Distributed Web Proxy Caching in a Local Network Environment

Michael Piatek*
Department of Mathematics and Computer Science
Duquesne University
600 Forbes Ave.
Pittsburgh, PA 15282
Advisors: Jeffrey Jackson, Patrick Juola†

Abstract

While web proxy caching has been extensively studied as a means for improving network performance, existing designs lack scalability or cannot guarantee a user performance benefit. We investigate the feasibility and performance of a locally distributed, self organizing network proxy that solves these technical problems and also eases the configuration burden on individual users and network administrators. Our implementation also features varying degrees of anonymity. Trace-driven simulation suggests cache hit rate justifies deployment of distributed caching systems.

1 Problem and motivation

The problem of limited bandwidth in network systems has permeated network application design since the development of the early Internet. Undoubtedly, web browsers are the most prolific of these applications and significant efforts have gone into making sure they make efficient use of available network bandwidth. The most straightforward and widely used method of minimizing unnecessary network traffic is web object caching, where previously accessed documents are stored locally to allow for fast retrieval when requested again.

The predominate strategies for increasing the number of successful cache queries are centralized and distributed web proxy caching. A web proxy simply acts as a network 'go between'. Rather than requesting a web object directly from a remote host, a client may request it from a proxy, which in turn obtains the object itself and forwards the response to the client. Centralized web proxy caching refers to many clients requesting objects from a single caching proxy. Specifically, all web requests and their results can be indexed at the local network's proxy server and immediately returned to other requesting clients. The most important feature of this design for comparison purposes is its cache hit rate. In one sense, the rate is optimal: if the proxy has the object being requested, it will be returned. Also, note that all cache hit returns are one hop in the overlay network sense. That is to say that there are no intermediate servers between the cache server and a requesting client. The

*ACM Membership: 0563072
†{piatek, jackson, juola}@mathcs.duq.edu

request/response transaction also has low network latency since both the proxy and its clients are on the same local network. A centralized proxy topology is shown in Figure 1.

Although centralized proxy caching can significantly reduce external bandwidth use and request latency, it suffers from problems of scalability and reliability. Cache lookups and storage can be computationally expensive and centralized proxy caching does not scale up to the thousands of users present in our target environment, described in Section 3.1. Further, if the centralized proxy server is incapacitated for any reason, web objects are rendered inaccessible to users. That is, centralized proxy caching represents a single point of failure for web object access.

The research community has responded to the problems of centralized proxy caching by developing a variety of distributed web object caching systems. Several of these are referenced in Section 2. The salient feature of these systems is that they alleviate the problems of scalability and reliability by distributing the caching task among many computers. A generic distributed topology is shown in Figure 2.

Although distributed systems solve many of the problems of centralized caching, they are infrequently used in practice. This is because they often attempt to maximize *server* performance at the expense of client performance. We present a system that offers the benefits of distributed proxy caching given the intuitive observation that users will accept proxy performance only if it is as good as performance without use of the proxy.

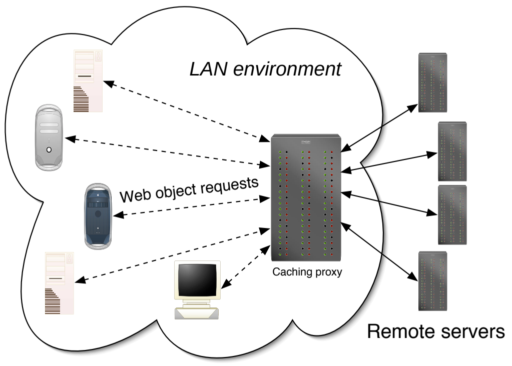

Figure 1: A centralized caching topology

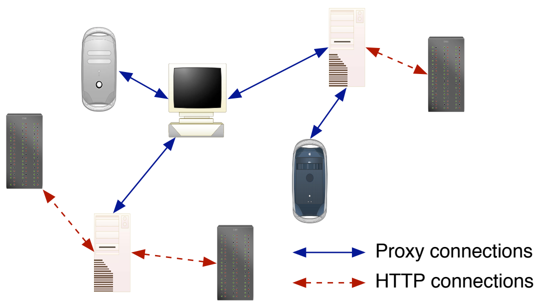

Figure 2: A distributed caching topology

2 Related work

Both centralized and distributed proxy caching have been extensively studied. A thorough investigation of the properties of web traffic as they apply to caching is available in [2]. Cooperative caching is studied formally in [9]. Particularly relevant is the observation that cooperative caching performance benefits are realized only within limited population bounds. We compare our technique with Squid [7], a popular distributed proxy cache. At issue in all caching systems is the consideration of cache expiration and replacement. Cached objects are not valid forever since they are drawn from a dynamic environment. Strategies for discovering the best time to replace objects are referred to as cache expiration and replacement algorithms. This problem is studied extensively in [5].

