Directory Structures for Scalable Internet Caches

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

October 3, 1997

Abstract

Use of Internet caches is a cheap and effective way to improve performance for all Internet users. Distributed caches offer the potential to serve larger user communities and to deliver higher hit ratios on shared Web documents. The key to building effective distributed caches is a directory structure that allows individual caching servers to locate objects cached at neighboring sites, combining them into a logically unified collective cache.

This paper uses Web traces to evaluate a range of alternatives for managing directories in distributed Internet caches. We use trace-driven executions and simulations of prototype caches to compare multicast-based queries of local maps (Harvest) with unicast queries of a global map (CRISP). We then use properties of the traces to predict performance of CRISP variants in which the global map is partitioned or replicated. Finally, we propose a novel lazy CRISP structure based on weakly consistent replication of the most valuable subset of the global map. We show that this scheme combines the best aspects of several distributed cache structures, while maintaining a near-ideal hit ratio.

1 Introduction

Ever since the introduction of HTTP traffic onto the Internet, bandwidth demands have been increasing dramatically. Growth pressure will continue with new applications and technologies, e.g., as ISDN lines and cable modems replace slow modems.

Caching proxies [14] are an essential tool for controlling Web user demands on Internet bandwidth and HTTP server capacity. Simple proxy caches are already highly effective for accesses to cacheable, static documents, which make up 89% of references (as we show in Section 2) in recent traces. The effectiveness of proxy caches will improve further with new techniques for maintaining coherency [10, 13] and caching dynamic documents [11, 7].

This paper explores *distributed* Web cache architectures, in which a group of proxies (*caching servers*) together form a "collective" Internet object cache that can serve more users over a wider area than any single proxy. The purpose of a collective Web cache architecture is to deliver the benefits of a shared cache to larger communities of users, while distributing the load among the individual caching servers. Similar caching schemes have been shown to be useful for file systems and/or virtual memory [1, 8].

We focus on collective architectures in which each proxy makes independent local decisions that determine how its cache space will be used to serve its primary users. In particular, this paper does not consider the impact of coordinated or "cooperative" replacement policies, e.g., global LRU replacement [8] or forwarding of evicted objects to neighboring caches [5]. Cache architectures that manage resources locally are in the spirit of other successful distributed Internet services (e.g. DNS), and are an appropriate starting point for Web caches that cross departmental or organizational boundaries. To contrast this class of caches with more closely coupled architectures with coordinated replacement, we refer to the former as "collective caches".

Several instances of collective cache architectures have been proposed and some are in use, including the Harvest cache [3], its successor Squid [19], and the Duke/AT&T CRISP cache [9]. A fundamental question for such architectures is how individual proxies locate objects held by other proxies, in order to service local misses from neighboring caches rather than from home HTTP servers on the Internet. Harvest and Squid use multicast queries to neighbor caches for this purpose. CRISP relies on a central map that tracks location of all cached objects directly.

In this paper, we study the tradeoffs between multicast- and directory-based collective caches and explore the scalability, effectiveness, and cost of a range of schemes for maintaining the directory, using the proxy trace [12] released by researchers at Digital Equipment Corporation. This trace consists of more than 24 million document accesses by 17,354 users over the 25 days from 8/29/96 through 9/22/96.

This paper makes three contributions:

- In Section 2, we present further evidence supporting deployment of collective Web caches serving large user communities, based on the access characteristics of the 25-day trace.
- In Section 3 we use trace-driven executions of Harvest and CRISP prototypes to motivate a global cache directory (*map*) as the means of locating objects in the collective cache.
- In Section 4 we consider fundamental alternatives for managing a global directory for a collective Web cache.

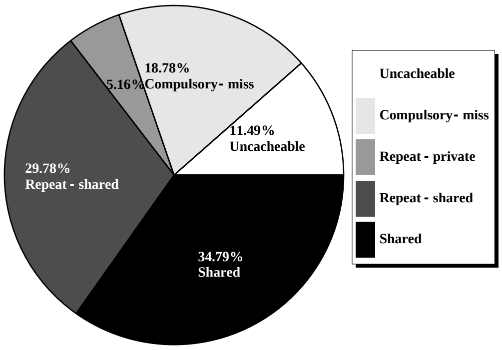

Figure 1: Reference breakdown for 25-day trace.

We show how to improve CRISP scalability and reduce latency of directory probes by replicating, partitioning, and/or distilling the map. We propose a novel lazy variant of CRISP, and show that it can achieve absolute hit ratios with less than a 6 point drop in hit ratio of the original CRISP prototype for the 25-day trace.

2 Why Collective Caches?

In this section we present reference studies of the 25-day DEC trace to motivate the collective architectures explored in this paper.[1] Figure 1 shows that almost 89% of the the references in this trace are to static documents that can be cached. About 70% of all references are *potential hits*; the purpose of any Web caching architecture is to serve as many of these accesses from the cache as possible at low cost.

