The Effect of Consistency on Cache Response Time

John Dilley
Internet Systems and Applications Laboratory
HP Laboratories Palo Alto
HPL-1999-107
September, 1999

E-mail: jad@hpl.hp.com

World Wide Web, proxy cache, response time, consistency validation, distributed object consistency protocol

Caching in the World Wide Web improves response time, reduces network bandwidth demand and reduces load on origin web servers. Caches achieve these benefits by storing copies of recently requested objects near end users, avoiding future need to transfer those objects. Cached objects are usually served more quickly and do not consume external network or server resources.

Before returning an object, a cache must guess if the object it holds is still consistent with the original copy of the object. The cache may choose to validate the object's consistency with the origin server or may serve it directly to the user. If the cache must communicate with the origin server the response will take longer than one directly from cache.

This report analyzes the impact of cache consistency on the response time of client requests. The analysis divides cache responses into classes according to whether or not the cache communicated with a remote server and whether or not object data was served from the cache.

Analysis of traces from deployed proxy cache servers demonstrates that a round trip to a remote server is the dominant factor for response time. This study concludes that improving cache consistency will reduce response time and allow a cache to serve more user requests.

The Effect of Consistency on Cache Response Time

John Dilley <jad@hpl.hp.com>

Hewlett-Packard Laboratories
Palo Alto, CA

Abstract

Caching in the World Wide Web improves response time, reduces network bandwidth demand and reduces load on origin web servers. Caches achieve these benefits by storing copies of recently requested objects near end users, avoiding future need to transfer those objects. Cached objects are usually served more quickly and do not consume external network or server resources.

Before returning an object, a cache must guess if the object it holds is still consistent with the original copy of the object. The cache may choose to validate the object's consistency with the origin server or may serve it directly to the user. If the cache must communicate with the origin server the response will take longer than one directly from cache.

This report analyzes the impact of cache consistency on the response time of client requests. The analysis divides cache responses into classes according to whether or not the cache communicated with a remote server and whether or not object data was served from the cache.

Analysis of traces from deployed proxy cache servers demonstrates that a round trip to a remote server is the dominant factor for response time. This study concludes that improving cache consistency will reduce response time and allow a cache to serve more user requests.

1 Introduction

Cache servers in the World Wide Web provide a way to deliver information to end users more quickly and efficiently than serving every request directly from the origin server. Cache servers are typically placed close to a group of end users and serve all HTTP requests from those users. Requests for objects that are in the cache can be served to the user without remote communication and wide area data transfer. Requests for objects not in the cache are always resolved externally.

A cached copy of an object may differ from the current copy of that object at the origin server. This happens when a cache holds an object after the origin server changes that object. Currently origin servers do not communicate changes to caches; a cache must ask about them. Cache consistency is discussed in greater depth in [3], which proposes a new protocol to support stronger object consistency in caches.

When a request arrives for a cached object, the cache must decide whether to serve the object immediately or to validate it with the origin server. The user's request headers and the content headers may instruct the cache to validate the object. Otherwise the cache will determine whether to validate the object based upon the *freshness* of the object in cache. The cache will serve locally any object that it considers to be *fresh*, but it will attempt to check an object's consistency before serving the request if the object is *stale*. The determination of fresh and stale are made by the cache, not by the origin server. Note also that the cache has no way to know when the object actually changes, so *fresh* does not imply that the object is *consistent* with the current copy of the object.

This study explores the impact of the consistency decision on the response time of a cache, and finds that making a round trip to the origin server to validate freshness is the dominant component of the response time of most user requests.

Section 2 defines the response classes used in this analysis and describes a widely used consistency protocol that is used by a cache to identify whether an object is fresh or stale. Using these definitions Section 3 analyzes data from three cache servers, and Section 4 summarizes the findings. Three appendices present supporting detail from the log file analysis.

2 Cache Consistency

This analysis classified responses according to how the cache behaved: whether it served the data or just a validation of freshness, and whether or not it made a consistency check. The analysis measured the response time of each of the following classes of responses.

- *Fast hits*, where the cache returns object data to the requestor without remote communication. The cache considers the object to be fresh and serves it directly.
- *Fast validations*, where the cache returns only a validation of freshness to the requestor (HTTP code **`304 Not Modified`**), who presumably already has a copy of the data. The cache considers the object to be fresh without remote communication.
- *Slow validations*, where the cache returns a validation of freshness after contacting the origin server and learning that the cached copy is consistent with the original version of the object.

- *Slow hits*, where the cache returns object data to the requestor after validating it with the origin server. In this case the client does not have object data but the cache does and the cache determines it is consistent by contacting the origin server. (This type is also referred to as a *slow validation with data*, with label *sdat* in the graphs.)
- *Consistency misses*, where the cache returns a new copy of the object after contacting the origin server and getting a fresh copy of the object. The cache had the data but considered it to be stale and so made a consistency check, which determined that the object had been modified.
- *Regular miss*, where the cache returns an object that was not in cache after contacting the origin server to retrieve a copy of that object. The analysis does not distinguish between cold misses (the first request for an object) and capacity misses (where the object had formerly been in cache but had been evicted by the cache replacement policy).
- *Direct*, where the cache determines an object is not cacheable through configuration (certain types are declared to be non-cacheable) or through server response headers (such as **`Pragma: no-cache`**). The request is sent to the origin server and the response data is relayed to the client. The cache does not keep a copy of the object.

The key comparisons in this analysis are between a fast validation and slow validation, and between a fast hit and a slow hit. In each of these cases the difference is the consistency check from the cache to the origin server. The validation case returns only HTTP headers from the cache to the client; the fast hit and slow hit cases return object data.

