A Taste of Crispy Squid

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

Michael Rabinovich
AT&T Labs - Research
108 Park Avenue
Florham Park, NJ 07932
misha@research.att.com

Abstract

Distributed proxy caches are in use throughout the world to reduce access latency and bandwidth demands for Internet object transfer. The CRISP project seeks to build more effective distributed Web caches by exploring alternatives to the hierarchical structure and multicast handling of probes common to the most popular distributed Web cache systems. CRISP caches are structured as a collective of autonomous Web proxy servers sharing their cache directories through a common mapping service that can be queried with at most one message exchange. Individual servers may be configured to replicate all or part of the global map in order to balance access cost, overhead and hit ratio, depending on the size and geographic dispersion of the collective cache.

We have prototyped several CRISP cache structures in *Crispy Squid*, an extension to the Squid Internet Object Cache. We are evaluating these cache structures using *Proxycizer*, a full-featured package for replaying traces of observed request traffic into Web cache prototypes.

1 Introduction

Hierarchical Web proxy caches descended from the Harvest project [3] are now in wide use. Popular Harvest descendents include the commercial product NetCache [4] and the public-domain Squid caches [20]. Hierarchical Web caches are built as a collective of independent proxy caching servers that share their cache contents through a simple multicast query mechanism using the Internet Cache Protocol (ICP) [19]. The National Laboratory for Applied Network Research has engineered a structure of Squid caches spanning the United States, and is coordinating with participants across the world to build a "global mesh" of Web proxy caches.

However, our experiments have shown that hierarchical caches have scalability and coverage problems that manifest themselves in large cache configurations [9]. Other researchers have confirmed these observations [17, 6]. The CRISP project is exploring alternative Web cache structures that provide more scalable and efficient structures for large, geographically dispersed collective Web caches. The key idea behind the CRISP Web cache architecture is to supplement each caching server's local cache directory (i.e. the index of all the documents the server holds locally) with an independently managed *global directory* that allows it to locate objects cached at other servers. This frees the individual caching servers from responding to directory queries, and allows the global directory to scale independently of the caches themselves.

A CRISP cache (shown in Figure 1) consists of a set of *caching servers* (proxies), which directly field requests from clients/browsers, and a *mapping service*, which is responsible for maintaining the global directory of objects stored in the caching servers. On a local cache miss, each caching server queries the mapping service to determine if the object is resident elsewhere in the collective cache. If the mapping service reports a hit, then the object is fetched directly from the peer caching server. Caching servers keep the global directory current by reporting object fetches and evictions to the mapping service.

The key design issue for a CRISP cache is the structure of the mapping service. Several factors influence the placement of proxies for a particular environment; likewise, different proxy organizations dictate different structures for the mapping service.

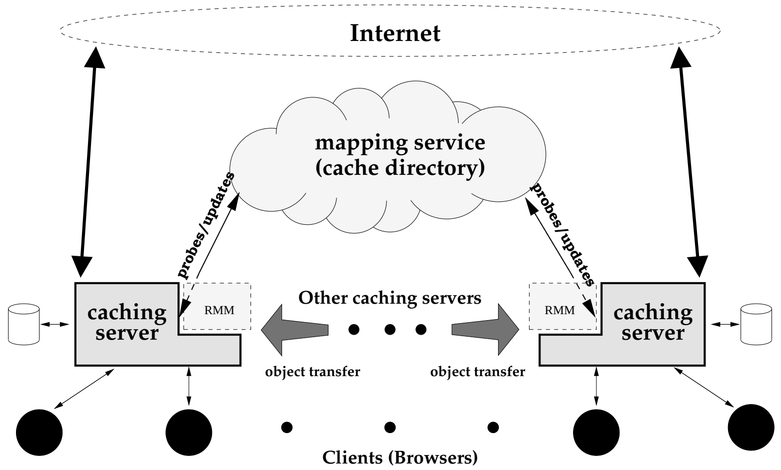

Figure 1: The CRISP cache architecture.

