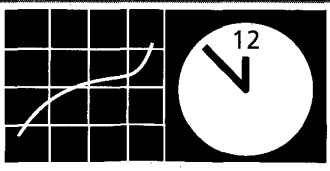


The Effect of Consistency on Cache Response Time

John Dilley, Hewlett Packard Laboratories

Abstract

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.

Proxy cache servers in the World Wide Web deliver information to end users more quickly and efficiently than serving every request directly from the origin server. They can improve response time, reduce network demand, and lighten the load on origin Web servers.

Proxy caches (or simply caches) achieve these benefits by storing copies of recently requested objects, avoiding the need to transfer those objects again. Cached objects are usually served more quickly and do not consume external network or server resources. Cache servers are typically placed close to a group of end users and handle 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 explored in [1, 2]. In [3] we propose an invalidation-based protocol motivated by Cao's work [2] to support stronger cache consistency.

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 in cache may instruct the cache to validate the object. Otherwise, the cache will determine whether to validate the object based on the *freshness* of the object in cache. The cache will serve locally any object it considers to be *fresh*, but will attempt to check an object's consistency before serving the request if the object is *stale*. The determination of fresh or stale is made by the cache, not 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.

We define the response classes used in this analysis and describe a widely used consistency protocol that is used by a cache to identify whether an object is fresh or stale. Using these definitions, we then analyze data from three cache servers and summarize the findings.

Cache Consistency

This report classifies responses according to cache behavior: whether it serves the data or just validates freshness, and whether or not it makes a consistency check. The analysis presents the response time of each of the following classes of responses:

- *Fast hits*, where the cache returns object data to the requester without remote communication. The cache considers the object fresh and serves it directly.
- *Fast validations*, where the cache returns only a validation of freshness to the requester (HTTP code `304 Not Modified`), who presumably already has a copy of the data. The cache considers the object 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 requester after validating it with the origin server. In this case the client does not have valid object data, but the cache does, which it determines by contacting the origin server.
  This type is also referred to as *slow validation with data*, labeled *sdat* in the figures herein.
- *Consistency misses*, where the cache returns a copy of the

0890-8044/00/$10.00 © 2000 IEEE

object after contacting the origin server and getting a fresh copy of the object. The cache has the data but considers it stale and makes a consistency check, which returns the modified object.

- *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 was formerly in cache but has been evicted by the cache replacement policy).
- *Direct*, where the cache determines that an object is not cachable through configuration (certain types are declared noncachable) or through server response headers (e.g., `Cache-Control: nocache`). 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.

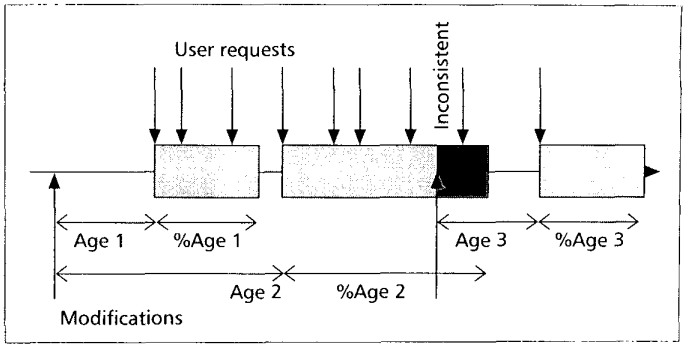
The key comparisons in this analysis are between fast and slow validations, and between fast and slow hits. In each of these cases the difference is the consistency check from the cache to the origin server. The validation case returns only HTTP headers from the cache to the client; the fast and slow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

HTTP Consistency Validation

One mechanism to determine object freshness is the Adaptive Time To Live (TTL) algorithm from the Alex file system [4]. The Alex protocol is described below and depicted in Fig. 1. The Alex protocol has been shown to be effective at maintaining consistency in practice and is widely used by cache implementations, which is why we chose to analyze its performance.

The Alex protocol defines content freshness as follows:

- Upon first retrieval of an object, the object's modification time is noted. This is a cold miss.
- The time between modification and current time defines the object's *age*. This value is shown as `Age 1` in Fig. 1.
- The cache computes a percentage of the object's age and defines that as the TTL. During the TT L period the object is considered *fresh* and will be served from cache. This first TTL period is `%Age 1` in Fig. 1. Requests for an object within its TTL period will result in one of two responses:
  - –If the requester has a copy of the object and makes an `If-ModifiedSince` (IMS) request, the response is a fast validation from the cache.
  - –If the requester 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 *stale*. Upon the next request the cache will make an HTTP conditional GET request with an 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 requester had object data, a slow hit if not.
  - –If the object has changed, a new copy will be resumed 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 Fig. 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 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 object freshness 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 validating objects in HTTP, such as HTTP/1.1 `Etags` and `Cache-Control`, but the `Last-Modified` freshness checking approach is the most widely used today.