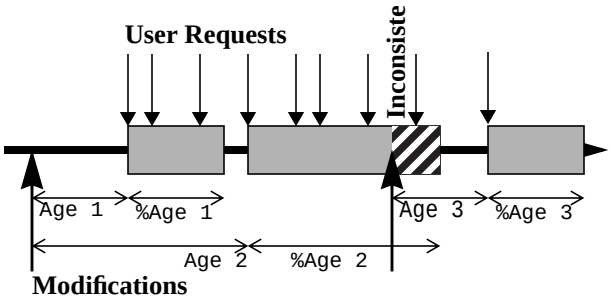
Appendix A defines the method that determined cache response type from the data in cache log files.

2.1 HTTP Consistency Validation

One mechanism to determine object freshness is the "adaptive Time To Live" (TTL) algorithm from the Alex file system [2]. The Alex protocol is described below and depicted in Figure 1. The Alex protocol has been shown to be effective in practice and is widely used by cache implementations. The definition of fresh and stale content are determined by such a protocol as follows.

- Upon first retrieval of an object the object's modification time is noted. This is a cold miss.
- The time between the modification time and the current time defines the object's *age*. This value is shown as Age 1 in Figure 1.
- The cache computes a percentage of the object's age defines that as the time to live (TTL). During the TTL period the object is considered *fresh* and will be served from cache. This first TTL period is % Age 1 in Figure 1. Requests for an object within its TTL period will result in one of two responses.
  - If the requestor has a copy of the object and makes an IMS request the response is a fast validation from the cache.
  - If the requestor does not have the object and makes a regular GET request without an IMS header the cache immediately sends the object data. This is a fast hit.
- After the TTL expires the object in cache is considered *stale*. Upon the next request the cache will make an HTTP GET request with an **`If-Modified-Since`** (IMS) header to the origin server to determine if the object has been modified. The origin server will respond to the cache in one of three ways.
  - If the object has not changed the origin server will reply with the HTTP status code **`304 Not Modified`**. This results in a slow validation if the requestor had object data, a slow hit if not.
  - If the object has changed a new copy will be returned with HTTP status code **`200 OK`**. This is a consistency miss.
  - The origin server may fail to respond or may respond with an error, such as to indicate that the object does not exist on that server.
- After the slow validation check in Figure 1 a new object age (Age 2) and TTL period (% Age 2) are calculated. During this second TTL interval a second modification is shown. A request to the cache after this modification will result in an *inconsistent* reply, where data from cache is different from data on the origin server. The cache and user only become aware of the new data after the TTL period expires or if the user forces a reload of the object.
- On the next retrieval after the second TTL expires a new copy of the object is retrieved (a consistency miss), a new age is computed from its last modification time (Age 3), and a new TTL period is computed as a percent of this new age (% Age 3)

FIGURE 1. The Alex Protocol



A cache can only validate objects if the origin server and cache implement enough of the HTTP specification. The origin server must supply the object's **`Last-Modified`** time for the cache to validate the object. There are other mechanisms for checking object consistency in HTTP, such as HTTP/1.1 **`Etags`** and **`Cache-Control`**, but the **`Last-Modified`** approach is the most widely used.

Furthermore, if an object has an **`Expires`** header the cache will consider the object to be fresh until the time specified in the **`Expires`** header, after which it is stale. Note that the object will be served inconsistently from a cache if it changes while still considered fresh.

2.2 Analysis Methodology and Limitations

Cache log files contain information about the response time of user requests, as well as other information. These logs include information about each request, including the object URL, the object size in bytes, the service time at the cache, whether object data was returned to the client, and whether the cache made a consistency check with a remote server.

This analysis used cache log information to identify response time and cache behavior. The mapping of log fields to cache behavior is described in Appendix A.

There are some limitations with such a log-based analysis scheme:

- Cache logs do not indicate when the client received and displayed the response in the browser application, and therefore can not determine client response time. They record only the request residence time at the cache server (request service time).

  Service time is related to user response time; clearly response time can be no less. It is also related to cache throughput: a long response time at the cache consumes system resources for a longer time, making them unavailable for other requests.

- The logs indicate only how long the cache application took to service the request. They do not include processing time of the operating system kernel or network card on the cache system. The cache believes its task is complete when the write of the last byte returns, but the task is not actually complete for the cache until the OS kernel has successfully transmitted all response bytes to the client, received an acknowledgment, and closed the connection.

- The logs can not identify when objects are served from the browser's cache. These objects would likely be displayed much more quickly than requests to the cache, just as local cache responses are faster than responses requiring remote communication. The logs do identify when the browser validates an object: it sends a GET IMS request for the object to the cache.

- The logs do not indicate whether external requests are served by other parent proxies or by the origin server. The aggregate response time of external caches and origin servers is nevertheless useful, since this determines perceived user response time.

- The logs do not always enable us to identify all of the response classes described earlier.

- The users of the caches we studied were connected to high speed networks. The response time to users on slow modems would be substantially different. Also the logs we were able to obtain were from LAN and cable modem users in the United States. Results from significant-duration cache traces from other geographical regions would likely show more dramatic differences in response times.

We attempted to obtain European and Asian cache logs; either the logs were not available or they did not contain sufficient information to perform our analysis. We are still interested in analyzing non-US cache logs. Please contact the author if you have logs that can contribute to this study.

3 Servers Studied

The analysis is based upon the study of log files from three web cache servers.

