Strong Cache Consistency for the World-Wide Web

Chengjie Liu and Pei Cao
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706
chengjie,cao@cs.wisc.edu

October 21, 1996

The exploding popularity of the World Wide Web (WWW) has led to increasing traffic on the Internet and consequently an increased latency in document retrieval. To mitigate the network bandwidth constraints, caching has been widely employed by browsers and client side proxies. Cache consistency must be maintained, that is, cached copies should be updated when the originals change.

Existing Web caches mostly employ mechanisms that provide weak consistency. That is, a stale document maybe returned to the user, though infrequently. Weak consistency mechanisms include *TTL (Time-To-Live)*, which uses a time-to-live attribute to determine whether a document is up to date, and client polling, in which clients periodically contact Web servers to verify cached copies.

Weak consistency, however, is not always satisfactory. We investigated the cost and performance of two approaches that provide strong consistency, *invalidation* and *polling-every-time*. In the invalidation approach, the Web server keeps track of all the client sites that cache a document, and when the document is changed, sends invalidation messages to the clients. In the polling-every-time approach, every time the user requests a document and there is a copy of the document in the cache, the cache first contacts the Web server to validate the cached copy before it returns the copy to the user.

To study the performance of these cache consistency protocols, we designed a simulation system employing three cache consistency protocols, *adaptive TTL*, polling-every-time, and invalidation. We built our simulation based on Harvest cache system. Original Harvest cache uses plain TTL approach. We changed it to adaptive TTL so that it sends out polling message during a cache miss. In our simulation, we also modified Harvest source code and made it to poll every time to obtain strict consistency. Another subsystem we implemented uses invalidation approach.

We used several Web traces to run simulations and study three approaches. We found that polling-every-time is not as good as we thought. Although it maintains strict consistency, it sends polling message for every request and therefore has longer average response time. Adaptive TTL and invalidation are similar in average response time and network traffic for most traces. However, both of them have drawbacks. Adaptive TTL does not guarantee strict consistency. Invalidation approach has some "bursty" behavior while sending out invalidate message for a widely shared document.

In the future, we plan to run more simulations with various traces, especially traces from busy Web servers. We also plan to build a hierarchical caching system and implement another approach combining both adaptive TTL and invalidation.