Furthermore, if an object has an `Expires` header, the cache will consider the object 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 prior to its expiration time.

Analysis Methodology and Limitations

Cache log files, written by proxy cache servers as they complete user requests, contain information about the service of those requests. This usually includes the object URL, the object size in bytes, the service time of the request 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 extracted request service time, response size, and response type fields from a set of proxy cache server logs. For each response type we computed the frequency distribution of the response size and request service time metrics. From the frequency distribution we observed the mean, the median, and the 80th and 90th percentiles of the distribution, which led to conclusions about service time as a function of consistency. The mapping of log fields to response type is described in the full version of this article [5].

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 cannot 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.
- Service time indicates only how long the cache application took to service the request. It does not include processing time of the operating system (OS) kernel or network card on the cache system. The cache believes its task is complete

Attribute	Description
Response type	The response type as described in "Classes of Responses"
Percent of responses	The fraction of all HTTP GET responses of the class
Response time	The mean response time for the class
Response size	The mean response size (kbytes)
Percentiles	The median (50th) and 80th percentile response times for the class

■ Table 1. *Response attributes.*

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 cannot identify which objects are served from the browser's cache. These objects are likely to be displayed much more quickly than requests to the proxy 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.

Servers Studied

The analysis is based on the study of log files from three Web cache servers:

- **granite.hpl.hp.com**, January 1998–April 1999. This is a Squid server serving a workgroup in HP Laboratories (HPL), Palo Alto, California. During the period there were 3.3 million cachable requests.
- **proxy.hpl.hp.com**, April 1999. This Netscape cache serves all of HPL Palo Alto. During the period there were 11.4 million cachable 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 cachable requests.

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

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 percent or more of 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 presents the mean as well as the 50th percentile (the median) and the 80th percentile responses.

The analysis also excludes non-HTTP, error, and other response types, so the sum of the percent of responses may be less than 100 percent.

Squid on granite.hpl.hp.com

The first server studied was a workgroup 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 cachable requests.

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 than the median response time for fast validations. The response size is that of the HTTP response headers. No object data is sent or 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 5 kbytes. 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 resumed to the user, but fast hits require no remote communication; slow hits receive only a small validation from the origin server; and 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 and slow hits. Most of the consistency misses were for HTML objects. The increased response time is due to both larger mean HTML object size (compared with the mean object size of images, which accounts for most responses; see [5] for a breakdown of object size by type), 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

Response type	%	Mean size (kbytes)	Response time (s)		
			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 (fbit)	12.6	4.986	0.218	0.041	0.141
Slow hit (scat)	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. *granite.hpl.hp.com responses.*

Response type	Median size (kbytes)	80th %ile	90th %ile
Consistency miss	3.018	12.015	19.485
Miss	2.882	12.580	27.523

■ Table 3. *Squid response size and percentiles.*

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 Mbytes, so each response over 4 Mbytes is necessarily a 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 percent of misses from this server were larger than 27 kbytes.

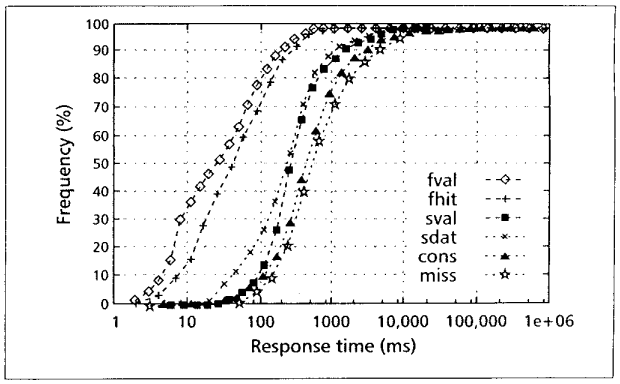
Figure 2 shows the distribution of response times observed at this server. Figure 3 shows the distribution of response sizes. These graphs present a cumulative distribution function (CDF), which indicates on the y axis the percentage of responses less than or equal to the time (or size) on the x axis.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

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. Seventy percent of fast hits and 80 percent of fast validations complete in under 100 ms. Fewer than 10 percent of slow validations complete in under 100 ms.

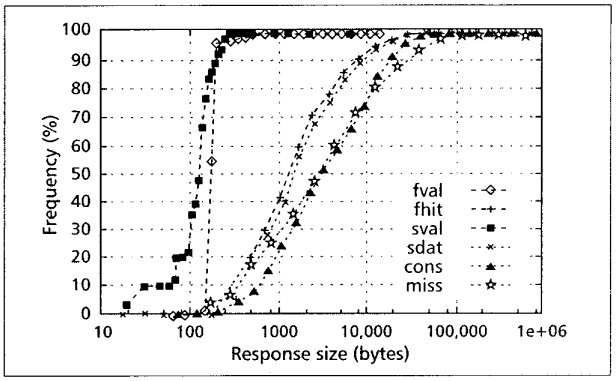
The response size distribution for fast validations is in a very small range, indicated by a nearly vertical line in the CDF in Fig. 3. The response size is determined by the headers sent by the cache, which are very similar for every response. Slow validation headers come from various origin servers, and show greater variability and smaller overall size. The Squid proxy 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.