- **`granite.hpl.hp.com`**, January 1998 - April 1999. This is a Squid server within our department. During the period there were 3.3 million cacheable requests.
- **`proxy.hpl.hp.com`**, April 1999. This Netscape cache serves HPL Palo Alto. During the period there were 11.4 million cacheable requests.
- **`cable-modem`** site, one week of June 1997. This Netscape cache served a residential cable modem deployment. During the week there were 8.2 million cacheable requests.

The responses from these cache servers are evaluated in the remainder of this section. The responses are presented in tables using attributes described in Table 1.

Table 1. Response attributes

Attribute	Description
Response Type	The response type as described in "Classes of Responses" above
Percent of Responses	The fraction of all HTTP GET responses that were of the class
Response Time	The mean response time for the class
Response Size	The mean response size in K bytes
Percentiles	The median (50th) and 80th percentile response times for the class

In a web workload, mean response time (and response size) is often skewed by a relatively small number of long duration (or large) responses. For this reason the median response time, which indicates what most responses see, is often much less than the mean. Sometimes 80% or more of the responses are satisfied in less than the mean response time. Since the median is not skewed by a few large (or long) responses it may be a better indicator of response time than the mean. For this reason this analysis includes the mean as well as the 50th percentile (the median) and the 80th percentile responses.

The analysis excludes non-HTTP, error, and other response types, so the sum of the "Percent of Responses" may not total 100%.

3.1 Squid on granite.hpl.hp.com

The first server studied was our workgroup's cache server. It serves requests from a local population of researchers. It is connected to the external network through a second Squid proxy (parent) that does not cache objects; the parent acts as a firewall proxy.

Sixteen months of responses were analyzed, consisting of 3.3 million cacheable requests.

Table 2. granite.hpl.hp.com responses

Response Type	Pct	Mean Size (KB)	Response Time (sec)		
			mean	50th	80th
Fast val (fval)	12.7	0.196	0.088	0.025	0.098
Slow val (sval)	4.3	0.124	0.769	0.263	0.589
Fast hit (fhit)	12.6	4.986	0.218	0.041	0.141
Slow hit (sdat)	13.0	4.871	0.776	0.229	0.501
Cons miss (cons)	4.5	8.242	1.910	0.437	1.148
Miss (miss)	49.2	16.440	2.332	0.501	1.514

Table 2 shows that the mean response time for slow validations is approximately eight times longer than fast validations, and that both transfer about the same amount of data to the client. The median response time for slow validations is about 10 times longer then the median response time for fast validations. The response size is that of the HTTP response headers. No object data is returned to the user for a validation.

The mean response time for slow hits is 3.5 times that for fast hits; the median is 5.5 times. Both of these response classes return object data to the client of about five KB. Table 2 also shows that consistency misses take approximately nine times longer than fast hits and twice as long as slow hits; the median time for consistency misses is seven times longer than fast hits. In these cases object data is returned to the user, but fast hits require no remote communication; slow hits receive only a small validation from the origin server; consistency misses retrieve full object data from the remote server. The data transfer size, wide area round trip, and server demand all affect response times.

Note also that consistency misses are considerably larger than fast hits and slow hits. Most of the consistency misses were for HTML objects. The increased response time is due both to larger mean HTML object size (see Table 14, "Most Bytes Transferred by Type," on page 9), and the complexity of generating dynamic HTML objects. The log-based analysis could not determine when HTML objects were dynamically generated but we know from experience that some of them are.

The mean response times by class in Table 2 are closer to the 80th percentile value than to the median value for the class. In some cases the mean is larger than the 80th percentile. This indicates a heavy-tailed distribution where a few very long transfers skew the mean such that it no longer corresponds to the response time of the majority of requests.

Cold misses are slower than consistency misses, but the mean object size is again much larger. This is due to a few very large cold misses. The granite cache did not keep any objects in cache over 4 MB, so each response over 4 MB is necessarily a capacity miss.

The median and 80th percentile object sizes for consistency misses and cold misses are nearly equal, as shown in Table 3; it is only at the 90th percentile and above that cold misses have a distinctly heavy tail relative to other response types. This is likely due to a few large objects being requested through the cache. The 90th percentile value indicates that 10% of misses from this server were larger than 27 KB.

Table 3. Squid response size (KB) - percentiles

Response Type	Median	80th pct	90th pct
Consistency miss	3.018	12.015	19.485
Miss	2.882	12.580	27.523

Figure 2 shows the distribution of response times observed at this server. Figure 3 shows the distribution of response sizes. These graphs present a cumulative distribution function (CDF), which indicates on the y axis the percentage of responses less than or equal to the time (or size) on the x axis. With a CDF the median value is the x value at which a curve crosses 50%.

FIGURE 2. granite.hpl.hp.com response time CDF

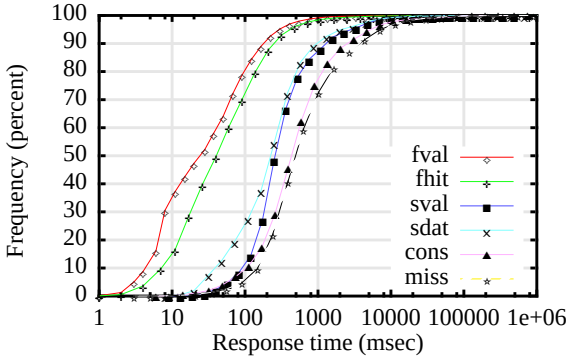


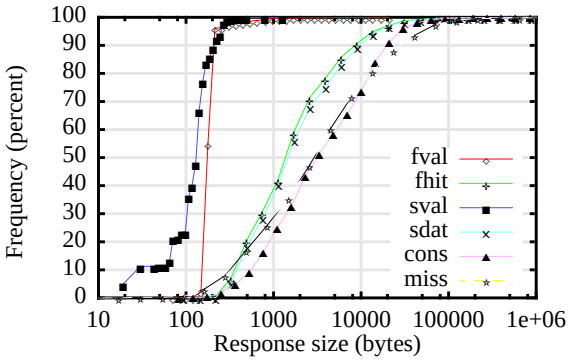
Figure 2 shows that fast validations and fast hits are significantly faster to complete than the other response types, and that misses are the slowest response type. 70% of fast hits and 80% of fast validations complete in under 100 msec. Fewer than 10% of slow validations complete in under 100 msec.

