Not All Hits Are Created Equal: Cooperative Proxy Caching Over a Wide-Area Network

Michael Rabinovich*
AT&T Labs
180 Park Avenue
Florham Park, NJ 07932
misha@research.att.com

Jeff Chase   Syam Gadde
Department of Computer Science
Duke University
Durham, NC 27708
{chase,gadde}@cs.duke.edu

Abstract

Given the benefits of sharing a cache among large user populations, Internet Service Providers will likely enter into peering agreements to share their caches. This position paper describes our approach for inter-proxy cooperation in this environment. While existing cooperation models focus on maximizing global hit ratios, values of cache hits in our environment depend on peering agreements and access latency of various proxies. It may well be that obtaining an object directly from the Internet is less expensive and faster than from a distant cache. Our approach takes advantage of these distinctions to reduce the overhead for locating objects in the global cache.

1 Introduction

Recent studies of Web access traces have showed significant benefits of providing a shared cache to a large user population. This prompted an active interest in large-scale distributed proxy caching platforms, in which multiple proxies cooperate and share the cached objects.

Existing approaches to inter-proxy cooperation try to maximize global hit ratios. However, there are cases in which *distant* copies of objects in the cache may not be worth fetching, and for which the source server may be a better choice. This may certainly be the case in very wide area collective caches as might occur with cooperating and independently managed ISPs. Depending on peering agreements and characteristics of links between ISPs, it may be beneficial in terms of costs and/or latency to fetch an object from the source rather than an outside proxy. Note that for a caching platform contained entirely within an ISP, tradeoffs may be different. In this case, it might be valuable to to maximize hit ratio even at the expense of the latency, to reduce the inflow of outside traffic. Such environments would be better served by existing approaches (e.g., [2, 7, 6], and many commercial products).

*Contact author; tel. (973)360-8778.

Given that there is little value in distant hits, our observation is that the system should not try to maximize the global hit ratio. Consequently, instead of paying the overhead for maintaining global location information of cached objects, each node in our approach maintains a directory of objects cached in its vicinity. For each proxy this directory is different. While the global hit ratio is reduced, our approach sacrifices hits of little value.

In addition to reducing the space overhead for directories, our approach also eliminates the need for long-haul communication to propagate directory updates. Indeed, since a proxy does not need to know about objects cached at distant nodes, directory updates are always propagated among proxies in the vicinity of each other.

There are currently two proposals for cache cooperation at Internet peering points, from NLANR and Mirror Image Internet [10, 9]. These are the kinds of approaches which, when deployed over many peering points and ISPs, would benefit from our proposal.

2 Target Environment

Consider a collection of ISPs, each with a proxy. We do not consider the implementation of the ISP proxy - there are many commercial proxy products targeting the ISP market. ISP proxies cooperate to share cached objects among their clients. For each proxy, there is a "neighborhood" of other ISP proxies such that it is better to fetch objects from these proxies as opposed to the Internet directly. The neighborhoods of different proxies overlap. The information about the neighborhood of a proxy is provided by the proxy administrator based on the peering agreements of the proxy's ISP with other ISPs.

We assume each pair of neighbors is assigned a distance metric reflecting the costs of transferring data between them. This may include monetary costs, latency and bandwidth of the connection, etc. We assume that these metrics are globally comparable, that is, if the metric of a pair (i, j) is greater than that of (p, q), then sending data between i and j is more expensive than between p and q. As an approximation, one could adopt an approach from BGP routing and assign metric 1 to all neighbor pairs.

Our goal is to exploit the knowledge of the neighborhoods to provide an efficient model for inter-proxy cooperation.

3 Related Work

A fundamental question distributed proxy architectures must resolve is how a cache that received a request for an object it does not have finds out if any other cache has it before declaring a global miss. The three main existing approaches include broadcast probe, exemplified by Harvest [1] and Squid [3], hash-partitioning of the object namespace among caches [2] and a directory service first proposed in CRISP [7].

None of these approaches scale well to inter-ISP caching. Broadcast probes degrade performance of global misses. On a global miss, a cache at every level of hierarchy must wait for the *slowest* response from all probed nodes before proceeding to the next level, which is especially harmful when any of the nodes do not respond or a message is dropped in the network. Moreover, the document is then passed through the tree on the way from the end-server to the client. Hash-partitioning does not respect any topological locality: requests from all clients for an object must go the cache responsible for that object regardless of the distance between the client and the cache. The simple directory service implemented in the first version of CRISP involves synchronous queries to the centralized directory on a local miss.