The first CRISP prototype [10] was based on a set of independent *mapping servers*, each maintaining an exclusive portion of the global directory. This model is well-suited to typical ISP environments in which proxies are "close" to each other with respect to network latency, and the cost of a mapping service probe is small compared to the total perceived client latency. In this paper we describe extensions to this structure that replicate some or all of the global directory in order to reduce query latency in large cache configurations.

As a basis for evaluating the CRISP cache architecture, we implemented several CRISP variants in *Crispy Squid*, an extension to a recent Squid release. Crispy Squid introduces a configurable CRISP mapping service to replace the multicast ICP probes used in Squid and other Harvest-derived caches. Crispy Squid caches can deliver hit ratios equal to or better than ICP caches, with lower query costs for both hits and misses. Moreover, a replicated global directory can reduce query latency further by removing network communication from the critical path of probes, at the expense of higher overhead to store and manage the replicas. Crispy Squid supports several alternatives for managing replicated directories that exploit the observed properties of Web access to reduce this overhead, while delivering close to ideal hit ratios.

Our primary approach to evaluating Web cache prototypes is to subject them to traffic loads corresponding to real Web access traces, which are now widely available. To support our experiments we have developed a suite of tools called *Proxycizer* for replaying Web traces under controlled conditions. Proxycizer accommodates multiple trace formats, allows a choice of request timing properties for average-case or worst-case analysis, and can emulate a variety of Web server characteristics.

This paper is organized as follows. Section 2 summarizes the problems with hierarchical ICP caches, and outlines a range of CRISP alternatives that respond to the factors influencing Web cache design. Section 3 describes the implementation of configurable CRISP caches in Crispy Squid. Section 4 describes our experimental methodology and the Proxycizer package for evaluating performance of Web caches by subjecting them to trace-driven load. We conclude in Section 5.

2 Directory Structures for Collective Internet Caches

Efficient directory management is the key design problem for any Internet cache. Several considerations influence the design choices:

- *Load on the internal network.* Important to almost all configurations, internal network usage affects link contention and congestion and the potential for lost packets on the network connecting the caching servers, e.g., an ISP's internal network.
- *Costs of external network usage.* Web caches exist in part to reduce bandwidth demands on

the external network. One reason to deploy Web caches is altruistic: if everyone is a "good citizen", then everyone benefits from reduced congestion on backbone links and Web servers. A second reason is that caches can directly affect access costs when Internet access charges are metered by traffic volume.

- *Object fetch latency.* From the perspective of an individual client, object fetch latency is the most important motivation for using a Web cache, and the only real measure of its success.

- *Latency of the internal network.* As the network distance between the servers in a collective cache increases, synchronous network exchanges with remote parts of the cache are no longer practical. Geographically dispersed caches must localize query traffic by replicating the directory or reducing the "coverage" of queries.

The following subsections explore several directory structures and evaluate them with respect to these considerations.

2.1 Hierarchical ICP Caches

Squid and other Harvest-derived hierarchical caches maintain no directory apart from each caching server's directory of its own contents; the global directory consists of the aggregate of the local directories. Caches are organized in a loose hierarchy: to probe the collective cache, a caching server multicasts an ICP message to its neighbors (siblings) and parents, which respond with a HIT only if they have a local copy of the requested object. If the request misses or times out, a selected parent continues the probe by repeating the process, multicasting to its neighbors and parents. If the request misses at each level of the hierarchy, then a top-level parent fetches the object from the source, caches it, and passes it back down the hierarchy.

This approach has several drawbacks:

- The "cache coverage" available to any caching server does not include the contents of its descendents or its "cousins" and their descendents. This is because queries only travel up the hierarchy, never down. This can result in a large number of false misses unless the parents are large enough to replicate most of the objects held by their descendents.

- The directory probe mechanism is combined with object transfer; fetched objects may pass through several intermediate caching servers on their way to the original requester. Tewari et al. [17] show that this leads to remarkably high latencies in some configurations, motivating shallower and "bushier" hierarchies. It also places unnecessary load on high-level servers that field the cache misses from all of their descendents.

- Multicasting can increase the probe load on all caching servers to unmanageable levels. Our studies [9] showed that a three-level hierarchy of caches using ICP can send an order of magnitude more query traffic through the network than a CRISP cache of equivalent size equipped with a central mapping server.

