Piggyback Cache Validation for Proxy Caches in the World Wide Web

Balachander Krishnamurthy
AT&T Research Laboratories

Craig E. Wills
Worcester Polytechnic Institute

Abstract

This paper attempts to tackle the cache coherency problem for documents without an expiration time. Rather than use a strictly weak or strong cache consistency approach, we piggyback cache state information onto HTTP requests to a server. In the simplest approach, whenever a proxy cache has a reason to communicate with a server it piggybacks on a list of cached documents for which the expiration time is unknown.

This mechanism can reduce the number of requests and connections between a proxy cache and a server, while providing close to strongly consistent cache information. Alternatives that we are considering could also be used to provide a more cost effective server invalidation mechanism.

1 Introduction

A proxy cache is a machine that acts as an intermediary between potentially hundreds of clients and the remote web servers by funneling requests from clients to various servers. In the process, the proxy caches frequently requested pages to avoid contacting the server repeatedly for the same page (if it knows the information on the page has not changed on the server).

A problem with caching documents at a proxy (or within a browser cache) is the issue of cache coherency—how does the proxy know that the cached document is still current? In one case the server knows how long a document is good (e.g. daily newspaper generated each morning at 5:00am) and the server can provide a precise expiration time. Cached copies are always fresh until the expiration time.

In the other, more common, case the document that is made available has no clear expiration time. It may change in 5 minutes or remain unchanged for a very long time. Two schemes have been proposed and investigated [LC97, GS96].

At one end of the spectrum, strong cache consistency is maintained via one of two approaches. In the first approach, *client validation*, the proxy treats cached documents as potentially out-of-date on each access and sends a If-Not-Modified header with each access of the document. This approach provides strong cache consistency, but can lead to many 304 (Not Modified) responses by the server if the document does not actually change often. The second approach is *server invalidation*, where the server keeps track of lists of clients to use for invalidating cached copies of changed documents. This approach becomes unwieldy for a server when the number of clients is large. In addition, they can become out-of-date causing the server to send invalidation messages to clients who are no longer caching the document.

In contrast, weak consistency approaches seek to minimize the proxy validation and server invalidation messages by using a heuristic or a pre-defined value as an artificial expiration time on a cached document. One such approach, based on the last modified time, is for the proxy to adopt the FTP Alex protocol [Cat92]. The older the document, the longer the time period between validations. This heuristic is reasonable, but can leave periods where the cached document is potentially stale.

2 Piggyback Cache Validation Approach

Rather than use a strictly weak or strong cache consistency approach, we piggyback cache state information onto HTTP requests to a server. While individual browser clients could use our approach, it will mostly benefit proxy caches because the number of documents cached from a particular server is small and likely short-lived in a individual browser cache. Thus, we focus on the use of the piggyback mechanism for a proxy cache.

In the simplest approach, whenever a proxy cache has a reason to communicate with a server it piggybacks a list of cached documents (obtained from that server) for which the expiration time is unknown. The server handles the request and indicates which cached document on the list are stale (or indicates they are all fresh). The proxy then updates its cache. We then use a relatively small expiration time threshold, such as an hour, at the proxy cache. Client requests for cached documents with a validated age of less than this expiration time are treated as current while requests for cached documents that have not recently been validated cause a If-Not-Modified request to be sent to the origin server.

The performance of this approach depends on the number of documents cached at a proxy for a particular server and the number of requests that are sent by the proxy to the server. If there are few such requests so that the cache contents do not get validated with piggybacking then our approach is close to strong cache consistency validation. At the other extreme, if there is traffic between the proxy and server, then the cache contents are validated at the granularity of the time threshold. At this extreme, our approach is like weak cache consistency approaches, but with a smaller, tight bound on the potential staleness of cached documents.

The added cost of our mechanism is mainly in the increased size of the regular request messages due to piggybacking. There are however no new connections made between proxies and servers. The number of piggybacked checks appended to a single request can be controlled by the proxy cache. The cost for the proxy cache is to maintain a list of cached documents on a per server basis. The cost for the server is that it must validate the piggybacked documents in addition to processing the regular request but this is validation that it may have to do anyway in the absence of piggybacking.

3 Implementation

We are currently working on a simulation study of our ideas with existing proxy cache logs and expect to have results shortly.

Implementation of piggybacking requires the introduction of a new field type for validation list requests and replies. This idea of piggybacking validation requests could also be implemented using the pipeline mechanism of HTTP 1.1 with no changes to the protocol. If a server does not implement the mechanism, proxy cache continues to work fine, albeit without the piggybacked validation information, in either implementation.

We are also looking at variations of our basic approach. A potential problem with the approach is a potentially large piggybacked validation list. An alternate approach is to generate a group validator for the set of cached documents (a compressed representation of the unique resource ids, like *etags*). This validator could be generated by the client for just the set of documents it caches, but the client must still identify this set to the server.

Alternately, a server could provide a validator for all the documents at its site or selected sets of documents [Mogul]. This server-provided group validator could then be piggybacked by the proxy cache in subsequent requests. The advantage of this approach is that the amount of piggybacked information is smaller, but there is potential for cached documents to be invalidated by a new server validator when in fact the specific cached documents may have not been changed on the origin server.

The logical extension of this alternative is to consider having a server break the set of documents it manages into volumes or levels where the first level could be the document type, the next level based on co-location and finally a level determined by the rate at which a document is likely to change.

A validator for a set of documents could also be combined with the server maintaining the list of modified documents that cause the validator to change. The validator could be used by proxies to determine if a change has occurred at the server and if so the server could supply the list of modified documents. This alternate resembles server invalidation, but the key difference is that servers only respond if a proxy expresses an interest (i.e. still has cached documents) and servers do not need to maintain explicit client lists.

4 Summary

We believe the mechanism of piggybacking validation checks to requests that would be normally made to a server has potential. It is a low-cost mechanism in that can reduce the number of requests and connections between a proxy cache and a server, while providing close to strongly consistent cache information. The strength of the cache consistency can be traded off for the amount of piggybacked traffic that is introduced and the maximum potential staleness of the information.

Alternatives that we are considering could also be used to provide a more cost effective server invalidation mechanism. The general approach may also be applicable to situations other than cache validation, where the proxy cache and server can communicate on a topic unrelated to the specific request itself. This approach provides a mechanism for communication to take place between client and server without the server needing to explicitly initiate the communication.

References

[GS96] James Gwertzman and Margo Seltzer World-Wide Web Cache Consistency In *Proceedings of USENIX 1996 Conference*

[LC97] Chengjie Liu and Pei Cao Maintaining Strong Cache Consistency in the World-Wide Web: In *Proceedings of ICDCS 1997 Conference (to appear)*

[Cat92] V. Cate Alex - A Global Filesystem Proceedings of the Usenix File Systems Workshop, May 1992, pages 1-11.

[Mogul] Jeffrey Mogul An alternative to explicit revocation? http-caching mailing list, 2 January 1996