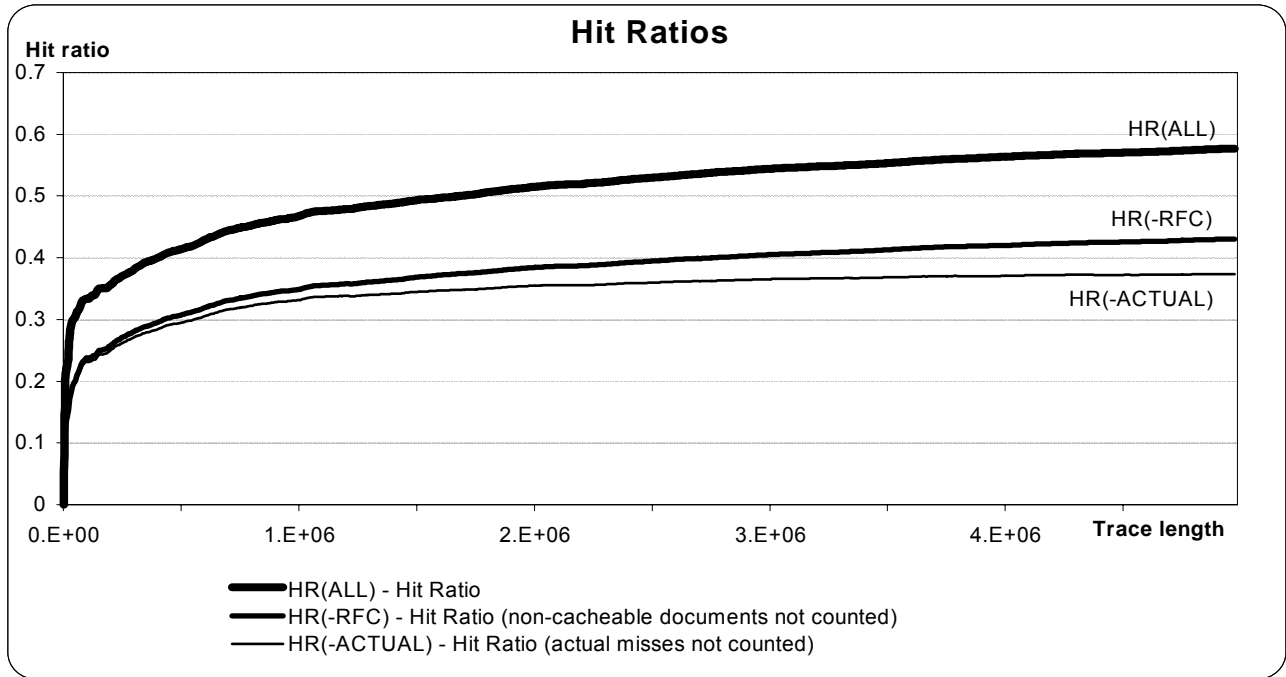


Figure 1 - Prodigy Trace

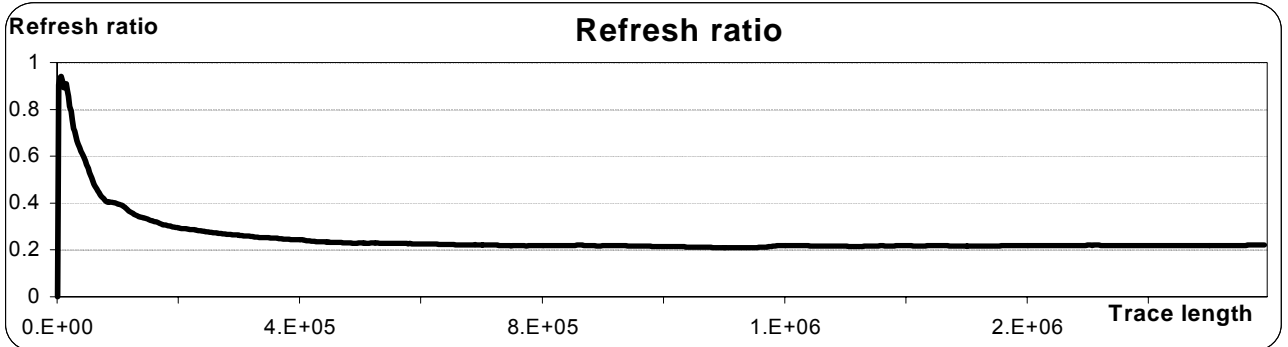