The response size distribution for fast validations is in a very small range, indicated by a nearly vertical line in the CDF in Figure 3. The response size is determined by

the headers sent by the cache, which are very similar for every response. Slow validation headers come from various origin servers, and show greater variability and smaller overall size. The Squid cache evidently includes more header information than most origin servers do.

The file size distribution also shows that response sizes for fast hits and slow hits are closely matched. This eliminates size variation as a likely cause for the difference in response times.

FIGURE 3. granite.hpl.hp.com response size CDF



More detail about hit and validation response times and sizes types is presented in Appendix C as a histogram and CDF for each response type.

3.2 Netscape on proxy.hpl.hp.com

The next server studied was the HP Laboratories Palo Alto external Netscape cache server. It serves requests from local users and other cache servers. It is connected to the external network through a packet filtering firewall. It does not use a parent cache. One month of traffic included 11.4 million cacheable requests.

Table 4. proxy.hpl.hp.com responses

Response Type	Pct	Mean Size (KB)	Response Time (sec)		
			mean	50th	80th
Fast val	13.9	0.00[a]	0.071	0.032	0.072
Slow val	9.6	0.00[a]	0.993	0.447	0.661
Fast hit	13.9	6.194	0.159	0.051	0.117
Slow hit	5.9	5.249	0.908	0.468	0.692
Cons miss	1.25	9.427	1.355	0.724	1.202
Miss	26.4	768.38	2.130	0.794	1.413
Direct	28.4	331.75	1.926	n/a	n/a

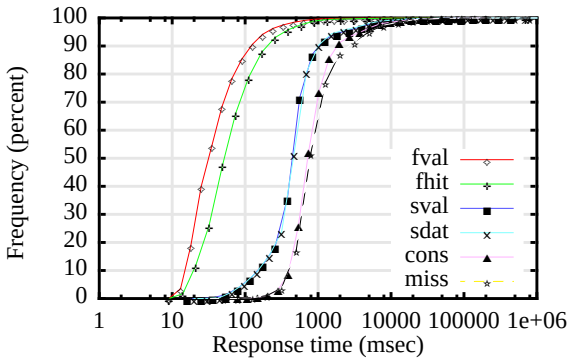
a. Netscape did not report HTTP header size

Table 4 indicates that fast validations were more than ten times faster than slow validations: about 14 times faster in the mean and median; and about nine times faster at the 80th (and 90th) percentiles. In these cases the reported data size was zero bytes; the Netscape cache did not report header size in the access log.

The response time for slow hits is about six times longer than fast hits at the mean, nine times at the median.

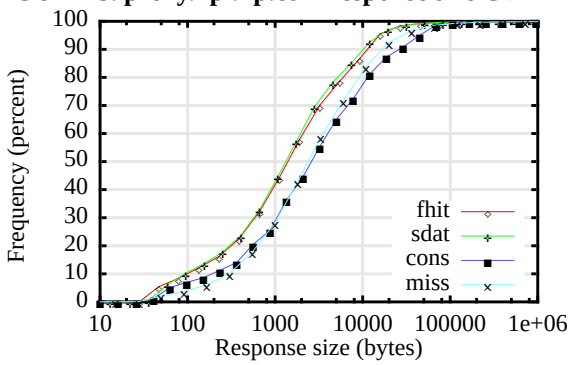
Note the large response size for misses. Some very large objects were served through this cache, which significantly affected the mean transfer size. The response size cumulative distribution shows that there were very few of these objects: only 5% of objects were over 27 KB.

FIGURE 4. proxy.hpl.hp.com response time CDF



The response time distribution for this server shows that fast validations and fast hits are significantly faster than the other response types. Furthermore slow validations and slow hits take almost exactly the same amount of time. This indicates that the round trip to the remote server is the dominant component in cache response time, not the amount of data transferred to clients. Consistency misses and regular misses are slower than slow validations and slow hits.

FIGURE 5. proxy.hpl.hp.com response size CDF



The response size distribution shows that fast hits and slow hits return similar amounts of data. Consistency misses and regular misses are also closely matched. Validations (fast and slow) were reported as zero bytes and do not appear on this graph.

3.3 Cable modem site

The third server studied was a Netscape cache server at a cable company acting as an ISP for residential users connected to the Internet through high-speed cable

modems. This cache was connected to the external network through a packet filtering firewall without a parent proxy. One week of traffic was analyzed for this report, consisting of 8.4 million cacheable requests.

The workload of the users at this site during the entire five month log collection period was studied in detail by Arlitt et al [1]. This report extends that characterization to examine cache response time by cache behavior.

The logs from this site did not contain enough detail to differentiate between consistency, proxy-only, and regular misses. Table 5 summarizes the results from this site.