The summary cache proposal [4] significantly improves Harvest/Squid by providing a compact way of maintaining very strong hints on whether or not an object is cached at any node in the system. Thus, probes are performed only when the object is cached somewhere, and the global miss penalty is all but eliminated. Still, synchronous probes remain on a critical path of request processing for hits.

Another improvement to Harvest/Squid scheme, proposed in [13], improves utilization of aggregate cache space by eliminating duplicate copies stored in parents and children. On a wide area network, however, having multiple copies may actually be beneficial, since it reduces hit distances.

A follow-up work on CRISP studied various alternatives for a directory service [6]. One of the proposed alternatives is to fully replicate the directory on every proxy and asynchronously propagate local changes in each cache to the rest of the directories to maintain all replicas weakly consistent. The main concern with this approach is space overhead for maintaining replicated directory on every node. Another approach uses a central directory service in a novel way - only to identify a subdirectory of objects that are shared by more than one client. It is this subdirectory that is then replicated among the proxies. Beside identifying objects to be included in the subdirectory, the

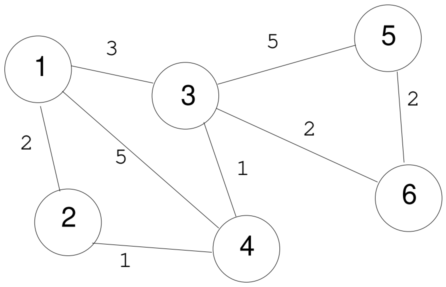

Figure 1: A neighborhood graph.

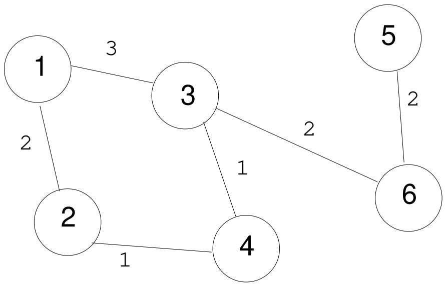

Figure 2: The communication graph corresponding to the neighborhood graph of Figure 1.

central directory is also used to keep subdirectory replicas on all proxies loosely consistent. While cutting substantially the space overhead, this approach sacrifices hits due to the second accesses to shared objects, which resulted in over 5 percentage point drop in hit ratio. Our approach also reduces global hit ratio, but it sacrifices hits of little value.

The work of [12] proposes to use a Harvest-style hierarchy to propagate object location hints. Since the system attempts to capture all global hits, it pays the overhead for maintaining and propagating hints even in cases when hits delivered by these hints would be of little value. Also, our proposal incorporates a much more direct notion of topological proximity.

4 The Protocols

The system is represented by a (not necessarily connected) *neighborhood graph* with proxies as nodes and edges connecting neighboring proxies (Figure 1). Based on this graph, a *communication graph* is constructed where each pair of nodes is connected via a shortest path on the neighboring graph. The communication graph is constructed using a protocol similar to the Internet's BGP routing protocol [8]. The same mechanism is also used by the system to automatically reconfigure the communication graph in response to node or link failures and repairs. Figure 2 shows the communication graph corresponding to the neighborhood graph of Figure 1.

Each node i maintains a directory of objects cached in the "vicinity" of the node, where the vicinity is defined to be a set of nodes whose distance from i is below a threshold v. An entry for object x, $dir(x)$ contains the id of the proxy believed to be the closest one having x in its cache, $dir(x).node$, and the distance from that proxy to i, $dir(x).dist$. The directory maintenance is done by forwarding changes in local cache contents along the communication paths and will be described shortly.

One important detail our design must deal with is that there may be multiple communication paths between the same pair of nodes i and j. This may cause a node to learn about cache additions and deletions at other nodes in a wrong order, even if every edge in the communication graph is a FIFO channel. As an example, nodes 1, 2, 3, and 4 on Figure 2 are all in the same neighborhood (assuming $v = 5$). Node 4 may learn about cache updates at node 1 via node 2 or node 3. Assume that node 1 added object x into its cache, then sent the notification to its neighbors (nodes 2 and 3) then removed object x from the cache and then sent another notification. If node 4 learns about both updates via node 2 and then receives a notification from node 3 containing just the first update (addition of x to node 1's cache) then it should not act on this last notification. Otherwise, it would erroneously add into its directory an entry indicating that x exists in node 1's cache.

