

Optimizing Object Freshness Controls in Web Caches

Mark Nottingham

12/192 Little Collins Street Melbourne VIC 3000 Australia

mnot@pobox.com

Abstract

Mechanisms for controlling object freshness and refresh by are examined. Preliminary trace-driven analysis is preformed to explore object freshness relationships, a possible means of optimizing them based on traffic characteristics is introduced, and possible modifications to current mechanisms are contemplated.

Introduction

Although there are mechanisms in the HTTP, especially HTTP 1.1 [1], for publishers to control how Web caches handle objects, particularly their freshness, they are seldom used [2]. This may be because Web publishers see little subjective value in adding information (such as meaningful Expires, Last-Modified and Cache-Control response headers) that would be helpful to caches.

Due to this lack of server-hinted freshness information, shared cache administrators often attempt to simulate it in the cache configuration. One mechanism is *refresh control*, which determines when an object is no longer considered fresh, and must be revalidated with the origin server.

In this paper, optimizing refresh controls means to balance the need to keep an object fresh as long as possible against the possibility of serving stale content. Some cache operators will use refresh controls for other purposes, including the extension of freshness periods to increase bandwidth savings.

We will primarily focus on refresh control in Squid [3], as source is available for it, refresh control has been implemented and is stable, and it is in wide use. However, other products, such as NetCache [4] and CacheEngine [5] do allow similar control.

Benefits of Refresh Control

From a cache's perspective, latency can be reduced by two means:

- **avoiding object validation** – if the freshness of an object can be determined by server hints already stored on the cache, validation can be avoided. This allows substantial reduction in latency, because the cache can satisfy the request locally.
- **avoiding object transfer** – if freshness cannot be determined, but validation is successful (due to the object being unmodified, and the server supporting validation for the object), then the network transfer time can be avoided. This allows a substantial savings in bandwidth use, but not as much as avoiding object validation.

Well-balanced refresh controls will allow the server to minimize validation checks on origin servers, while still delivering reasonably fresh objects. This implies that the primary benefits are in reduction of user-perceived latency, not bandwidth consumption. However, refresh controls can be used to artificially prolong freshness of objects, allowing bandwidth savings to be achieved at the cost of object freshness.

As an aside, refresh control should only be applied to cacheable objects. If an object has no validators (e.g., Last-Modified response headers), it should not be considered cacheable for the purposes of refresh control.

Refresh Control Mechanisms

If objects on the World Wide Web are classified by one of several criteria, they can exhibit different characteristics, including the length of time before changes occur, which implies object lifetime before becoming stale. One common classification in this context is by file extension, or MIME type.

Cache administrators can take advantage of these characteristics to obtain more fine-grained control over their freshness. Often, this is done by declaring a set of regular expressions to be matched against the request URL, and a number of parameters.

Current Practice

The most widely used form of refresh control in use consists of a regex compared to the request URL, and the following settings:

- minimum time to cache
- maximum time to cache
- percentage of Last-Modified time to consider an object fresh

The minimum and maximum settings can be thought of as sanity checks; that is, they determine the point at which an object should always be considered fresh or stale, respectively.

More interesting is the percentage control. In the model below [6], the entered value is compared against the LM_FACTOR, as explained below, to determine freshness.

```
AGE = NOW - OBJECT_DATE
LM_AGE = OBJECT_DATE - LAST_MOD_TIME
LM_FACTOR = AGE / LM_AGE
```

The LM_FACTOR, in other words, is the ratio of the amount of time the object has aged since it was retrieved to the age of the object when it was retrieved. This gives a method of dynamically controlling the determination of object freshness, based on the age of the object relative to how long it has been unmodified.

The default in Squid and other products is to set a single refresh control for all objects. In an informal survey of use, most respondents used either the default, single refresh control, a modification of it, or a published list of suggested patterns. These lists are often conservative in their estimations, and represent a 'best-guess' of the publisher [7] [8].

More sophisticated operators may attempt to tune their refresh controls based on raw or byte hit rate, but these metrics give little feedback, and as such enforce a painfully slow optimization process.