Table 5. Cable modem cache, all responses

Response Type	Pct	Mean Size (KB)	Response Time (sec)		
			mean	50th	80th
Fast val	9.24	0.00	0.037	0.013	0.030
Slow val	10.1	0.00	0.717	0.191	0.380
Fast hit	15.0	11.188	0.952	0.028	0.049
Slow hit	11.1	9.912	0.976	0.214	0.437
Miss	50.8	17.034	2.582	0.562	1.479

This data set confirms the LAN findings of fast validations being more than an order of magnitude faster than slow validations. Both the mean and the median response time for slow validations indicate that they take between 15 and 20 times longer to complete than fast validations.

The mean response time for slow hits is about the same as fast hits, but this is evidently due to a few very long fast hits: the median and 80th percentile response times show that slow hits take eight to nine times as long to complete as fast hits. In Table 5 the mean and median response time for misses is about 160% longer than for slow hits; the mean response time for misses is 170% longer than fast hits. However, the median response time for misses is 20 times longer than fast hits.

The data size does not appear to be a dominant factor in response time, but distance does. Table 6 presents mean, median, and 80th percentile response size for fast hits, slow hits, and misses.

Table 6. Cable modem cache size distribution (KB)

Response Type	Mean	50th	80th
Fast hit	11.188	2.343	8.506
Slow hit	9.912	2.510	8.906
Miss	17.034	4.263	13.174

The response size of fast hits is 10% larger on average than slow hits, but slightly smaller at the median and 80th percentiles. The mean response size for misses is 72% larger than for slow hits, the median is 70% larger.

4 Summary and Conclusions

Caching of objects in web cache servers near end users can significantly reduce object retrieval time. When an object is in cache it takes approximately half the time to validate and return the object to the requester than when an object is not in cache and must be retrieved. This corroborates earlier research in caching, and supports the deployment of web cache servers.

In this report, we have further demonstrated that performing a consistency check prior to returning an object from cache has a significant impact on object response time. When a validation is returned from cache directly it is returned approximately an order of magnitude faster than an object that requires a consistency check. Reducing the service time of requests will allow a cache to support more total user requests.

Based upon these results we suggest that there is an opportunity to improve user response time from caches by improving the cache consistency mechanism. We have developed such a mechanism based on server invalidation and shown that it provides better object consistency, is faster for end users, consumes slightly less network bandwidth, and reduces origin server load [3]. As faster end systems and residential networks are deployed consistency validation mechanism will have more relative impact on the performance of user requests and caches.

A page-based response time analysis is underway. The cache logs include some information about page composition and cache hit rates. From this data and assumptions about object modification and access patterns we can derive an analytic model of page composition and consistency. Using this model we can estimate cache response time based upon the consistency protocol.

5 Acknowledgments

We are grateful for the time and effort of Tai Jin who supplied logs from the local Squid cache and provided insight into its operation, and Mike Rodriquez of HPL Research Computing Services who ran our scripts on the HPL logs. Thanks to Rich Friedrich for suggesting the work, and for his review comments. Thanks also to Martin Arlitt, John Barton, Ilja Bedner, Radhika Malpani, Stéphane Perret, Jerry Rolia, and Anna Zara for their helpful comments and feedback.

6 References

[1] M. Arlitt, R. Friedrich, T. Jin, Workload Characterization of a Web Proxy in a Cable Modem Environment, in ACM SIGMETRICS Performance Evaluation Review, vol 27 no 2, pp25-36, August 1998.

[2] V. Cate. “Alex -- A Global Filesystem”. In Proceedings of the USENIX File System Workshop, pages 1--12, May 1992. USENIX Association.

[3] J. Dilley, M. Arlitt, S. Perret, T. Jin, “The Distributed Object Consistency Protocol”, Technical Report HPL-1999-109, Hewlett-Packard Laboratories, Sept. 1999.

Appendix A: Log File Syntax

This appendix describes the syntax of the cache access log files and the patterns that correspond to each of the response types discussed in this report.

In the access log files, each request that was served by the cache is indicated by a variable-width line of information with a fixed number of fields. To analyze the cache behavior we parsed the logs and examined the values of each of the fields. The following sub-sections present the format, contents, and analysis method used for each of the three cache servers we studied.

Squid/HPL

The Squid 1.1.20 server running on granite.hpl.hp.com logged the following information. The Squid log format is described in the Squid FAQ, particularly in Section 6.

Table 7. Squid log fields (granite.hpl.hp.com)

Log field	Translation of the field code.
date	Date and time of the request.
xfer-time-msec	Transfer time for the request, cache to client.
ip	IP address of the requestor.
squid-status	Result code from Squid. A translation of field codes can be found in the Squid FAQ, section 6.6.
remote-status	HTTP status code returned to the client. HTTP status codes can be also be found in the Squid FAQ, section 6.7.
content-size	Size of the content returned to the client, including headers.
request-type	HTTP request type (GET, HEAD, POST).
URL	URL of the remote resource being accessed including parameters.
user	Local user who made the request (always “-”).
route	Route which the request took, indicating whether it was served locally or was routed to a parent cache.
mime-type	The content MIME type.

We used the squid-status and remote-status fields to identify the Squid cache behavior. Table 8 lists the combination of squid-status and remote-status values that corresponded to each of the categories in our analysis.

Table 8. Response type by Squid field code

Response Type	squid-status/remote-status values
Fast val	TCP_HIT/304 TCP_MEM_HIT/304 TCP_IMS_HIT/304
Fast hit	TCP_HIT/200 TCP_MEM_HIT/200 TCP_IMS_HIT/200
Slow val	TCP_REFRESH_HIT/(302|304) TCP_CLIENT_REFRESH/(302|304)
Cons miss	TCP_REFRESH_MISS/(200|40*) TCP_CLIENT_REFRESH/(200|40*)
Miss	TCP_MISS/* TCP_CLIENT_REFRESH/206

Netscape/HPL

The Netscape Proxy Server on proxy.hpl.hp.com logged the following information. The Netscape log format is described in the Netscape Proxy Server documentation.