To handle this issue efficiently, we observe that a node should discard either all or none of the updates to any given node's cache from any notification message received. This allows us to avoid keeping timestamps for individual directory entries. Instead, we use *timestamp vectors*, which contain timestamps per node rather than per entry. Timestamp vectors of various flavors have been used in distributed computing for multiple purposes, and to the best of our knowledge, were first introduced in [11]. Formal treatment of timestamp vectors including proofs that they indeed avoid problems like the one of the above example, is given, e.g., in [5].

In our approach, each node i maintains a timestamp vector T_i. T_i has an element for every node in i's vicinity. Element T_{ij} contains the time of the last notification from j that i is aware of. (Note that i may become aware of a notification transitively, through other nodes.)

To see intuitively how timestamps can help, let us assume that, in the above example, node 1 generated its notifications at local time 10, 11. Then, the notification messages to nodes 2 and 3 will carry with them the timestamp vector with the element T_{11} equal to 10 and 11, respectively.

Since the notification from node 2 to 4 was sent after node 2 had seen both notifications from node 1, the notification from 2 to 4 will carry a timestampt with $T_{21} = 11$. Then, after receiving this notification, node 4 will also have timestamp vector with $T_{41} = 11$, and therefore will discard any updates originated from node 1 if they came in any notification message carrying a timestamp vector in which the element corresponding to node 1 is less than or equal to 11. In this way, the stale update coming with the notification from node 3 will be discarded (note that other updates in this notification message, those regarding other nodes' caches, may still be fresh and should be added to node 4's directory).

4.1 Directory Maintenance

Every node i keeps a log of updates of its local cache (i.e., deletions from and additions to its cache), and also updates to its directory due to notifications from other proxies. Each update has a format $(p, x, action, dist, time)$, where p is the id of the node that originated the update (i.e., whose local cache changed), x is the id of the object, *action* tells whether an object was added to or removed from the *nodes*'s cache, *dist* gives the total distance between *node* and node i, and *time* gives the time of the change according to *node*'s local clock. Node i also maintains a *notification set*, which is a subset of updates that i has yet to communicate further to its neighbors. An update is included in the notification set if *dist* is below a *vicinity threshold*, v. Finally, each node maintains a variable *age* which records the time of the oldest update in its notification set.

Whenever the difference between *age* and the current local time exceeds a delay threshold τ, node i sends its notification set to all its neighbors in the communication graph.[1] Prior to sending the notification set, i records the current value of its local clock in the timestamp vector element corresponding to itself, T_{ii}, and includes the entire timestamp vector in the notification message. i also records the local clock value in its *age* variable.

A node j, upon receiving the notification message from a neighbor i, executes the following steps (the algorithm below stresses clarity over efficiency - several straightforward performance improvements are possible).

1. For each node k such that the received timestamp vector has an element for k, T_{ik}, compare

[1]Note that we neglect the effects of clock mis-synchronization on deciding when to propagate the notification set. Given that τ is in the order of minutes and clock synchronization on well-administered machines is within seconds, this seems a sensible practical decision. For a theoretically inclined, clock synchronization could be piggybacked on the notification exchange messages.

this element with T_{jk} if present. If T_{jk} exists and is equal or greater than T_{ik}, discard all update records with $node = k$ from the notification set;

2. For each remaining update $(p, x, action, dist, time)$,

   - Update $dist$ to equal $dist_{new} = dist + d_{ij}$, where d_{ij} is the distance label of the edge between i and j.
   - If $dist_{new} \leq v$ (i.e., i is in j's vicinity) apply the update to the directory. For a deletion, if the directory contains an entry for x with the same node as that of the update record (i.e., $dir(x).node = node$), remove the entry. For an addition, if either the current directory has no entry for x or the entry lists a more distant node than that of the update (i.e., $dir(x).dist > dist_{new}$) add or replace the existing entry with a new entry with $dir(x).node = p$ and $dir(x).dist = dist_{new}$.
   - If the directory was modified in the previous step, add the update $(p, x, action, dist_{new}, time)$ to the notification set, to be propagated further to j's neighbors in the next notification message.

3. Update j's timestamp vector. For each node k that has a corresponding element in T_i but not in T_j, create a new element $T_{jk} = T_{ik}$. For each node q that has elements in both vectors, set $T_{jk} = \max(T_{jk}, T_{ik})$.