Traffic-Driven Optimization

One potential means of optimizing refresh controls is to analyze past traffic for patterns in object freshness.

When limited to the information available to a cache, determining the appropriate lifetime for each class of object is a difficult and inexact task. However, determining the performance of the current patterns may be possible. Every request made to the cache (discounting requests for objects not in the cache, errors and other exceptions) is either fresh or stale. That is, it

will either be served directly from the cache, or first revalidated on the origin server. Stale requests, when revalidated, will either be *Not Modified* or will return new content.

The ratio of unmodified to modified stale requests, when filtered into the same classes as specified by refresh controls, can be used as a metric to adjust them with. A ratio that favors unmodified hits, for example, may indicate that the refresh control is conservative, while one that favors modified hits may indicate a refresh control that may be returning stale objects to clients. Over time, an optimum setting could be arrived at by applying these metrics.

Trace-Driven Analysis

To validate this model, trace-driven analysis is being performed, simulating a cache with a number of different refresh control parameters to establish a relationship between actual freshness (according to the trace), and the caches' determination of freshness. Then, this metric can be compared to the method outlined above.

Trace Characteristics

Because a cache that uses refresh control must be simulated, a trace containing validation information (i.e., Last-Modified HTTP response headers) must be used. Two suitable trace sets were found; the UCB Home-IP traces [13], and the DEC traces [14].

Both traces cover a period of less than a month, consist of multiple millions of object requests, and were taken in 1996. Because the analysis is sensitive to freshness, and inter-access and inter-change times are in excess of the scope of both traces, the DEC trace was chosen.

Due to the size of the trace, combined with limitations in equipment available, these preliminary results were run on a subset; every tenth object access in the log. This allows analysis over the entire period of the trace.

Analysis Model

To isolate the effects of LM_FACTOR refresh control from other decisions a cache makes about freshness, only objects with Last-Modified headers were considered.

Output is generated for each refresh control specified, and includes the number of cache hits actually fresh (according to the validator in the response), false fresh hits, hits actually stale and false stale hits.

680974 MISS					
10_percent.conf					
	. i	14353Tf	12526Ff	2603Ts	208Fs
	\.gif\$ i	371660Tf	31875Ff	2838Ts	1654Fs
	\.html\$ i	24235Tf	14607Ff	3901Ts	497Fs
	\.jpg\$ i	64398Tf	4733Ff	755Ts	819Fs
50_percent.conf					
	. i	13815Tf	14687Ff	1141Ts	44Fs
	\.gif\$ i	371879Tf	34530Ff	1267Ts	337Fs
	\.html\$ i	23945Tf	17443Ff	1738Ts	106Fs
	\.jpg\$ i	64726Tf	5484Ff	336Ts	159Fs
100_percent.conf					
	. i	13782Tf	15029Ff	802Ts	21Fs
	\.gif\$ i	371915Tf	35079Ff	847Ts	167Fs
	\.html\$ i	23924Tf	17983Ff	1277Ts	43Fs
	\.jpg\$ i	64793Tf	5588Ff	253Ts	71Fs
500_percent.conf					
	. i	13753Tf	15501Ff	373Ts	7Fs
	\.gif\$ i	371960Tf	35685Ff	330Ts	33Fs
	\.html\$ i	23903Tf	18760Ff	554Ts	10Fs
	\.jpg\$ i	64821Tf	5747Ff	125Ts	12Fs
1000_percent.conf					
	. i	13753Tf	15654Ff	280Ts	3Fs
	\.gif\$ i	371976Tf	35831Ff	210Ts	10Fs
	\.html\$ i	23874Tf	19010Ff	352Ts	4Fs
	\.jpg\$ i	64817Tf	5784Ff	99Ts	5Fs

figure 1: preliminary trace-driven results

Tf – Truly fresh *Ff* – false fresh *Ts* – Truly stale *Ff* – False stale (from the cache config's perspective)

The refresh controls chosen were somewhat restricted by the trace characteristics. LM_FACTORS of 10%, 50%, 100%, 500% and 1000% were run against the following patterns: '\.gif\$', '\.jpg\$', '\.html\$' and '.'.