Although distributed caching handles some problems arising from a centralized strategy, notice that we no longer necessarily have single hop latency between proxy requester and responder. The request may have to be forwarded through several intermediaries before being fulfilled or failing with certainty. Also, it is not immediately clear how to organize the overlay request network topology or how to introduce new hosts into the overlay when no centralized dispatcher exists. This is the problem of bootstrapping the overlay. These are serious research issues in distributed systems and have been studied extensively. The problem of efficient network organization for object retrieval is examined in [8, 11, 6], among others. The bootstrapping problem is studied in [3].

3 Distributed caching in a local environment

We have described generic approaches, both centralized and distributed, to the problem of Internet object caching. Benefits and deficiencies of both have been discussed. We now present our specific strategy for aggregate caching, noting its solutions to the various problems and design issues arising from the previously discussed strategies.

3.1 Characteristics of the local environment

Our strategy for distributed caching is intended to be carried out in a local environment. Before proceeding with a presentation of our technique, discussion of the salient features of the local environment is warranted. Here, the local environment is the set of hosts in a large organization that share a common connection to the Internet and have a high speed interconnect amongst them.

Where our expense metric is time, notice that transfer rate in the local environment is cheap compared to the WAN. Latency is also relatively low. A useful analogy is that of disk algorithms considering memory and disk access. Although the disparity is not as large, disk access algorithms are willing to sacrifice many memory accesses to avoid disk access because of the vastly differing latency and bandwidth of the mediums.

Another noteworthy characteristic of the local environment is the connectedness duration of peers. In a heterogeneous network environment, the duration of host connectivity varies greatly. We note that in the local environment, connectedness is much more permanent, with hosts often constantly connected with fixed addresses whenever they are powered on.

Our approach to proxy caching is a distributed one in the local environment. Connections are performed over TCP. The proxy overlay is organized as follows. Each peer is strongly connected to those peers in its subnet. This connectivity is self organized among the peers using the Zeroconf self configuration IP protocol [10]. As such, this phase of the connection requires no bootstrapping. However, since good cache hit rate requires more peers than would be contained in a single subnet, bootstrapping servers may be designated in each subnet to form a more extensively overlay. These subnet spanning peers have no specific network arrangement amongst themselves. They are strongly connected up to a user specified connection ceiling to prevent overburdening. Notice that subnet spanning peers would require one or more centralized bootstrapping servers, but this limited centralization does not exhibit the problems of total proxy centralization described previously. Observe that once a peer has been bootstrapped to *any* of the existing subnet bootstrapping servers, *all* of these servers would have to go down before requiring a centralized bootstrapping again. This makes load on any centralized server quite low. The described topology is shown in Figure 3.

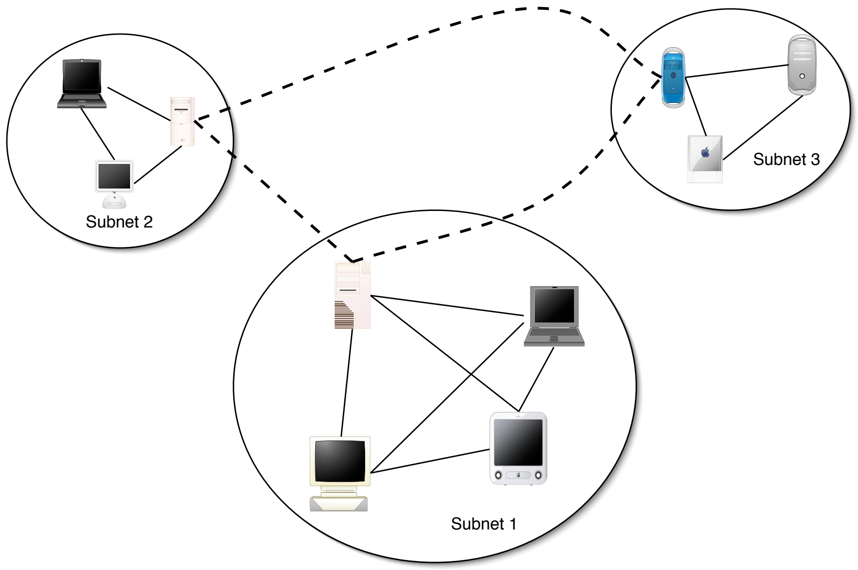

Figure 3: Our topology, where dotted lines represent bootstrapping connections

3.2 Message propagation

Messages on the overlay consist of bootstrap server lists, web object requests, and web object responses. Bootstrap server requests are direct request/response messages to the current subnet bootstrapping server. Each object message is encoded with a time-to-live value (ttl) set by the initial client as well as a global unique identifier (GUID) to prevent duplicate message forwarding. Messages are flooded to the subnet and forwarded from subnet spanning peers as the message's ttl dictates. Message routing is performed with hash table lookup on the GUID.