The trace data provides further evidence to support a central premise of our work, that the highest hit ratios are achieved by shared caches serving large user communities:

- About 93% of all potential hits are to documents that are accessed by more than one client. About half of these are references issued by clients that have never accessed the document before. We refer to these as *shared* references; they are distinct from *repeat-shared* accesses, which reference shared documents previously accessed by the same user. Shared references can never be absorbed by per-user browser caches of any size. The high percentage of shared references presumably reflects filtering of the accesses by browser caches in the user community.

- While private caches of sufficient size can yield temporal hits for repeated references to a document by a given

[1] All our studies except those using real web proxies use a basic LRU replacement scheme; the operation of real proxies may differ somewhat when objects are invalidated due to age, HTTP expiry indicators, etc. - some of this information is not available in the trace.

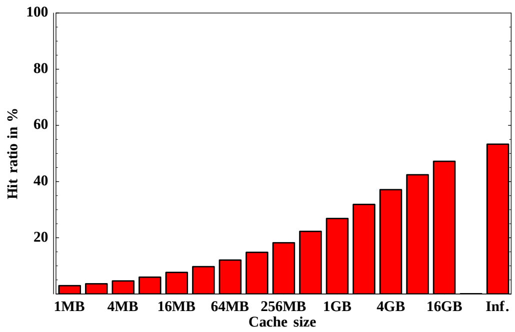

Figure 2: 25-day hit ratios for shared LRU caches of various sizes, with ideal per-user caches.

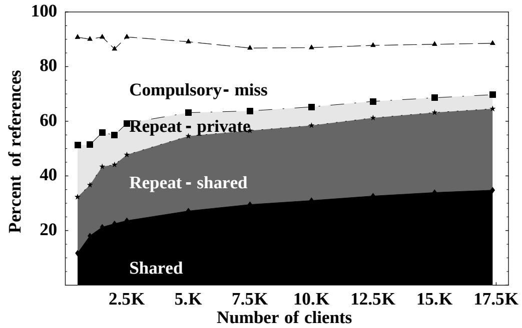

Figure 3: Reference breakdown for varying client populations. These are averages of four random client subsets.

user (the 35% of traced references in the *repeat* categories), they cannot produce hits for any of the 35% of references in the *shared* category. However, Figure 2 shows that supplemental shared caches of practical size can deliver the majority of these potential hits. Even assuming perfect per-user caches that filter out all potential temporal hits, a monolithic 16GB shared cache can deliver a 50% hit ratio for the remaining references using simple LRU replacement, significantly reducing external bandwidth demands to service the trace. The best achievable hit ratio at the proxy, assuming infinit per-user caches, is 53%.

- The percentage of references that are to shared documents increases with the size of the client community. Figure 3 shows reference breakdowns for randomly chosen subsets of the client community in the 25-day trace. We see that caches serving larger user communities can exploit more of the sharing that exists in the full trace.

Although the 25-day trace shows a clear benefit to large, shared caches, experience has shown that the scalability of individual proxy servers is limited. While we can expand proxy server capacity by scaling the hardware and streamlining the software, a simpler and ultimately lower-cost solution is to install additional proxy servers for portions of the reference stream. The following subsections consider three basic choices for structuring distributed Web caches.

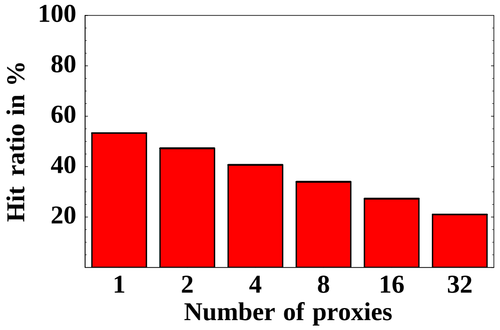

Figure 4: Best achievable 25-day hit ratios for isolated proxies, with round-robin client assignment and ideal per-user caches.

2.1 Option 1: Isolated Proxy Caches

It is easy to build a scalable Web cache by distributing users across a collection of isolated caching servers. However, this structure sacrifices most of the benefits of a shared cache. Figure 4 shows the best hit ratios achievable by such an "uncooperative" collective cache as a function of the number of independent caching servers, with balanced round-robin client assignment. As in Figure 2, these shared proxies supplement ideal per-user caches.

Adding isolated proxies to serve a given client population causes hit ratios to decline because it decreases the probability that a client accessing a shared object for the first time can benefit from another client's previous reference to the object. A collective cache of 32 isolated proxies (540 clients per proxy) can yield hits for less than half of the shared references; the absolute achievable hit ratio for the trace drops to 21%, down from 53% for a monolithic shared cache.