More detail about response times and sizes for each response type is presented in [5].



■ Figure 2. *granite.hpl.hp.com response time CDF.*



■ Figure 3. *granite.hpl.hp.com response size CDF.*

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 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 to 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 CDF shows that there were very few of these objects: only 5 percent of objects were over 27 kbytes.

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, as observed earlier.

The response size distribution shows that fast hits and slow hits return similar amounts of data. Consistency misses and

Response type	%	Mean size (kbytes)	Response time (s)		
			Mean	50th	80th
Fast val	13.9	0.00[1]	0.071	0.032	0.072
Slow val	9.6	0.00[1]	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

[1] Netscape did not report HTTP header size

■ Table 4. *proxy.hpl.hp.com responses.*

Response type	%	Mean size (kbytes)	Response time (s)		
			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

■ Table 5. *Cable modem cache, all responses.*

Response type	Mean size (kbytes)	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

■ Table 6. *Cable modem response size distributions.*

regular misses are also closely matched. Validations (fast and slow) were reported as zero bytes.

Cable Modem Site

The third server studied was a Netscape cache server at a cable company acting as an Internet service provider (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 cachable 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.* [6]. 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.

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 percent longer than for slow hits; the mean response time for misses is 170 percent longer than fast hits. However, the median response time for misses is 20 times longer than fast hits.

Table 6 presents the mean, median, and 80th percentile response size for fast hits, slow hits, and misses.

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

Object size does not appear to be a dominant factor in response time, but object service location (direct from the proxy vs. requiring validation at the origin) does.

Summary and Conclusions

Caching of objects near end users significantly reduces 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 it is not in cache and must be retrieved. This corroborates earlier research in caching, and helps to explain the widespread deployment of Web proxy cache servers.

In this report we demonstrate that performing a consistency check prior to resuming an object from cache also has a significant impact on object retrieval time. When a validation is returned directly from a cache it is resumed eight to 15 times faster than an object that requires a consistency check with the origin server. This effect is especially significant if the origin server is slow, distant, overloaded, or has failed. Reducing the service time of requests will also allow a cache to support more total user requests.

Our results are similar across two very different proxy cache implementations and user bases, and widely ranging levels of request demand. This indicates that the characteristics observed are likely to be fundamental. This concurs with intuition: requests satisfied nearby are expected to be satisfied faster than requests to more distant servers.

Based on these results we suggest that there is an opportunity to improve user response time from caches by improving cache consistency. Earlier work in this area suggests strong consistency is possible to achieve in the Web at the same cost as weak consistency by using server invalidations [2]. We have proposed such a mechanism and shown via simulation 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, content consistency will have more relative impact on the performance of user requests and caches.

Acknowledgments

We are grateful for the time and expertise of Tai Jin who supplied logs from the local Squid cache and provided insight into its operation, and to Mike Rodriquez who ran our scripts across the full HPL logs. Thanks to Rich Friedrich for suggesting and supporting this work, and for his review comments. Thanks also to Martin Arlitt, John Barton, Ilja Bedner, Nina Bhatti, Radhika Malpani, Mark Nottingham, Stephane Perret, Jerry Rolia, Bill Weihl, Anna Zara, and the anonymous reviewers for their helpful comments, suggestions, and feedback.

References

[1] J. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency," *USENIX 1996 Annual Tech. Conf.*, Jan. 1996.
[2] C. Liu and P. Cao, "Maintaining Strong Cache Consistency in the World-Wide Web," *IEEE Trans. Comp.*, vol. 47, no. 4, Apr. 1998, pp. 445–57.
[3] J. Dilley *et al.*, "The Distributed Object Consistency Protocol," Tech. rep. HPL1999-109, HP Labs, Sept. 1999.
[4] V. Cate, "Alex — A Global Filesystem," *Proc. USENIX File Sys. Wksp.*, May 1992, pp. 1–12.
[5] J. Dilley, "The Effect of Consistency on Cache Response Time," Tech. rep. HPL-1999-107, HP Labs, Sept. 1999.
[6] M. Arlitt, R. Friedrich, and T. Jin, "Workload Characterization of a Web Proxy in a Cable Modem Environment," *ACM SIGMETRICS Perf. Eval. Rev.*, vol. 27 no. 2, Aug. 1998, pp. 25–36.

Biography

JOHN DILLEY [M] (jad@akamai.com) is a distributed systems architect with Akamai Technologies. His research interests include architecture and design of distributed applications, object location and distributed directory services, and object-oriented application design and development. He received B.S. degrees in mathematics and computer science from Purdue University in 1984 and an M.S. in computer science in 1985. He is a member of ACM.