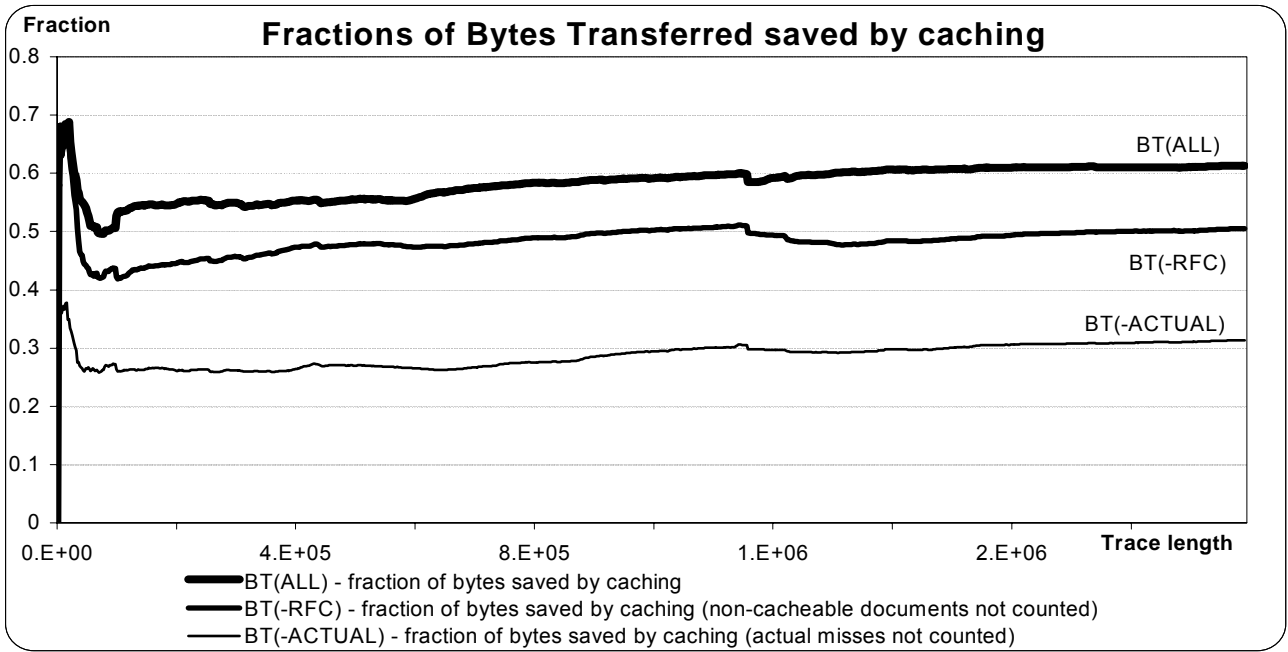
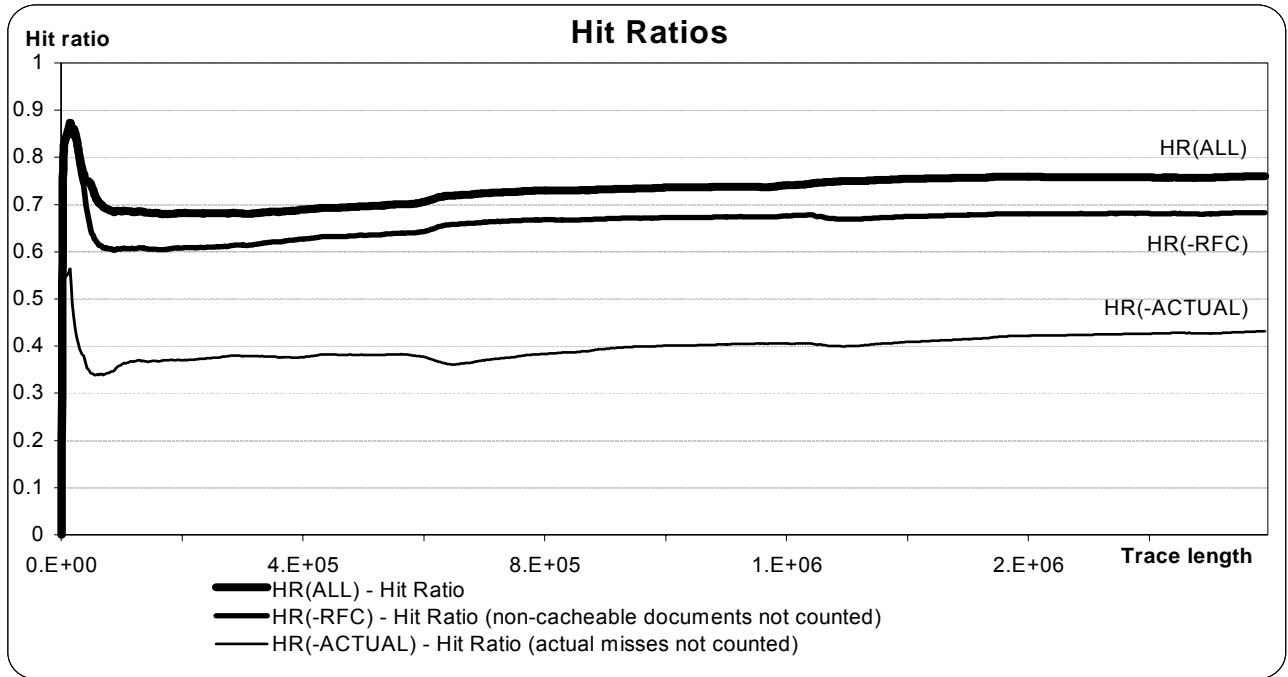