Table 9. Netscape log fields (proxy.hpl.hp.com)

Log field	Translation of the field code.
ip	IP address of the client making the request.
user	User who made the request (always “-”).
date	Date and time of the request.
URL	URL of the object, including parameters.
clf-status	Status returned from the proxy to the client.
p2c-cl	Proxy to client content length.
remote-status	Status returned from the origin to the proxy
r2p-cl	Remote to proxy content length.
content-length	Value of the content-length field in the HTTP headers.
p2r-cl	Proxy to remote (POST) content length.
c2p-hl	Client to proxy header length.
p2c-hl	Proxy to client header length.
p2r-hl	Proxy to remote origin server header length.
r2p-hl	Remote to proxy header length.
xfer-time-total	Total transfer time for the request (seconds.msec).
actual-route	Route which the request took (SOCKS, DIRECT, NONE).
cli-status	Client status, indicates if the client aborted.
svr-status	Remote server status.
cch-status	Cache status, indicates if data was fetched to disk, object was not cacheable, and so on.

For the Netscape cache we used the clf-status, remote-status, route, and cch-status fields to identify cache behavior. Table 10 lists the combination of clf-status, remote-status, route, and cch-status that corresponded to each of the categories in our analysis. In the table, a "-" indicates that no value was present in this field for the request; a "*" indicates that any value is accepted for this field.

Table 10. Response type by Netscape field code

Resp Type	clf-stat	rmt-status	route	cch-status
Fast val	304	-	-	NO-CHECK
Fast hit	200	-	*	NO-CHECK
Slow val	304	304	*	UP-TO-DATE
Slow hit	200	304	*	UP-TO-DATE
Cons miss	200	200	*	REFRESHED
Miss	200	200	*	WRITTEN
Direct	*	*	*	DO-NOT-CACHE NON-CACHEABLE

Netscape/Cable Modem

The Netscape server at the cable ISP site was configured differently from the local Netscape cache. In particular, a key piece of information was missing from the logs: cch-status. Without this information it was not possible to determine if an object was already in cache in the case of a consistency miss (when the cch-status would have been REFRESHED), nor if the response was direct (proxy only, when cch-status would indicate DO-NOT-CACHE or NON-CACHEABLE).

The log did have a few additional fields, in particular to record DNS lookup time, but we did not use this information in our analysis. It had some other response time metrics for some Netscape internal states (cwait, iwait, fwait), but these values were always zero.

Table 11 lists the log fields that were present in the logs.

Table 12 lists the combination of clf-status and remote-status that corresponded to each of the categories in our analysis of this server. In Table 12 a "-" indicates that no value was present for this field for the request.

Table 11. Netscape log fields (cable modem cache)

Log field	Translation of the field code.
ip	IP address of the client making the request.
user	User who made the request (always "-").
date	Date and time of the request.
URL	URL of the object, including parameters.
clf-status	Status returned from the proxy to the client.
p2c-cl	Proxy to client content length.
user-agent	The browser version string from the client.
remote-status	Status returned from the origin to the proxy.
r2p-cl	Remote to proxy content length.
content-length	Value of the content-length field in the HTTP headers.
p2r-cl	Proxy to remote (POST) content length.
c2p-hl	Client to proxy header length.
p2c-hl	Proxy to client header length.
p2r-hl	Proxy to remote origin server header length.
r2p-hl	Remote to proxy header length.
xfer-time-total	Total transfer time for the request (seconds.msec).
xfer-time-dns	Total time to lookup remote IP address with DNS.
xfer-time-cwait	Time in cwait state (was always zero).
xfer-time-iwait	Time in iwait state (was always zero).
xfer-time-fwait	Time in fwait state (was always zero).

Table 12. Response type by Netscape field code (cable modem proxy)

Response Type	clf-status	remote-status
Fast val	304	-
Fast hit	200	-
Slow val	304	304
Slow hit	200	304
Miss	200	200

Appendix B: Large File Analysis

The largest objects observed in the logs are multimedia objects (.mpg, .mp3, .mov, .ra, .rm), Microsoft Office documents (.ppt, .doc, .xls), compressed objects (.zip, .gz), applications (.exe), and print-quality documents (.ps, .pdf). The following file analysis is based upon one day's data from the HPL Netscape cache.

Table 13 examines the largest file classes. In each case these object types accounted for less than 0.2% of the total requests, yet they were responsible for a significant fraction of the bytes transferred. Table 14 presents the top classes by total demand.

Table 13. Largest Object Types

Object Type	Mean Size (KB)	Total (MB)	% Bytes
MPEG	7,394	332	16.1
MP3	3,412	37	1.78
QuickTime Movie	1,878	59	2.85
PowerPoint Presentation	1,664	84	4.10
RealMedia Audio/Video	1,465	36	1.88
Microsoft Word document	903	56	2.70
Microsoft Excel spreadsheet	782	18	0.85
PostScript + PDF documents	393	118	5.75
ZIP (compressed) archive	566	49	2.42

Table 14. Most Bytes Transferred by Type

Object Class	% Reqs	Total Demand (MB)	Mean Size (KB)	% Bytes
GIF	55.0	353	4.2	17.2
MPEG	0.03	332	7,394	16.1
JPEG	16.7	326	12.6	15.8
HTML	14.8	245	10.8	11.9
PDF + PS	0.19	118	393	5.75
EXE	0.26	101	255	4.94
PPT	0.03	84	1,663	4.10
ASP	4.13	65	10.4	3.19
MOV	0.02	59	1,878	2.85

GIF images are responsible for the most data transfer into the cache, but only by a small margin over MPEG video objects. 30 in 1 million requests are for MPEG objects, but one byte in seven is from an MPEG object.