- Lost ICP messages or busy neighbor caches increase miss latencies. A caching server must wait for all neighbor caches to respond with a miss (or timeout) before directing the request upwards through the hierarchy. Bushier hierarchies increase the probe load on all children of a common parent, and increase the probability that some caching server will time out on a query. Our studies showed that ICP message drops and timeouts contribute to higher miss latencies, even in small configurations.

These problems motivate "flat" cache structures with a scalable directory structure that provides a more complete index of the collective cache's contents.

2.2 CRISP structures

The CRISP cache architecture resolves these problems by managing a separate global directory of the collective cache contents. A caching server probes the cache with at most a single unicast message exchange to a mapping server, which may be colocated with the caching server. If the probe is a HIT, the object is fetched directly from the peer that caches it. If the probe is a MISS, the requesting proxy fetches the object directly from the Web site.

The following subsections describe the three basic directory configurations we have prototyped with Crispy Squid.

2.2.1 Partitioned Synchronous Directory

In this structure, the mapping service consists of several mapping servers, each assigned some exclusive subset of the URL-space using a hash function in a manner similar to the Cache Array Routing Protocol (CARP) [18]. Caching servers synchronously query the assigned mapping server when a client's request results in a local cache miss, and notify the appropriate mapping server when an object is added to or evicted from the local cache.

The degenerate case with one mapping server is the structure implemented and evaluated for the initial CRISP prototype. A CRISP mapping server on a 266 MHz AlphaStation 500 system with 100 Mb/s Ethernet yielded an average query latency of less than 3 ms at a request rate corresponding to approximately 85,000 clients (estimated by determining the average per-client access rate seen at an AT&T proxy during peak demand periods) [10]. In fact, the mapping server has little effect on access latency since it merely performs a hash lookup on an in-memory directory. A muchlarger share of access latency appears at the caching server where queued requests are waiting to be serviced and competing for resources with TCP connections in progress.

The advantage of these configurations is that they always reflect the complete contents of the collective cache, yielding ideal hit ratios for a cache of a given size. Their key limitation is that the query to the mapping server lies on the critical path of object requests that miss in caching servers. This requires that caching servers be close enough to all mapping servers that the latency of the query is imperceptible to the user. A tightly-coupled cluster of proxies or an ISP with several proxies on an internal network could use these structures successfully.

2.2.2 Replicated Asynchronous Directory

In geographically dispersed CRISP caches, the network latency to query any shared mapping server will be unacceptable to some proxies, regardless of where that mapping server resides in the global cache. We have implemented CRISP variants that asynchronously replicate the global directory. Replicas can be placed closer to a group of caching servers, or even colocated with each caching server. In the latter case queries are satisfied by a simple local table lookup, removing a network exchange from the critical path of cache misses. In a sense, the replicated directory structure is the opposite of the hierarchical ICP caches: rather than multicasting queries and generating no updates to the map, this structure multicasts the updates and generates local queries.

The key advantage of multicasting updates over multicasting queries is that the updates to the map can be propagated asynchronously, whereas queries are by their nature synchronous. Updates to CRISP directory replicas are propagated through the mapping service using an asynchronous protocol. Note that the replicas may be weakly coherent, since inconsistencies do not compromise correctness of the cache, only its performance. In particular, false hits in the directory are detected on access to the caching server, while false misses simply cause an extra Web server access for an object that could have been fetched from the cache.

Asynchronous propagation of directory updates enables the collective CRISP cache to cover larger network distances without affecting miss latency. For example, proxies may batch updates to the map and propagate them with periodic anti-entropy exchanges using an efficient stream protocol [15]. Our current prototype batches updates into TCP streams sent to a central distribution node at (configurable) periodic intervals, in order to amortize network connection costs over a group of updates. On typical traces, propagation intervals of up to an hour cause hit ratios to drop by at most 5%.

One problem with this structure is that map replicas increase the storage requirements for the cache. If the size of the map becomes cumbersome, a map compression scheme such as *Summary Cache* [6] may be used, although our prototype does not support this.