Figure 2 - UBS Trace

BT numbers are important because they determine bandwidth saving⁶, which is a primary purpose of caching. BT values are lower than HR values because small documents constitute high percentage of the retrieved documents (see the table in Section 5). Analyzing gaps between the BT(-RFC) and BT(-ACTUAL) curves leads to conclusions consistent with the analyses in the previous paragraph.

5. The Relationship Between Cache Size, Document Expiration and Hit Rate

Cache replacement algorithms in proxy servers traditionally remove older documents from a cache first, to recover space needed for current requests. Increasing the size of a cache increases the time an unreferenced document can reside in the cache without hurting performance. Many proxies allow one to limit the time a document can reside in the cache; we call this limit the *maximum residence time* (or simply residence time).

What is the relationship between the residence time and the cache size? As the cache size and the residence time are important operator tuning parameters for proxy servers, the analysis of their mutual impact deserves closer analysis. In the following case study, we perform such an analysis for a sequence of requests recorded at a real proxy server.

The Data

We prepared the Prodigy traces for analysis as follows (the UBS trace was unavailable):

- All trace entries that were marked as erroneous were removed. Errors can be caused by events such as a client interruption or a server timeout.
- The content length was not recorded if the response contained HTTP response code 304 (not modified). Those entries were updated to contain the correct content length by consulting the most recent preceding request for the same URL (15.1% of all documents). If there was no such request, the 304 log entry was removed (2.6% of all documents).
- Each entry was classified as *cacheable* if and only if the request was not explicitly marked as non-cacheable in the log. Each cacheable entry was classified as *refreshed* if it was marked accordingly in the log. About 66% of all entries

⁶ Other factors may also have a significant impact. In [CACE98], for example, it was determined that bandwidth savings can largely depend on the way proxy handles client interrupts, especially if client-to-proxy bandwidth is significantly smaller than proxy-to-origin-server bandwidth.

were marked as cacheable, and 1% were marked as refreshed.

The distribution of document sizes after our preparation is as follows:

Removed	5.0% out of 5.9 million
1-10 bytes	0.1%
11-100 bytes	3.9%
101-1000 bytes	24.0%
1 KB to 10 KB	48.0%
10 KB to 100 KB	18.0%
100 KB to 1MB	0.4%
1 MB to 10 MB	0.074%
10 MB to 100 MB	~0.002%
Greater than 100 MB	~0.0003%

Prophetic and Packrat Cache Replacement Algorithms

To understand the magnitude of the opportunity for caching we consider, for a given reference trace, the smallest cache size which guarantees the absence of cache misses as a function of the replacement algorithm and residence time.