It is important to note that web object requests are sent to the WAN and proxy network *simultaneously*. The client uses whichever data is returned first. Although this does not reduce WAN bandwidth use in the case of small object requests, it does guarantee web performance using the proxy will be at least as good as without. We discuss this design strategy further in Section 3.5.

3.3 Anonymization and request encoding

In order to prevent all users on the overlay from knowing the web browsing habits of other users, several anonymization measures are inherent in our protocol. First, each object is requested using an MD5 hash of its associated URL. This masks the requested object from all users except those possessing it as it is computationally infeasible to recover the plain-text URL from only the MD5 hash. In other words, if a user recognizes the hash, it is because they have cached the URL. Further, each message can be set with a random ttl to obfuscate how many times it has been forwarded amongst peers. This eliminates possible certainty about the origin of any given object request. Finally, if total anonymity is desired, a user can completely avoid requesting documents from the WAN whatsoever and only request them from the anonymizing proxy overlay. Although, the proxy network may not be capable of fulfilling all web object requests in this case.

3.4 Cache storage and replacement

The local cache for each peer is a single file repository for retrieved objects that has a per-peer configured maximum size and maximum cacheable object size. Compaction occurs when fragmentation inhibits a newly obtained object from being stored in contiguous space. If the peer's local cache still cannot support the new object after compaction, least recently used objects are discarded until sufficient space is available.

Time to cache expiration has a user configurable initial value on a typed file basis. For instance, web documents of type htm, html, shtm, etc. are given a lower initial replacement time than often static document content types such as pdf, swf, and mpg. Replacement values are adjusted using an additive increase/additive decrease strategy. That is to say, if an object is retrieved after cache replacement delay and it is unchanged, the next replacement time is set to the previous replacement time incremented by a constant number of minutes. If the retrieved object is changed as determined by comparison to the existing local cached object, replacement time is decreased. In this fashion, constantly changing, dynamic documents are not kept in cache and static documents are rarely retrieved from the remote host.

3.5 Comparison and discussion

One of the main problems with traditional web proxies such as Squid is the overhead of searching the distributed cache. If the link between network proxies is slow or particular hosts are overburdened, proxy performance can be much worse than simply requesting the document from the original website. Our design makes this concern nonexistent, as documents are simultaneously requested from both the proxy network and the remote host. Further, hosts are unlikely to be overburdened, as they are only responsible for a small number of hosts on their subnet. In the case of large files or busy websites, this provides a bandwidth savings as well since downloads that have been fulfilled by the local network can be stopped.

The main problem with existing distributed proxies is that they are simply not used due to either configuration difficulty or server-centric design. By server-centric, we refer to proxy design that attempts to reduce *server* load rather than improve *client* performance. These strategies lack motivation for adoption. By designing our proxy to be easily configurable and to provide performance no worse than that when using no proxy, we hope to exhibit a more usable distributed caching system.

4 Simulation and results

4.1 Methodology

The most obvious method to evaluate our design is to examine its performance in trace-driven simulations. By trace-driven, we refer to simulating the proxy environment with usage patterns obtained from real world data. We employ the publicly available Berkeley Home-IP traces [4]. Some discussion of the applicability of these traces is in order. They were recorded from roughly 6,000 Berkeley home modem users over a period of 18 days in 1996. Obviously, the major deficiency of these traces is their age. However, notice that they do exhibit several traits required for our purposes that are uncommon to other publicly available web traces. They are traces from a large institutional user pool, which is our target network environment. Further, the traces extend over a period of several days, which is important in evaluating our cache expiration and replacement strategies, which do not reach stable replacement times until several accesses have been made.