2.2 Option 2: Partitioned Cache

A straightforward way to build a collective cache that avoids the problems with isolated proxies is to partition the document set across the servers by "homing" each document on a specific proxy using a URL hashing function [17]. We refer to this structure as *Partitioned Proxy Cache* (PPC). A similar cache organization is used in NetCache [4] (the commercial continuation of the Harvest project). With a good hashing function, PPC can deliver comparable hit ratios to a monolithic shared cache, while balancing load among the proxies. PPC is supported by a *proxy autoconfiguration* [2] feature in popular browsers, in which the browser executes a user-provided script to select the proxy to query for a particular reference. Netscape's facility can be programmed to mask failures using a timeout/failover/retry mechanism.

Pure PPC suffers from several weaknesses that motivate the more sophisticated schemes in this paper. First, it relies on a hash function that produces balanced distributions of objects and references among the proxies. This is difficult to achieve in practice, and it may be necessary to adjust the proxy sizes to match observed reference patterns, or to run the cache below capacity to minimize queuing delays at the most heavily loaded caching servers. Second, all participants must be made aware of a new hash function if proxies join or leave a PPC cache, and cache performance will degrade temporarily as the cache accomodates the new caching function through flushing, reorganizing, or rebuilding the contents of the cache.

Most importantly, PPC may impose high latency and network overhead for both hits and misses, since the proxy handling an access is predetermined by the referenced document, independent of the user issuing the reference or that user's location in the network. This is a problem primarily if the individual proxies are dispersed in the network, as we envision for caches serving large user communities.

2.3 Option 3: Collective Cache with Remote Queries

The third option is to partition the clients into groups, each bound to an autonomous caching server as in option 1, but also to supplement the collective with a remote querying mechanism. The purpose of the query mechanism is to bridge the gap in hit ratios between monolithic shared caches (Figure 2) and uncooperative collective caches (Figure 4), balancing the benefits of a shared cache with the scalability of isolated proxy servers.

On a local cache miss, a participating server probes the collective cache for the missing document before contacting a Web server to service the miss. This structure presents an opportunity to reduce latency and network traffic by (1) placing each proxy along the network path between the users it serves and the wide-area network link, and (2) placing the objects most likely to yield hits into caching servers closest to the users most likely to reference them. Moreover, the autoconfiguration mechanism can be used to implement a dynamic binding of clients to proxies in order to balance load or mask proxy failures.

All of the collective cache architectures considered in the remainder of this paper use this option. At issue is the question of which query mechanism delivers the best hit ratios with the lowest cost and network overhead. The next section compares two fundamental implementation choices for queries in collective caches.

3 Harvest and CRISP

In this section we compare the latency, hit ratios, and network overhead of functioning implementations of two collective Web caches in typical configurations. We also use simulation to predict the performance of larger configurations. The first system, Harvest [3], along with its successor Squid [19], is in widespread use. We proposed CRISP [9], the second system, as a simple alternative to Harvest. The CRISP prototype was implemented by modifying the last publicly available Harvest code base (version 1.4pl3).

Harvest and CRISP define two fundamental alternatives in the querying mechanism for locating objects in the collective cache. In Harvest, each server probes the cache by multicasting the requested URL to some or all of its neighbors. In contrast, CRISP maintains a global directory that is separated from the caching servers and probed with a unicast message exchange. Harvest avoids overheads to store and maintain a separate global directory while CRISP is easy to configure, generates minimal network traffic for cache probes, and can deliver as high hit ratios with smaller aggregate cache sizes. Harvest can continue to deliver some remote hits if a caching server fails, but CRISP reverts to isolated proxies if the global directory is unavailable.

This section compares three configurations of a Harvest cache using multicast-based probes with two CRISP cache configurations using directory queries. We first describe the Harvest and CRISP algorithms and prototypes in more detail, then describe our methodology and present our experimental results.

3.1 Harvest

Harvest and its successor Squid are hierarchical caches intended to be placed at network junctures along the paths from clients to Web servers. Each caching server is configured to interact with some subset of its peer servers as *parents*, *children*, or *siblings*. Children are typically "lower" (i.e., closer to the users) in the network than their parents, and siblings are usually children of a common immediate parent, but any DAG is legal. We consider only tree configurations, which are the intended Harvest topologies.

Each server satisfies misses to its local cache by multicasting a query for the requested document to all immediate parents and siblings, if any. The implementation uses a logical multicast (i.e., it repeats the unicast message to all the recipients), rather than an IP multicast (e.g., [6]). The requesting server then fetches the document from the first proxy that responds with a HIT. If there is no hit, it fetches the document through the first parent that responds with a MISS. If it has no parents, or no parents respond within a timeout interval, it fetches the object from the home Web site. Every fetched document is cached locally.

If the request is forwarded to a parent, the query process is repeated at the next level of the hierarchy. For example, in case of a miss, the parent multicasts the query to its siblings and waits for a response before forwarding the request up the hierarchy.