Call a cache replacement algorithm *suitable for cachesize n and residence time E* if there are no cache misses in the reference trace, assuming a cache of size n is provided, and documents are removed from the cache after time E . The lower bound $m(E)$ of the cache size is the smallest n for which there is an (n, E) suitable cache replacement algorithm. An algorithm for which $m(E)$ is attained is called *prophetic*.

A prophetic cache replacement algorithm uses cache space as efficiently as possible and keeps a document if and only if the document will be requested again within E time. It is, of course, impossible to implement a prophetic replacement algorithm in practice.

At the other extreme from a prophetic replacement algorithm lies a strategy which keeps *all* cacheable documents, and discards them once they have not been requested for E time. This algorithm, which uses cache space inefficiently, we call the *packrat* algorithm. Let us define $M(E)$ as the smallest cache size for which a packrat algorithm suffers no cache miss, given the reference trace. It is clear that $m(E) \leq M(E)$.

The notion of *online* versus *offline* algorithms can be used to delineate the essential qualitative difference between a prophetic and a packrat algorithm:

- The request sequence in the reference trace must be entirely revealed to the prophetic algorithm. The prophetic algorithm can only work offline.
- The packrat algorithm makes a caching decision after each request; it works online.
- Both algorithms guarantee absolutely no cache misses for the reference trace, provided the cache size is large enough.

Since in practice every replacement algorithm must be online, only the packrat algorithm leads to a straightforward implementation. The prophetic algorithm represents a theoretical optimum which can only be approximated with heuristic methods.

Notice that $m(E)$ and $M(E)$ have been characterized as the minimum cache size at which the prophetic and packrat algorithm, respectively, work without cache miss. Additional space added to the cache cannot improve optimal performance; thus one might label $m(E)$ and $M(E)$ maximum useful cache size for the prophetic and packrat algorithm.

Characterization of Replacement Algorithms

Prophetic and packrat cache replacement algorithms are the extreme points of a continuum of replacement algorithms. Each replacement algorithm A can be characterized by the smallest cache size $n(A,E)$ which makes it $(n(A,E),E)$ suitable for a given reference trace. It is easy to see that $n(A,E) \geq m(E)$. Under the reasonable assumption that A does not use trivially wasteful techniques such as keeping expired documents in the cache, $M(E) \geq n(A,E)$ also holds.

$m(E)$ and $M(E)$ are computable in linear time, with a simple scan through the trace. Computing $n(A,E)$ for general A such as LRU is more complicated and not attempted in this paper.

Preparatory studies showed that the required cache size $M(E)$ for the packrat algorithm dominates the required cache size $m(E)$ for the prophetic algorithm by an order of magnitude and leads to excessive demands of disk space. We conjecture $n(A,E) \ll M(E)$ for all prevalent A and conclude that the study of the packrat algorithm is uninteresting for tuning purposes. The following discussion focuses therefore exclusively on $m(E)$.

Minimum Cache Size as a Function of Residence Time

Let E be a residence time. For the Prodigy proxy log prepared as indicated above, we computed the minimum cache size required by a prophetic cache replacement algorithm for values of E ranging from 1

to 140 hours. For each request X in the log, the prophetic algorithm caches the document requested by X if and only if X is classified as cacheable, the time interval between X and the log entry Y containing the next request for the document is less than E , and Y is not marked as refreshed.

For small values of E it is sufficient to use the log without further modification. For large values of E , the time interval during which the repetition of a request near the end of the log would make caching beneficial lies mostly outside the time interval covered by the log. To correct this situation, we assume the sequence of requests as represented in the log is repeated in a periodic manner, and analyze one period to find the minimum cache size. We conjecture that the cache size computed in this manner is a reasonable estimate for the minimum cache size if the residence time is not greater than the period covered by the log: no document that is not contained in the log is kept in the cache by the optimal replacement algorithm during that period. One may furthermore make the assumption that, on average, the request frequency for each document during that period is also representative for the period immediately before and after. The periodic log is the simplest way to avoid estimating too small a value for the minimum cache size, due to the cutoff of the log.

Figure 3 depicts the dependency of the minimum cache size on the residence time. The gigabyte range is reached once the residence time exceeds 55 hours or covers about 2.2 million entries. The dependency of the minimum cache size on the residence time is remarkably linear: 16.9 MB of additional cache size are required per additional hour residence time, according to a linear fit. (Note that for any residence time less than 6 days, or 144 hours, there must be at least 2 requests in the log for any document that makes it into the cache.)