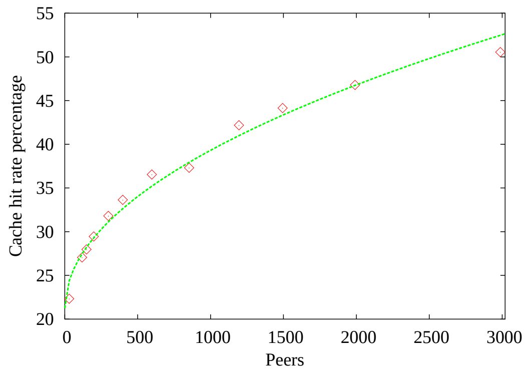

Figure 4: Hit rate vs. peers, infinite peer cache

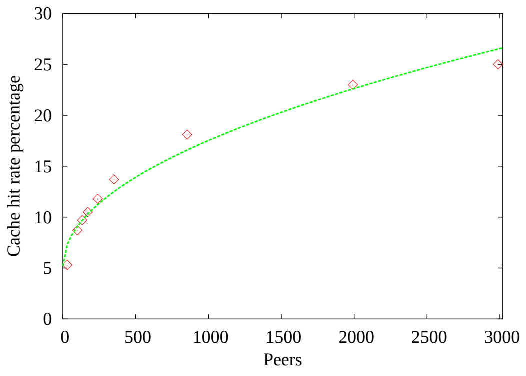

Figure 5: Hit rate vs. peers, 7 megabyte cache

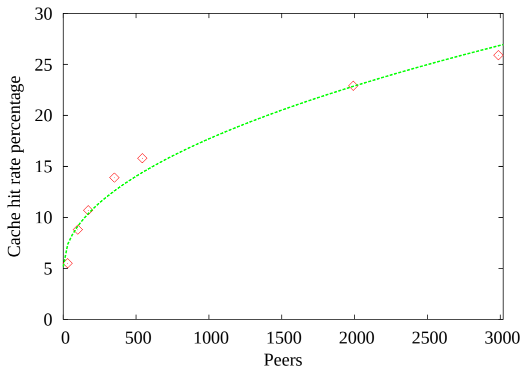

Figure 6: Hit rate vs. peers, random (0-50 megabyte cache)

We evaluate cache hit rate vs. population size in an infinite cache size environment, fixed cache size environment, and a random cache size environment between one and fifty megabytes. We will impose network topology ourselves as none is present in the anonymized simulation data. This is valid as our assumption about the local environment is that latency is negligible as compared to the WAN.

4.2 Results

The most immediate observation from the graphs in Figures 4, 5, and 6 is that cache size significantly affects hit rate over time. Also, as intuitively expected, we confirm here that population size has a significant impact on hit rate. We fit these graphs with square root curves. However, previous studies indicate that as population size increases we can expect cache hit rate to approach 100% of cacheable documents [9].

Previous work has suggested that curve properties for specific user studies are not generalizable. That is to say that properties like rate of cache hit improvement are not constant in different environments. However, the only specific property we must establish from user study is that hit rate makes proxy use worthwhile even in environments with a few hundred or few thousand peers. Our findings suggest meaningful bandwidth savings at these levels.

Also note that another significant benefit for user utility that our system provides is web hotspot relief. Web hotspots and modern responses are studied extensively in [12]. A common commercial, server centric response to web hotspots is the Akamai content distribution system [1]. However, server centric approaches are only effective when mirroring servers are widely deployed and only for expected hotspots. Distributed proxy caching has the added properties of accommodating unexpected hotspots and those without commercial mirroring support. Unfortunately, examining unexpected web hotspots from trace-driven simulations is difficult, as they are unexpected by nature. Further, commercial mirroring services such as Akamai use proprietary algorithms and do not make trace data or replication strategies public. We will have to rely on intuition to evaluate the benefits of distributed local proxy caching in this case.

5 Conclusions

This paper investigates distributed caching in a local environment. Our main observation is that existing strategies for distributed web object caching do not motivate use because they generally attempt to maximize server performance and often cannot guarantee performance for clients. We provide this guarantee by restricting our attention to the local environment. We have used trace-driven simulation to evaluate the performance of our scheme in practice. Our results suggest that cache hit rate in the local environment is highly dependent on population size and cache size. Further, the benefits of distributed caching can be realized for small populations. Although hit rate improves with increased population, there is sufficient benefit to smaller organizations that justifies deployment. These findings, coupled with intuition about the benefits of distributed caching to web hotspots, motivate the adoption of distributed caching in local environments.

References

[1] Akamai. `http://www.akamai.com/`.

[2] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM (1)*, pages 126–134, 1999.

[3] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks.

[4] Steven D. Gribble. UC Berkeley Home IP HTTP Traces, 1997. Available at http://www.acm.org/sigcomm/ITA/.

[5] Stefan Podlipnig and Laszlo Boszormenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, 2003.

[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[7] Squid internet object cache. `http://www.squid-cache.org/`.

[8] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[9] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna R. Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *Symposium on Operating Systems Principles*, pages 16–31, 1999.

[10] Zero configuration network protocol. `http://www.zeroconf.org/`.

[11] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[12] Weibin Zhao and Henning Schulzrinne. Dotslash: A scalable and efficient rescue system for handling web hotspots. Technical Report CUCS-007-04, Columbia, February 2004.