3.2 CRISP

The key idea behind CRISP is to avoid multicast queries by maintaining a global map of the collective cache, accessible to each caching server with a single exchange. In [9] we outlined the simplest CRISP structure in which a central mapping service tracks the contents of the cooperative cache and coordinates some aspects of location, replacement, and update management. We argued that the well-known difficulties with this structure are less significant for Web caches because (1) the cost to store and query the map is small relative to the cost to store and transfer documents, (2) the query latency for a mapping server with tens of thousands of users is imperceptible to humans, and (3) failures of the map, which are unlikely, do not interrupt service but only degrade performance.

We modified the Harvest code base to implement a CRISP prototype with a single central mapping server. We refer to this prototype as CRISP/CSD (Central Synchronous Directory). A CRISP/CSD process can be configured to act as either a mapping server or as a caching server with a single parent (its mapping server). Modifications to the Harvest code included adding new mapping service message types to the Internet Cache Protocol (ICP) [18] used in Harvest, short-circuiting the query code path and object eviction code path to send messages to the mapping server, and adding a cache directory module for the mapping server. Local misses result in a query to the mapping server, which returns either a REDIRECT message with the address of a peer server that holds a copy of the object, or a MISS, which causes the requesting server to fetch the object directly from its home site on the Internet.

3.3 Methodology

We performed two sets of experiments to study several CRISP and Harvest configurations. For small configurations, containing up to five nodes, we played the first day (8/29) of the DEC traces into real caches running within a 100 Mb/s local area network (re-executing the whole trace would make these experiments prohibitively long). Trace-driven execution allowed us to compare hit ratios, generated network traffic, as well as latencies imposed by the caches. For large configurations, we performed trace-driven simulation using the full 25-day trace. In these experiments, we measured hit ratios and network traffic.

For trace-driven execution, client simulators on a separate set of machines generate requests. Assignment of clients to caching servers is round-robin. 16 clients per proxy submit requests to their caching servers at the maximum rate. Misses are serviced by a HTTP 1.0 server, which responds to HTTP requests by generating a dummy document of correct size, determined from an external *DB* [16] file of records for the objects in the trace, hashed by URL and request type. With the hollow HTTP server, the cache configurations consume trace data at sustained rates between 60 and 140 references per second (550 max.). Clearly, fetch latencies for misses are artificially low, since they do not include realistic network delays or end-server latencies. However, because the network and end-server delays are independent of the cache architecture, the miss latencies we measure can be used for comparison between different caching platforms. We are exploring an extension to the hollow HTTP server to model observed object transfer delays (also available in the traces).

We explored five collective cache configurations:

- **Shallow Harvest.** This consists of four child caches with a common parent, running on separate 266 MHz DEC AlphaStation 500s. Each server uses default Harvest parameters: a 16MB memory cache and a 100MB disk cache, for a total of 500 MB.

- **Flat Harvest.** This is identical to *Shallow-Harvest* but without the common parent; proxies probe the cache by multicasting to their siblings.

- **Central Synchronous Directory (CRISP/CSD).** CRISP/CSD is a CRISP configuration that is identical to *Shallow-Harvest*, except that the parent is replaced with a mapping server.

- **32-way CRISP/CSD.** We simulated a larger CRISP configuration with 32 nodes and one central mapping server.

- **Deep Harvest.** This simulated configuration consists of eight *Shallow-Harvest* subtrees, with the eight parents cooperating as in Flat Harvest. On a local miss, the simulation chooses randomly among those peers and/or parent that return a HIT on queries.

The *Shallow Harvest*, *Flat Harvest*, and *CRISP/CSD* experiments accurately measure query latencies and hit latencies, although our results do not reflect network costs for geographically dispersed configurations.

3.4 Experimental Results and Analysis

Table 1 presents various data resulting from replaying the 8/29 trace through *Shallow Harvest*, *Flat Harvest* and CRISP/CSD. The results show comparable hit ratios in all three configurations. This is expected since all three configurations find objects cached on any node (both Harvest configurations probe every server while CRISP/CSD queries the global directory). Table 1 shows that both Harvest and CRISP configurations achieve a hit ratio of 38-40% (ideal for the original trace interleaving is 42% for a 400MB cache, 43% for 500MB). Shallow Harvest yields slightly higher hit ratio due to extra cache space of the parent (its aggregate cache space is 500MB vs. 400MB for Flat Harvest and CRISP).