2.2.3 Replicated Partial Directory

Replicated Partial Directory seeks to further reduce the cost of replicating the cache directory, but without compromising hit ratios significantly. The idea is to identify a subset of the cache contents that is

Method	Description
`init()`	Initialization
`touch_c()`	Called on a local cache hit, to identify when an object was accessed.
`query_c()`	Called on a local cache miss, used typically to query the local map or a remote mapping server.
`advertise_c()`	Called when a cacheable object has completed its transfer into the local cache.
`unadvertise_c()`	Called when a cacheable object is discarded from the local cache.
`query_s()` `advertise_s()` `unadvertise_s()`	Called on receipt of a query/advertise/unadvertise message for an object from a remote caching server.

Table 1: Methods of the Crispy Squid map module.

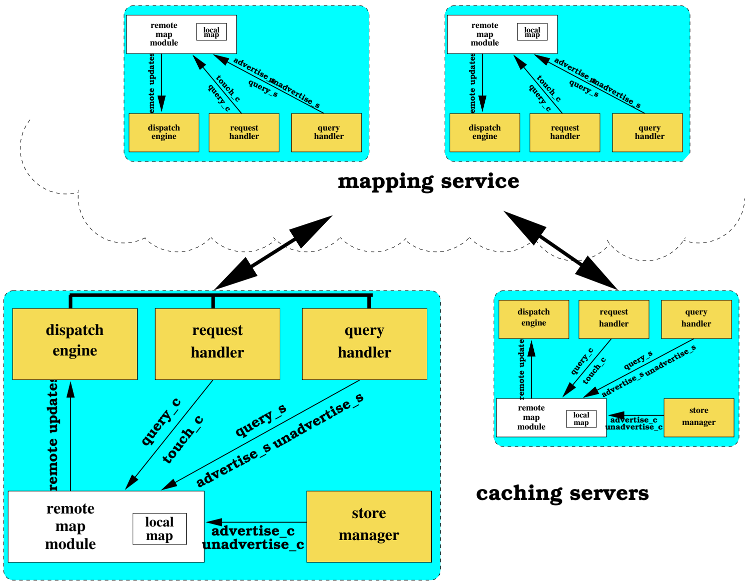

Figure 2: Crispy Squid. Non-shaded modules are extensions to Squid.

most likely to yield productive hits, and replicate only those entries (called the *submap*) at each mapping server. One approach we have investigated is to restrict the submap to only "shared documents", those that have already been accessed by more than one client. On typical Web traces from DEC WRL this configuration reduced the size of the map replicas by 70% while sacrificing potential cache hits on less than 6% of all references. This is because only about 30% of documents are shared, but many shared documents are widely shared.

Note that partial directory replicas can be used to supplement rather than replace central directories. One approach is to check the submap replica first to speed up the majority of hit cases, but fall back to querying a central mapping server for objects that miss in the submap. This approach delivers hit ratios and miss latencies comparable to the synchronous directory configurations, but with lower average hit ratios.

Recently we have begun to evaluate a variant of Replicated Partial Directory called *neighbor cache* in which each submap contains entries for only those objects that are closer than some proximity threshhold. Neighbor cache does not yield hits for objects resident in caching servers that are "too far away" from the requesting server. However, it is the most scalable of all configurations, and widely shared objects eventually propagate through the entire cache. This is our preferred approach for constructing collective caches that are highly dispersed geographically.

3 Implementation of Crispy Squid

We have modified both version 1.1.20 and version 1.2.beta20 of the Squid cache to replace its hierarchical multicast ICP probes with CRISP directories in a variety of configurations. The extensions are localized in a new *remote map* module added to the Squid implementation. Figure 2 illustrates the structure of the resulting system, *Crispy Squid*.

Any Crispy Squid server can be configured to act as a mapping server, a caching proxy server, or both. Each mapping server can be configured to maintain some or all of the global map, according to the policies described in Section 2.2. The configuration is specified with new fields in the Squid configuration file, and captured at startup time within the remote map module, which handles all incoming and outgoing queries on the cache directory. Figure 2 shows the relationship of the remote map module to other modules in the system.