4. Finally, if the oldest $time$ among the non-discarded updates received, $time_{oldest}$ is older (smaller) than the current value of age, then set age to $time_{oldest}$.

4.2 Request Processing

When a proxy receives a request for a page it does not have in its local cache, it checks its directory. If the object is found there, the proxy attempts to fetch it from the proxy listed in the corresponding directory entry. Otherwise, or if the above attempt fails, the proxy fetches the object from the Internet. In either case, the proxy may choose to add the object in its cache[2], in which case it adds an addition update to its notification set. If this caused removal of any objects from its cache, each removal generates a deletion update in the notification set. All these updates have $dist = 0$.

[2]We do not discuss policies for admitting objects in the cache here.

Since a proxy is notified of only updates that occur at proxies within distance v, its directory will only index the contents of these proxies. This limits the size of the directory and the notification traffic. On the other hand, modulo the notification delay, the directory on every reflects the whole contents of all proxy caches in the proxy's vicinity. The analysis of the DEC trace indicated that a propagation delay of $5 - 10$ minutes resulted in negligible decrease in the hit ratio. Thus, if the delay is limited to the above number (which is more than reasonable in practice), virtually all vicinity hits will be utilized.

5 Self-Synchronization

As described, our system may exhibit a *self-synchronization* property. This property, which is a common danger in networking, means that all nodes eventually start taking a certain action in unison, degrading the system performance. In our case, the system may converge to a situation where many nodes send notifications at about the same time. This could result in periodic bursts of traffic. While the likelihood of this danger in our case is unclear at this point, we could guard against it by varying τ randomly (within certain limits) at each node.

6 Conclusion

This position paper outlines a new model for cooperation between Internet proxy caches. The underlying observation is that as the network distance between proxies increases, the value of obtaining an object from another proxy quickly decreases, as compared to obtaining it directly from the Internet. Existing approaches that attempt to always maximize the hit rate incur overhead for locating objects in other caches without necessarily benefiting from finding those objects.

On the other hand, in our approach, each proxy indexes only the objects cached in its "vicinity" in the network. Thus, the overhead for object location is incurred only when it pays off to find the object.

Our approach should be especially useful for inter-ISP cache sharing. The growing interest to such sharing is evidenced by the proposals from NLANR and Mirror Image Interenet for cache cooperation at Internet peering points [10, 9]. The paper describes the basic architecture and protocols of our approach. An implementation of this approach is underway.

References

[1] Anawat Chankhunthod, Peter Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical internet object cache. In *Proceedings of the USENIX 1996 Annual Technical Conference*, January 1996.

[2] J. Cohen, N. Phadnis, V. Valloppillil, and K. W. Ross. Cache array routing protocol v1.1. http://ds1.internic.net/internet-drafts/draft-vinod-carp-v1-01.txt, September 1997.

[3] Duane Wessels et al. Squid internet object cache. http://squid.nlanr.net/.

[4] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. Technical Report 1361, Computer Sciences Dept, University of Wisconsin – Madison, February 1998.

[5] C. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *Proc. of the 11th Australian Computer Science Conf*, pages 56–66, 1988.

[6] S. Gadde, J. Chase, and M. Rabinovich. Directory structures for sclabale internet caches. Technical Report CS-1997-18, Dept. of Computer Science, Duke University, November 1997.

[7] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: An approach to building large internet ccaches. In *Proc. of the 6th Workshop on Hot Topics in Operating Systems (HOTOS-6)*, May 1997.

[8] C. Huitema. *Routing in the Internet.* Prentice Hall, 1995.

[9] Mirror image internet announces industry's first open two-tiered caching solution. press release. http://www.mirror-image.com/newsdoub.htm, 1998.

[10] Web caching as a viable exchange point service. http://ircache.nlanr.net/Cache/mae-west/.

[11] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Trans. on Software Eng.*, 9(3):240–246, May 1983.

[12] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay. Beyond hierarchies: design considerations for distributed caching on the internet. Technical Report TR98-04, Dept of Computer Sciences, University of Texas – Austin, 1998.

[13] Ph. Yu and E. A. MacNair. Performance study of a collaborative method for hierarchical caching in proxy servers. Technical Report RC 21026, IBM T.J. Watson Research Center, November 1997.