Limitations

The age of the trace is of some concern; HTTP 1.1 has become much more prevalent since 1996. Also, the nature of Internet content itself has changed in the intervening time.

While the trace has a large number of accesses, they fall within a relatively short period; 25 days. As the inter-access and inter-change times for objects have been demonstrated to have considerably larger means than this [15], this is the primary limitation of this analysis.

Results

Preliminary results, as shown in figure 1, are inconclusive. Because the period of the trace is insufficient to cover the freshness period of common objects, there are not significant numbers of stale objects to work with.

To properly validate the model (as well as explore other interesting relationships in freshness), traces that cover larger periods while including validators must be made available.

Experimental Implementation

To explore the proposed model in an active cache, a limited implementation is available. Squij [9] is a Squid-format logfile analyzer, written in Python [10]. By parsing the Squid configuration file for refresh patterns and applying them to the logs, a reasonably accurate picture of the characteristics of the traffic. It is not complete, due to the lack of some information in the logs, as well as the nature of the HTTP.

The output is expressed as the following, for each refresh classification found:

- **raw hit percentage, requests and bytes** – this shows what portion of the objects requests were in the cache, due to shared hits.
- **ratio of fresh to stale hits, for objects in the cache** – an indication of how aggressive the current refresh control is.

refresh_pattern -i	\.gif\$	1440	500%	262800
refresh_pattern -i	\.jpg\$	1440	700%	262800
refresh_pattern -i	\.htm\$	20	50%	40320
refresh_pattern -i	\.html\$	20	40%	40320
refresh_pattern	\/\$	15	25%	20160
refresh_pattern -i	\.exe\$	2880	1000%	262800
refresh_pattern -i	\.pdf\$	2880	1000%	262800
refresh_pattern -i	\.zip\$	2880	1000%	262800
refresh_pattern -i	\.mov\$	2880	1000%	262800

figure 2: example implementation configuration

- **ratio of objects modified to unmodified on the origin server, for hits which were stale** – shows whether the refresh control is to aggressive or conservative.
- **total number of hits and bytes to the classification** – to give an idea of how effective the classification has been.
- object has become invalid in the cache.
- Success is dependent on the appropriate classification of object types; poorly chosen controls will not respond well to optimization.
- Objects not following the behavior of their classification may be marked fresh for too long, or stale prematurely.

This output was then used to redefine the refresh controls over time, until the modified to unmodified ratio approached 1:1 (usually on the conservative, unmodified side). This was done predominantly by adjusting the refresh percentage (as outlined above).

Limitations

Because of the limited knowledge available through the logs, there are several limitations to this method, including;

- Downstream ICP traffic is ignored. The effects of ICP introduce too many variables and unknowns to be considered here.
- Its knowledge of object freshness is limited to when the cache validates the object with the origin server. The implementation cannot know when an

Despite this, the implementation gives a more solid metric to determine refresh control than those currently available.

Results

Due to the relative nature of the analysis, combined with the limited knowledge available to the cache, it is difficult to validate the implementation. However, use on real-world caches gives provocative results.

As an example, we examine both the refresh controls for a medium-size, corporate Squid proxy [12].

The squid.conf used, and the test implementation's output are listed in figures 2 and 3.

```

squid j output: Mon Jan 11 00:01:50 1999 to Mon Jan 11 23:59:40 1999
  regex      ave svc   hit/byte   fresh/   unmod/   total hits/
            time      rate      stale   modified  bytes
-----
  \.gif$ i     1.6    85%/ 68%    2:1     24:1     27749/  44M
  \.jpg$ i     6.7    61%/ 38%   13:2     9:2     3043/  18M
  \.htm$ i     4.1    35%/ 17%    5:2     1:1     1723/   8M
  \.html$ i    9.1    23%/ 10%    5:2     1:1     1952/  16M
  \/$ i       7.3    42%/ 21%    5:2     1:1     1343/   7M
  \.exe$ i   450.3   31%/ 16%    1:0     0:0      16/   8M
  \.pdf$ i   10.1    42%/ 44%    1:0     0:0      57/   5M
  \.zip$ i    0.1   100%/100%  1:0     0:0       3/   0M
  \.mov$ i    0.0   100%/100%  1:0     0:0       3/   0M
  .         7.8    29%/  5%    2:7    23:2     9528/  63M
  OVERALL  3.0    66%/ 29%    3:2    25:2    45417/ 169M