In some configurations, the remote map module may maintain a local replica of some or all of the global directory. This replica is referred to as the *local map*. The local map is distinct from the local cache directory maintained by the store manager; the former is a local map of remotely cached objects, while the latter is a local map of locally cached objects. Table 2.2.2 lists the methods used to query the map and notify it of changes to the local cache contents; the configuration determines the behavior of these methods, e.g., whether a particular query is directed at the local map or a remote mapping server.

This structure is general enough to implement several directory configurations (including Squid's default policy), while placing most policy-specific code in one location and disrupting as little of the original Squid machinery as possible.

4 The Proxycizer Tool Suite

This section gives an overview of the tools we have developed to evaluate the performance of the various directory structures implemented in Crispy Squid.

Our approach to evaluating performance of Web cache structures is rooted in the wide range of proxy traces available today. Because these traces are indicative of current real-life proxy usage patterns, they are a powerful basis for predicting performance of collective cache structures in actual use. [13, 16] Used in conjunction with continuous online trace gathering, trace-based performance studies can track the effects of Web usage patterns as they evolve.

However, we have found that the tools available to analyze these traces are inadequate for our needs. Therefore, we developed a collection of tools to process trace files and generate trace-driven Web access traffic for testing proxy caches. We later added tools to simulate the behavior of proxy caches under workloads driven by various proxy traces. We have consolidated these programs into a carefully-designed suite of tools and reusable libraries called *Proxycizer*. [7] *Proxycizer* is written in C++, will compile with any recent version of `g++` (as well as many vendor-specific compilers) and has been tested on platforms running FreeBSD, Digital Unix, and Solaris.

The `simclient` and `webulator` make up the "proxy exercising" component of *Proxycizer*. `simclient` generates a request stream to a proxy from a log file. It can wait a specified delay between requests, and/or simulate several simultaneous clients each reading from separate log files. `simclient` can also generate a "real-time" request stream based on timestamps given in the log file. This feature can be used to accurately replay a specified trace in order to replicate a traffic pattern seen in the past.

`webulator` provides the server side of the testing infrastructure. `webulator` will respond to simple HTTP requests and return HTTP replies (content-filled with '*') of a size specified by a previously constructed on-disk database. `webulator` can also use duration information stored in the database to delay the return of data for a given amount of time (keeping the connection open while other connections are serviced).

Proxycizer provides an abstract representation of Web trace entries, accessed through a strict procedural interface, to enable maximum code reuse for analysis of traces from different locations. The abstract log entry class currently exports common fields including request time, request duration (both at microsecond granularity), reply size, HTTP method, and URL, which is itself an abstract type that has both character string and numeric forms for extremely fast comparisons in those traces that provide numeric URLs. *Proxycizer* automatically senses the log format and selects the correct filter. The currently supported log formats are Squid access logs [20], Crispy Squid access logs [8], Harvest access and hierarchy logs [3], UC Berkeley Home IP traces [11], DEC traces [12], and `simclient` logs. *Proxycizer* also implements several proxy simulator classes, including those for Harvest and the CRISP cache architectures described in Section 2.

Several other useful packages have recently been made available, each addressing different aspects of Internet server simulation and benchmarking. While Proxycizer is purely trace-driven, the Wisconsin Proxy Benchmark (WPB) [1] uses synthetic workloads modeled to emulate typical temporal locality patterns to test proxies under load. hbench:Web [14] is a benchmark for Web servers that is not explicitly trace-driven, but does preprocess Web server logs and automatically generates an appropriate traffic model for driving a Web server. S-Clients [2] introduces a scalable mechanism for driving Web servers to overload conditions. Squid Proxy Analysis (SPA) [5] provides trace-driven simulations that are similar in nature to our simulation programs, but are tailored to emulate the replacement behavior of Squid caches.

5 Conclusion

Distributed proxy caches are an important tool for reducing Internet congestion and access latency. This paper gives an overview of research on distributed Web proxy caches in the CRISP project. The CRISP Web cache architecture is designed to allow the servers in a distributed cache to cooperate in an efficient and scalable way, by introducing a *mapping service* that maintains a global directory of objects in the cache. The mapping service consolidates the responsibility of responding to directory queries into an entity independent of the proxy caches themselves, enabling the mapping service to scale independently from the participating proxy caches.