Maximum Hit Rate as a Function of Residence Time

In the previous section, we computed the minimum cache size $m(E)$ required by a prophetic cache replacement algorithm to process a given access trace without any cache misses. The prophetic cache replacement algorithm equipped with a cache of size $m(E)$ delivers the optimal hit rate for the trace.

Figure 4 shows the optimal hit rate for the Prodigy trace as a function of residence time E . For $E \geq 20$, the benefit of extending the residence time further compared to the hit rate becomes linear. Between $E = 40$ and $E = 140$ hours the optimal hit rate increases from 40% to 45%.

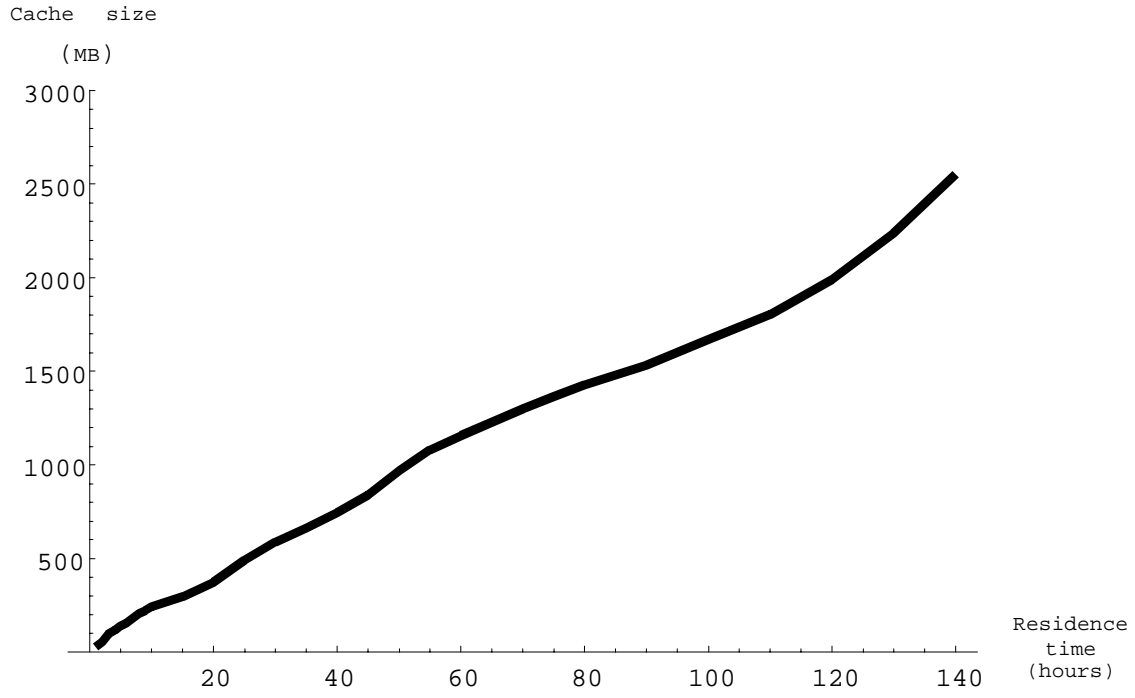


Figure 3: The minimum cache size required for a prophetic replacement algorithm, as a function of the residence time in hours for documents in the cache.