	Shallow Harvest				Flat Harvest				CRISP/CSD			
HIT	40.19%		28 msecs.		39.41%		147 msecs.		38.74%		256 msecs.	
MISS	59.81%		1554 msecs.		60.59%		460 msecs.		61.26%		330 msecs.	
Avg.			941 msecs.				337 msecs.				302 msecs.	
	requests		*time (msecs.)*		*requests*		*time (msecs.)*		*requests*		*time (msecs.)*	
breakdown:	number	of total	avg.	of total	number	of total	avg.	of total	number	of total	avg.	of total
local HIT	185687	32.21%	17	0.57%	186174	32.29%	93	8.94%	186367	32.33%	127	13.60%
NEIGHBOR_HIT	44796	7.77%	59	0.49%	41048	7.12%	393	8.33%	36932	6.41%	908	19.30%
PARENT_HIT	1103	0.19%	634	0.13%	-	-	-	-	-	-	-	-
probe TIMEOUT + PARENT_HIT	11	0.00%	2469	0.01%	-	-	-	-	-	-	-	-
MISS	310746	53.90%	1384	79.26%	321946	55.84%	277	45.95%	350006	60.71%	289	58.24%
probe TIMEOUT + MISS	34068	5.91%	3110	19.53%	27403	4.75%	2606	36.78%	3230	0.56%	4771	8.86%

Table 1: Comparison of *Shallow Harvest*, *Flat Harvest*, and CRISP/CSD performance in trace-driven execution.

Though they have similar hit ratios, latencies of the three configurations vary significantly. In *Shallow Harvest*, the parent is responsible for acting upon every request that misses in any child cache. Table 1 shows *Shallow Harvest* exhibiting a very high miss latency as a result of the bottleneck parent. In contrast, the parentless *Flat Harvest* configuration shows a marked improvement in miss latency, but it is still higher than CRISP/CSD. The reason for this worse miss latency in Harvest is due to multicast queries. When an object is not stored in the collective, the requesting cache waits for responses from all siblings and parents (bounded by a timeout) before forwarding a request to the Web server or to the next level in the hierarchy. Latencies increase to the timeout if any of these nodes fails, or if a query or response is dropped (both are UDP messages delivered on a best-effort basis). This limits the branching of the Harvest tree, as the probability of such a failure grows with the number of siblings and parents. We see evidence of this in Table 1, where 4.75-6% of all Harvest requests result in query timeouts, accounting for 20-37% of all time spent in the cache. In contrast, CRISP/CSD unicasts queries to the mapping server; dropped messages or timeouts are less likely, simply because fewer request/response exchanges are needed, and account for less than 5% of time spent.

On the other hand, *Shallow Harvest* yields much lower average hit latencies than *Flat Harvest*, which in turn has slightly lower hit latencies than CRISP/CSD. The reasons are twofold. First, the bottleneck parent in *Shallow Harvest* limits the overall request rate, reducing the load on child caches, which service a majority of hits. Second, if an object is stored on more than one target server, servers in both *Shallow* and *Flat* Harvest configurations will fetch the object from the target server that responds most quickly to a multicast query. CRISP/CSD does not have that flexibility. This could be addressed to a large extent by making each CRISP/CSD proxy piggyback load information on each exchange with the mapping server. Then, the mapping server may choose the best candidate from which to fetch the object.

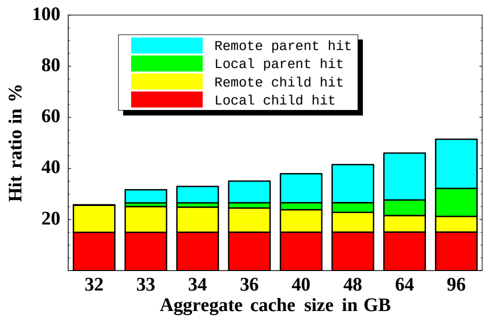

Figure 5: Hit ratio of simulated *Deep Harvest* as mid-level parent size varies (trace filtered through ideal per-user caches). Aggregate child cache size is held constant at 32GB (1GB per child).

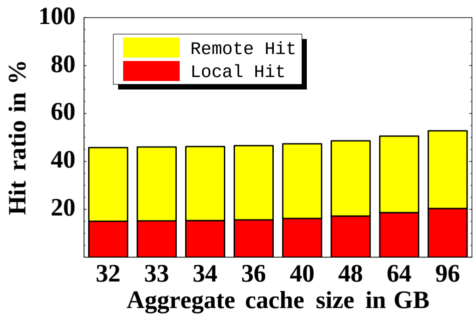

Figure 6: Hit ratio of simulated 32-way CRISP/CSD (trace filtered through ideal per-user caches).

A hierarchical cache with multicast probes such as Harvest faces basic tradeoffs between scalability and hit ratio. Shallow hierarchies (like *Shallow* or *Flat Harvest* considered above) can deliver near-ideal hit ratios because the entire cache is probed on every request. However, shallow hierarchies do not scale; the number of queries that must be carried by the network and handled by each participating cache increases linearly with the number of neighbors (siblings and parents), for a given reference trace. Another problem limiting the branching of the hierarchy is higher miss latency as discussed above.