We have described several alternative mapping sevice structures implemented in Crispy Squid, a modular extension to the popular Squid Cache. We have also described the *Proxycizer*, a general tool for empirical studies of Web cache structures under real observed workloads and controlled conditions.

6 Availability

Crispy Squid is available from `http://www.cs.duke.edu/ari/crisp/`.

References

[1] Jussara Almeida and Pei Cao. Wisconsin Proxy Benchmark 1.0. Available at `http://www.cs.wisc.edu/~cao/wpb1.0.html`.

[2] Gaurav Banga and Peter Druschel. Measuring the capacity of a Web server. In *Proceedings of The USENIX Symposium on Internet Technologies and Systems (USITS)*. Department of Computer Science, Rice University, December 1997.

[3] 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.

[4] Internet Middleware Corporation. Harvest cached-3.0 user's manual. San Jose, California.

[5] Brad Duska, David Marwood, and Michael J. Feeley. The measured access characteristics of World-Wide-Web client proxy caches. In *Proceedings of The USENIX Symposium on Internet Technologies and Systems (USITS)*, December 1997.

[6] Li Fan, Pei Cao, Jussara Almeida, and Andrei Broder. Summary Cache: A scalable wide-area Web cache sharing protocol. In *Proceedings of ACM SIGCOMM98*, September 1998.

[7] Syam Gadde. The Proxycizer Web proxy tool suite. `http://www.cs.duke.edu/ari/Proxycizer/`.

[8] Syam Gadde, Jeff Chase, and Michael Rabinovich. Crispy Squid. Available at `http://www.cs.duke.edu/ari/crisp/`.

[9] Syam Gadde, Jeff Chase, and Michael Rabinovich. Directory structures for scalable Internet caches. Technical Report CS-1997-18, Department of Computer Science, Duke University, November 1997.

[10] Syam Gadde, Michael Rabinovich, and Jeff Chase. Reduce, Reuse, Recycle: An approach to building large Internet caches. In *Proceedings of The Sixth Workshop on Hot Topics in Operating Systems (HOTOS-VI)*, pages 93–98, May 1997.

[11] Steven D. Gribble. UC Berkeley Home IP HTTP traces, July 1997. Available at `http://www.acm.org/sigcomm/ITA/`.

[12] Thomas M. Kroeger, Jeff Mogul, and Carlos Maltzahn. Digital's Web proxy traces. Available at `ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html`.

[13] Carlos Maltzahn, Kathy J. Richardson, and Dirk Grunwald. Performance issues of enterprise level Web proxies. In *Proceedings of The ACM SIGMETRICS Conference on Measurement and Modeling of Compter Systems*, pages 13–23, June 1997.

[14] Stephen Manley, Michael Courage, and Margo Seltzer. A self-scaling and self-configuring benchmark for Web servers. Available at `http://www.eecs.harvard.edu/~vino/web/hbench-web/`.

[15] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer, and Alan J. Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 288–301, October 1997.

[16] Alex Rousskov. On performance of caching proxies. Available at `http://www.cs.ndsu.nodak.edu/ rousskov/research/cache/squid/profiling/papers/`.

[17] Renu Tewari, Michael Dahlin, Harrick Vin, and John Kay. Beyond hierarchies: Design considerations for distributed caching on the Internet. Technical Report TR98-04, Department of Computer Sciences, University of Texas at Austin, February 1998.

[18] Vinod Valloppillil and Keith W. Ross. Cache Array Routing Protocol v1.0. Internet-Draft, June 1997. Available at `http://www.etext.org/Internet/Internet-Drafts/draft-vinod-carp-v1-03.txt`.

[19] Duane Wessels and K Claffy. Internet cache protocol (ICP), version 2. Internet-Draft, May 1997. Available at `http://ircache.nlanr.net/Cache/ICP/`.

[20] Duane Wessels et al. Squid Internet Object Cache. `http://squid.nlanr.net/`.