Most caches will not hold large objects, preferring instead to cache many smaller objects to improve object hit rate. Caching one average sized MPEG object would evict 1,760 average size GIF images. The decision a cache administrator must make is whether to optimize cache response time (by holding many smaller objects) or external bandwidth demand (by holding some larger popular objects).

One implication of this is that caches should be partitioned according to the type of object they will be storing, to allow some very large objects to be stored near the end users requesting them, but not everywhere. This is an active field of research that we continue to follow.

Appendix C: Response Analysis

The Cumulative Distribution Function plots convey information about the response time or size distribution across all classes for a population of responses. It is a fairly concise way to represent the data.

CFD plots do not convey well the characteristics of each class of responses. This appendix provides additional detail for the key response classes.

The images on the following pages illustrate the specific response time and response size for fast validations and slow validations, and fast hits and slow hits. This page contains graphs for the proxy.hpl.hp.com (proxy) server, the next page for the granite.hpl.hp.com (granite) server.

Fast and slow validation response size was reported as zero bytes by the Netscape cache proxy.hpl.hp.com, and are not graphed.

FIGURE 6. proxy: Fast Hit Response Size

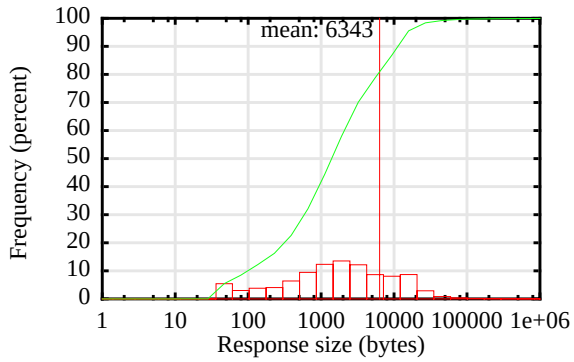


FIGURE 7. proxy: Slow Hit Response Size

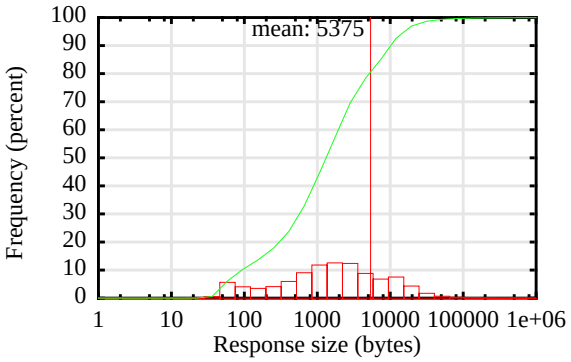


FIGURE 8. proxy: Fast Validation Response Time

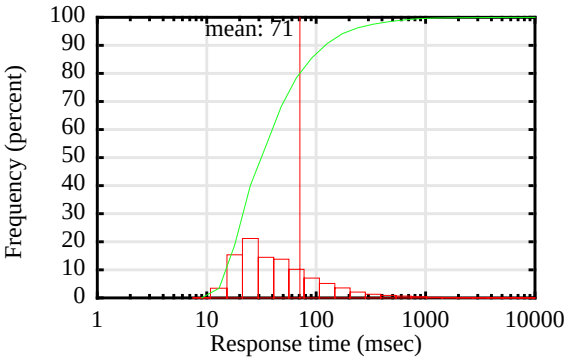


FIGURE 9. proxy: Slow Validation Response Time

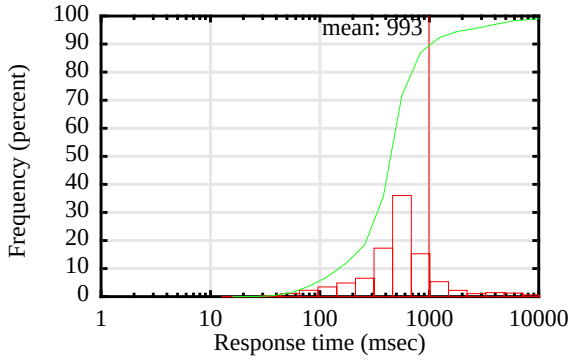


FIGURE 10. proxy: Fast Hit Response Time

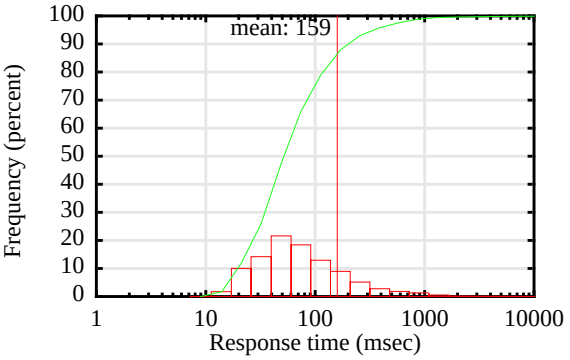


FIGURE 11. proxy: Slow Hit Response Time

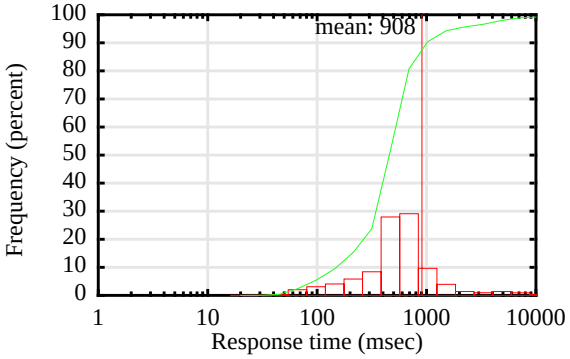


FIGURE 12. granite: Fast Validation Response Size

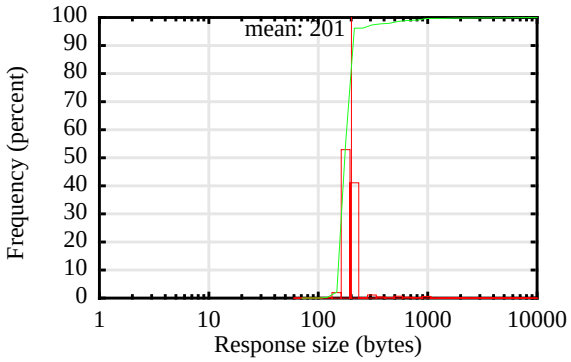


FIGURE 13. granite: Slow Validation Response Size

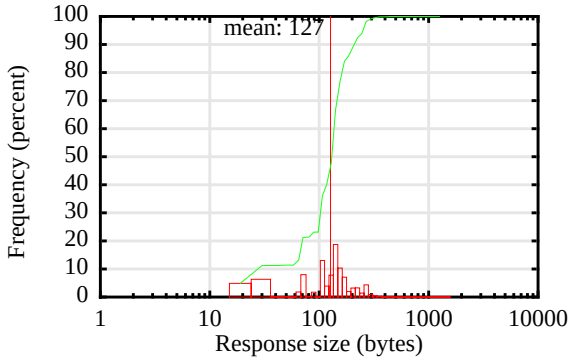


FIGURE 14. granite: Fast Hit Response Size

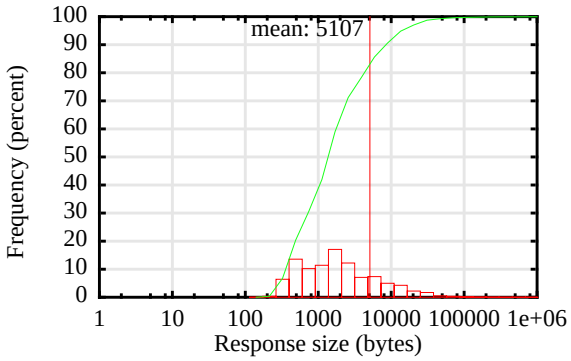


FIGURE 15. granite: Slow Hit Response Size

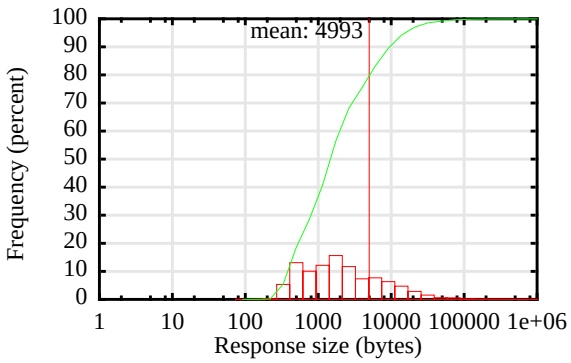


FIGURE 16. granite: Fast Validation Response Time

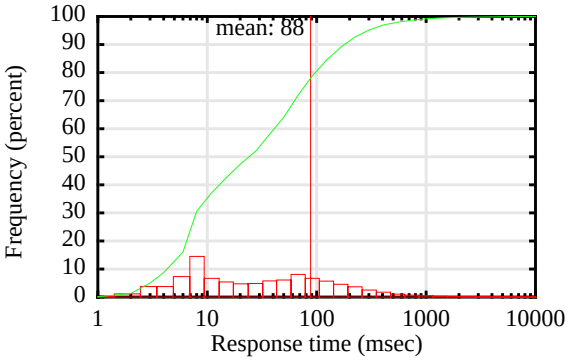


FIGURE 17. granite:Slow Validation Response Time

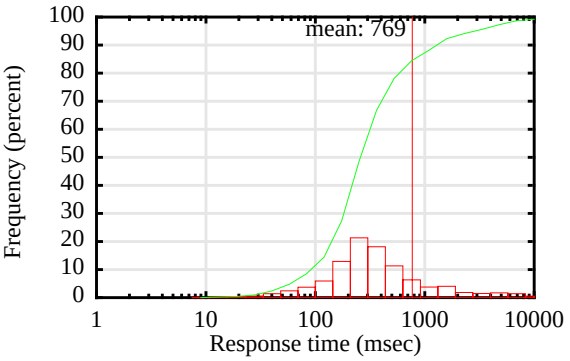


FIGURE 18. granite: Fast Hit Response Time

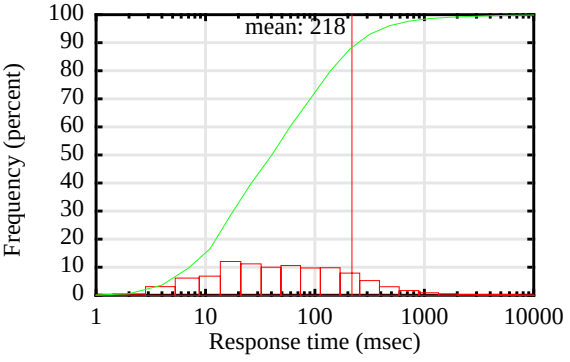


FIGURE 19. granite: Slow Hit Response Time