```

figure 3: example implementation output

If the basic premises outlined in this paper are accepted, several points are evident;

- Image classifications (.gif|.jpg|.jpeg\$) can be used with extreme values. In this case, 700% refresh controls still saw 24:1 unmodified to modified ratios for .gif's, with a 2:1 fresh:stale ratio. Although 85% of these objects were in the cache, the server could be tuned more aggressively to take advantage of their relatively high freshness lifetimes.
- Because of the lack of traffic in (.exe|.pdf|.zip|.mov\$), it is difficult to make meaningful conclusions about them from this example.
- HTML (.html?\$) and directory (\/\$) classifications need to be much more conservative than others. Here, at between 25% and 50%, it is likely that too many pages are being served stale. Of the 2/5 that the cache determines are stale, half fail validation on the origin server.

In comparison with both the default and recommended values mentioned above, these are several orders of magnitude more aggressive. Despite this, users of this cache do not report a high occurrence of stale objects.

Interestingly, the CacheEngine allows classification on only one parameter; whether or not a file is binary. The results above support this as a viable, if not very flexible, method for refresh control classification.

Further Work

Self-modifying refresh patterns

Once the basic premises involved are validated, a potential application would use the output of traffic-driven analysis to automatically configure refresh control. In such a system, administrators could either use a supplied baseline configuration, based on common object characteristics, or supply their own. In either case, such a mechanism could allow self-tuning of the refresh controls based on traffic characteristics.

Alternative refresh control functions and selectors

Another avenue of potential exploration is defining more efficient means of selection for object classes. Currently, the only available mechanism is by regular expression applied to the object URL. However, other work in the field suggests grouping by such metrics as object size, MIME type, temporal locality and number of shared references. [2] [11]

Conclusions

Optimizing refresh control is a relatively unexplored means of predicting objects freshness that can have meaningful effects in reducing latency, while mitigating the risk of serving stale objects.

Although there are potentially interesting relationships in the limited data that cache logs afford, relevant traces must be produced, in order to validate assumptions made about the freshness characteristics of objects, as viewed through this as well as future models.

References

1. R. Fielding, U. C. Irvine, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. Hypertext transfer protocol HTTP/1.1. <http://www.w3.org/Protocols/>
2. S. Gribble, E. Brewer. System Design Issues for Internet Middleware Services. USITS '97 proceedings. December 1997, Monterey, CA. Table 2.
3. D. Wessels. Squid Internet Object Cache – <http://squid.nlanr.net/>
4. P. Danzig. Network Appliance NetCache – <http://www.netcache.com/>
5. Cisco CacheEngine – <http://www.cisco.com/cache/>
6. Object freshness model – Squid documentation (doc/Release-Notes-1.1.txt)
7. Private e-mail solicited from cache administrators on Squid-users list (squid-users@nlanr.net)
8. <http://auix.esc.net.au/refresh1.2>
9. Squij – <http://www.pobox.com/~mnot/squij/>
10. G. van Rossum. The Python Programming Language – <http://www.python.org/>
11. B. Duska, D. Marwood, M. Feeley. The Measured Access Characteristics of World-Wide-Web Client Proxy Caches. USITS '97 proceedings. December 1997, Monterey, CA.
12. Anonymous corporate cache, Sydney, Australia. Squid 2.1PATCH2, 8G disk, 172M RAM, ~160 users daily.
13. Berkeley Home IP traces. <http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html>
14. Digital Equipment Corporation proxy traces. <ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html>
15. F. Douglass, A. Feldmann, B. Krishnamurthy. Rate of Change and other Metrics: a Live Study of the World Wide Web. USITS '97 proceedings. December 1997, Monterey, CA.