Harvest can be made scalable by using a deeper hierarchy, but this yields lower hit ratios because "cousins" are not queried on a local miss. A deep hierarchy can guarantee to detect all possible hits only if there are common parents that are large enough to cache all documents held by their descendents. This increases the aggregate cache size requirements at least twofold. Moreover, deep hierarchies impose higher latency since fetches pass through multiple caches on their ways down the tree to the leaf proxies.

Figure 5 shows *Deep-Harvest* hit ratios for the full 25-day trace (with traces filtered through ideal per-user caches) as the cache sizes of the mid-level parents varies. The child proxies (32) each have 1GB of cache. Figure 6 shows hit ratios for the corresponding 32-way CRISP/CSD configuration with equivalent aggregate cache sizes. The figures

	Deep Harvest	32-way CRISP/CSD
Queries	98.74	9.87
Query reponses	98.74	9.87
Object requests	18.70	9.87
Object responses	18.70	9.87

Table 2: Number of messages sent (inter-cache and inter-net) by *Deep Harvest* and 32-way CRISP/CSD (in millions).

	Deep Harvest	32-way CRISP/CSD
Queries	5.92	0.59
Query responses	5.53	0.55
Object requests	0.90	0.47
Object responses	229.61	119.50

Table 3: Number of gigabytes sent (inter-cache and inter-net) by *Deep Harvest* and 32-way CRISP/CSD.

show that a hierarchical Harvest cache can deliver excellent hit ratios if the parents have enough disk to store a copy of all documents cached by their children, but that the hit ratio drops rapidly for smaller parent servers. With very small parents, the hit ratio is comparable to an eight-way (32GB aggregate) partitioned cache (as described in Section 2.2). With all caches (both children and parents) set at 1GB, resulting in the aggregate cache size of 40 GB, the 40-node *Deep Harvest* delivered a 38% hit ratio out of an achievable 53%, even though the aggregate leaf-level cache size is large enough to store the entire 25-day trace. CRISP/CSD, on the other hand, gives a 46% hit ratio for only 32 caching servers with 32GB aggregate cache size and one mapping server.

Table 2 and Table 3 detail network usage for the *Deep Harvest* and 32-way CRISP/CSD simulations with 96GB aggregate cache size. Client(browser)-proxy network usage is not included; both configurations perform identically in this regard. Because the DEC traces use an integer to represent a URL, we arbitrarily chose an average URL size of 32 bytes in determining the sizes of queries and HTTP requests/response headers. The table shows that *Deep Harvest* sends about 10 times as many queries as 32-way CRISP/CSD (recall that all multicasts are "logical" multicasts; even if IP multicast were used, the responses to the multicast query would still come from separate nodes). In addition, the number of bytes due to object fetches increases almost twofold. The reason is that all misses (and some hits, e.g. to an "uncle" cache) in *Deep Harvest* must travel through at more than one node, a parent and a child, before being returned to the requestor.

3.5 Summary

A basic tradoff between shallow Harvest configurations and CRISP is that of miss vs. hit latency. However, miss latency and generated network traffic limit the size of a shallow Harvest. For Harvest to scale, it must resort to deep configurations. Compared to deep Harvest configurations, significantly higher hit ratios are possible with lower cost, lower latency, and less network traffic using a CRISP configuration (CRISP/CSD) with a central mapping server for the same number of proxies and the same number of users. To achieve equivalent hit ratios while maintaining a scalable Harvest hierarchy would require more disk storage than the equivalent CRISP cache. Moreover, each document fetched by a leaf proxy from an external server must pass through predecessors at each level of the tree, with a synchronous multicast probe at each level. Though the use of a multicast query enables caches to detect the fastest responding server, it increases miss latency due to increased network and CPU usage. This limits the scalability and increases the cost of hierarchical caches with multicast probes.

CSD's low-cost queries and scalability to large numbers of caches come with some limitations, however. The synchronous query to a central directory requires that all caching servers have an acceptable query latency to the mapping server, ideally below the threshhold of human perception. This may limit geographic scalability. Second, a failure of the mapping server can temporarily degrade performance to that of isolated proxy caches. These issues are addressed in Section 4.

4 Alternatives for Managing the Cache Directory

We now consider three alternatives for managing the cache directory map in a CRISP collective Web cache. The purpose of these alternatives is to improve geographic scalability, reduce latency, and improve resilience to mapping server failures relative to CSD, by eliminating synchronous interaction with a central mapping server, while preserving unicast probe and high yields with low overhead.

- **Partitioned Directory (PD)**. The map is distributed among the caching servers, with the map site for each unique object determined by a hash function [8] on the URL.
- **Replicated Directory (RD)**. Weakly coherent replicas of the complete map are stored at every caching server.
- **Replicated Partial Directory (RPD)**. RPD is similar to RD, except that only the most valuable portion of the map (called the *submap*) is replicated. Updates to the submap replicas are propagated lazily, thus we also refer to this structure as *Lazy CRISP*. The version of RPD we consider, called *Replicated Partial Directory of Shared Documents (RPD-SD)*, is motivated by the observation that while almost 70% of references in the 25-day trace are to shared documents, only 29% of the documents are shared.