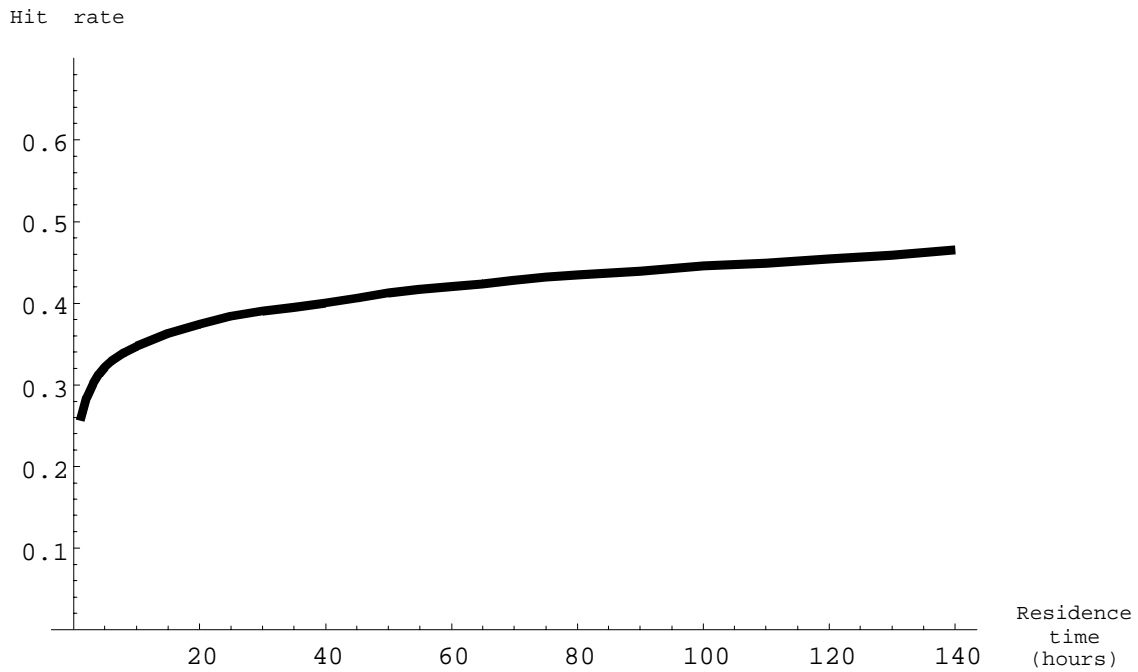


Figure 4: The maximum hit rate achieved by a prophetic replacement algorithm which removes unrequested documents after a certain number of hours. The algorithm uses the smallest cache which holds every document that would be hit before its removal. The graph shows the hit rate over all documents in the Prodigy trace. Recall that in our log, 66% of all requests refer to cacheable documents.

This analysis can also be done for documents categorized according to their size. Figure 5 shows the dependency of the hit rate on residence time for each document category. The large differences occur because larger documents are requested

much less often, and caching them is therefore less beneficial if the residence time is finite.

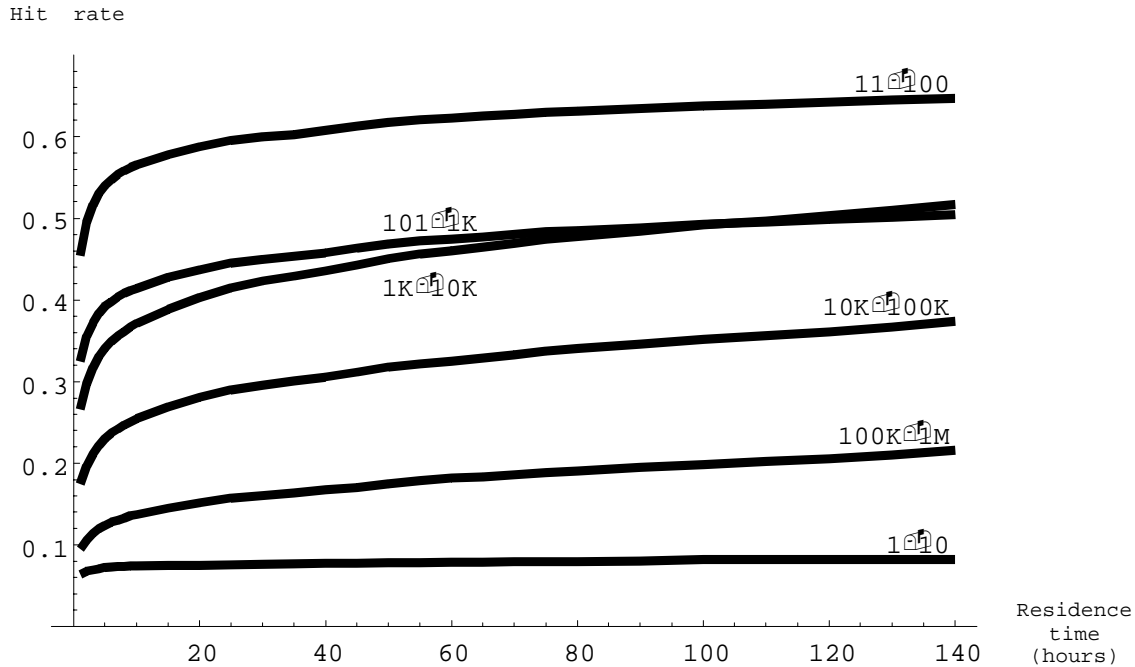


Figure 5: The maximum hit rate achieved by a “prophetic” replacement algorithm which removes unrequested documents after a certain number of hours. The hit rate is shown for each document size range. Document sizes without a suffix are in bytes. The suffix “K” stands for kilobyte, “M” stands for megabyte.