Many variants of these fundamental alternatives are possible. For example, the map could be replicated at a subset of the caching servers, partitioned as well as replicated, or distilled to a submap of some fixed size using LRU or other replacement algorithm. In all cases, the mechanics of the map are similar. Each proxy maintains some portion of the global map. On a local miss, a proxy queries the global map before fetching the document from the Web server. Each entry in the global map contains at minimum a URL and a copyset for the object. Stale map entries may produce false hits or misses, but are otherwise harmless. Additions and deletions to the map are idempotent.

We evaluate the potential effectiveness of these structures qualitatively and by analyzing the relevant reference characteristics of the 25-day traces.

4.1 Partitioned Directory

The primary benefit of PD over CSD is that it is scalable to arbitrarily large numbers of caching servers, since map query load is distributed among an arbitrary number of map sites. PD is also more resilient to failures of a mapping server; any failure affects only the portion of the directory stored at the map site. PD is similar to PPC (Section 2.2) in its reliance on a balanced hashing function and its need to reconfigure the cache if the hash function changes. The advantage over PPC is that UDP control messages generated by PD are much cheaper than long-haul TCP communication required in PPC.

In other respects PD is similar to CSD. The advantages for both structures are: (1) the map is guaranteed to deliver a hit if the requested object is present in the cache, and (2) exactly one unicast probe/update exchange is needed per local miss, and (3) only one copy of each global map entry is needed.

Unfortunately, PD fails to eliminate the need for synchronous query of a potentially distant map site, a central weakness of CSD. Moreover, while some map sites may be closer to the requesting proxy than the CSD mapping server, others may be more distant. Thus, compared to CSD, the worst-case latency would bound the geographical scalability of PD to smaller-diameter networks, where diameter is measured by maximum round-trip propagation latency.

4.2 Replicated Directory

Probe latency for RD is lower than either CSD or PD, since probe traffic is directed to a map replica local to the requesting proxy. However, the aggregate map storage required for RD is a factor of both the number of map replicas and the size of the map, as is the network traffic to keep the replicas coherent.

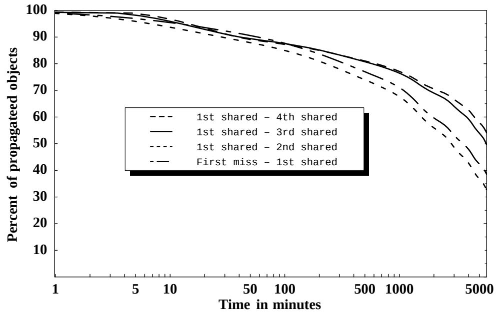

Figure 7: Inter-reference gaps between various reference classes The y-axis shows the percentage of shared objects with gaps over x minutes.

In one sense, RD is the reverse of the *Flat-Harvest* configuration studied in Section 3. Rather than multicasting queries and generating no updates to the map, RD multicasts the updates and generates no remote queries. One advantage of multicasting updates (RD) over multicasting queries (Harvest) is that the updates to the global map can be propagated asynchronously, whereas queries are by their nature synchronous. For example, proxies may batch updates to the map and propagate them with periodic anti-entropy exchanges using an efficient stream protocol [15]. While the propagation delay may produce some false misses, Figure 7 indicates that propagation delays of up to an hour can deliver over 90% of the potential *shared* hits.

4.3 Replicated Partial Directory

The goal of RPD is to reduce the storage and update costs of RD by replicating only a portion of the global map, without compromising hit ratios. We propose an instance of RPD called RPD-SD in which the shared submap contains entries for only objects that are shared.

RPD-SD can be implemented as a variant of CSD in which the role of the mapping server is to identify shared documents and propagate updates to the submap, rather than to respond to synchronous queries. As in CSD, the mapping server maintains an authoritative copy of the complete global directory. Each proxy periodically contacts the mapping server and passes a stream of URLs fetched or discarded at that proxy. The mapping server responds with a stream of updates to the submap replica at that proxy. The mapping server asynchronously updates the global directory, and identifies and adds shared URLs to the submap.