6. Related Work

Several computer scientists have analyzed Web Proxy traces [CUHN95], [GLAS94], [SEDA94], [MALT97], [BEST97]. They have measured and modeled temporal and spatial locality of reference.

Our work differs in that we focus on operational parameters such as cache size and residence time.

7. Conclusions

We analyze very long Web proxy traces from an ISP and an intranet.

We propose a new method to evaluate a proxy, which compares the actual hit rate with the potential hit rate. This method shows that the ISP proxy performs well while the intranet proxy performs poorly. The failure to cache documents with cookies may account for much of the intranet proxy’s poor performance.

An analysis of the sensitivity of the maximum achievable hit rate to the maximum allowed residence

time shows that it is important to keep the residence time above about one day. However, the maximum achievable hit rate continues to increase, albeit gradually, as residence time is increased to many days. The technique used for this analysis promises interesting results on much longer traces, extending weeks or months.

8. Acknowledgments

We thank Don DeRisi and Susan Hall at Prodigy and Mark Kennedy and Ron Gomes at UBS for access to traces, resources and information.

9. References

[ABRA95] M. Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, Edward A. Fox “Caching Proxies: Limitations and potentials”, Fourth International World-Wide Web Conference, Boston, December 1995

[BERN96] T. Berners-Lee, R. Fielding & H. Frystyk. "Hypertext Transfer Protocol -- HTTP/1.0". RFC 1945 (1996)

Second International World-Wide Web Conference, Chicago, 1994

[BEST97] Azer Bestavros "Discovering Spatial Locality in WWW Access Patterns using Data Mining of Document Clusters in Server Logs",. Tech. Rep. BUCS-TR-97-016, Boston University, Computer Science Department (1997).

[CACE98] Ramon Caceres, Fred Douglis, Anja Feldmann, Gideon Glass, Michael Rabinovich "Web Proxy Caching: The Devil is in the Details" ACM SIGMETRICS Workshop on Internet Server Performance, Madison, Wisconsin, June 1998

[CUHN95] Carlos R Cunha, Azer Bestavros, Mark E. Covella "Characteristics of WWW client based traces" Tech. Rep. BU-CS-95-010, Boston University, Computer Science Department (1995)

[FIEL97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. "Hypertext Transfer Protocol -- HTTP/1.1". RFC 2068 (1997)

[GLAS94] Steven Glassman "A Caching relay for the WWW" First International World-Wide Web Conference, Geneva (Switzerland) (1994)

[HAMI98] Martin Hamilton, Andrew Daviel "Cachebusting – cause and prevention" Internet Draft.

[LUOT94] Ari Luotonen, Kevin Altis: "World-Wide Web Proxies" Computer Networks and ISDN Systems **27 (2)**: 147-154 (1994)

[MALT97] Carlos Maltzahn, Kathy Richardson, Dirk Grunwald "Performance issues of Enterprise Level Web Proxies" ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems **25 (1)** 17-23 (1997)

[NETS97] Netscape Communications Corporation "Netscape Proxy Server Administrator's Guide Version 3.5 for Unix" (1997)

[NLAN96] NLANR "Squid Native Log Format" <http://squid.nlanr.net/Squid/features.html#accesslog>

[PEVZ98] Ilya Pevzner "Tcl implementation of trace analysis program" http://www.cs.nyu.edu/phd_students/pevzner/trace/analyze.tcl

[PEVZ98a] Ilya Pevzner "Pseudocode for computing trace characteristics" http://www.cs.nyu.edu/phd_students/pevzner/trace/fiel ds.html

[PHIL96] Phillip M. Hallam-Baker, Brian Behlendorf "Extended Log Format" W3C Working Draft WD-logfile-960221 (1996)

[SEDA94] Jeff Sedayao "Mosaic will kill my network!" – Studying Network Traffic Patterns of Mosaic Use"