We can predict the effectiveness of RPD-SD by analyzing the references to shared documents in the 25-day trace. Figure 8 shows a detailed breakdown of all shared document references. We partition the *shared* references into several subcategories. For every shared object, the *first* reference is always a compulsory miss, and the *first-shared* reference is the access that marks it as shared. The *second-shared* and *n-shared* references are subsequent accesses by

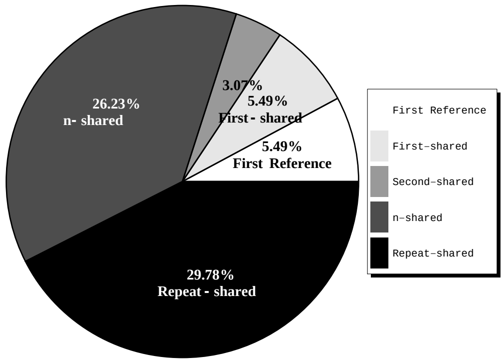

Figure 8: Breakdown of references to shared documents in the 25-day trace. Percentages are relative to all references in the trace.

new clients. A *repeat shared* reference is a further access to an object by a client who has already accessed it before.

While CSD and PD deliver all possible hits for *first-shared* references, RPD-SD yields hits only if the first and second clients to access a document happen to be bound to the same primary proxy. Our experiment with isolated proxies shows this probability declines with the number of proxies (Figure 4 in Section 2). Thus RPD-SD is likely to miss the majority of *first-shared* references, which make up 5.5% of all references in the 25-day trace.

Moreover, RPD-SD introduces two windows of vulnerability that may fail to capture potential hits from references in the *second-shared* and (less likely) the *n-shared* categories. First, such a reference may arrive before the mapping server could detect that the referenced object is shared (e.g, before the first and/or *first-shared* references are reported to the mapping server). Second, the *second-shared* and *n-shared* references may fail to yield hits if they occur before the new map entry has propagated to all submap replicas.

We can determine the size and impact of these windows from Figure 7, which shows the distribution of inter-reference gaps between the *first-shared* reference and the next three subsequent clients to access the object. The distribution predicts that RPD-SD will hit on over 95% of *second-shared* references if the maximum propagation delay of an object update (cache to map to cache submap) completes within 5 minutes. This can be satisfied, for example, by setting an interval of 2 minutes for the cache to mapping server update rate, and likewise for the mapping server to replicated submap. Similarly, RPD-SD will yield hits for 98% of the *third-shared* references and 99% of *fourth-shared* references. These three combined further decrease the overall hit ratio by 0.21 percentage points, an insignificant amount.

Clearly the highest cost is the loss of *first shared* references (a 5.5% percentage point drop). By way of a simple optimization, one can recapture these missed hits. Note that an RPD-SD cache may optionally issue a synchronous query to the central mapping server for a reference that misses in the local cache and submap replica. These queries will hit in the central mapping server if the notification of the *first miss* for that object was reported to the mapping server prior to this query. An update interval of 2 minutes as proposed above will make RPD-SD lose only 2% of the *first-shared* misses (0.10% of total), for a total drop of only 0.31 percentage points in the hit ratio). The miss latency and the best achievable hit ratio for this variant are then practically identical to CSD, but the shared submap eliminates the probe exchange for the majority of potential hits.

5 Conclusions

Caching Web proxies are an effective and low-cost tool for reducing bandwidth demands and document fetch latencies in the Internet. In a 25-day trace from Digital Equipment Corporation Web proxies, almost 70% of recorded references are to static objects shared with other users. The percentage of references to shared documents grows with larger numbers of users. This motivates the design of large-scale shared caches to exploit sharing of Internet documents among members of large communities.

Large-scale caches are most effectively built as distributed systems. Besides being more scalable, individual caching servers participating in a collective cache can be placed close to their users, improving access times and reducing the network traffic for document accesses that hit in the cache.

This paper addresses a key issue for distributed Internet caches: *how should individual caching servers share directory information to capture the largest percentage of shared document references as hits at the lowest cost?* We presented trace analyses and results from trace-driven execution of collective cache prototypes to evaluate a range of directory alternatives on the basis of cost, hit ratios, hit penalties, miss penalties, and network traffic.

While our analysis is not yet complete, we present sufficient evidence to conclude the following:

- Some form of quirying mechanism (e.g. multicast or shared directory) is needed to yield hits on the last 20% or so of references in the 25-day trace, for distributed caches with more than 16 caching servers.

- Multicasted cache probes as in Harvest can yield hits on all of these references in shallow configurations. For deeper configurations, Harvest hierarchies yield lower hit ratios unless parents are large enough to hold copies of the majority of documents cached by their children.

- By acting on the fastest positive response to a probe, shallow Harvest configurations achieve better hit latency than CRISP. On the other hand, Harvest's miss latency suffers and limits the scalability of shallow Harvest configurations.

- By maintaining a separate global map, CRISP caches yield the same hit ratio as shallow Harvest configurations, without their scalability limitation. CRISP also generates less traffic due to probes and, in the deep Harvest case, object fetches.

- Partitioned maps can improve CRISP scalability further, but result in higher query latencies. Replicated maps are scalable and produce the lowest query latencies, but at higher cost.

- By restricting map replicas to hold only the entries for shared documents, "lazy CRISP" caches can deliver the benefits of replicated maps while reducing the overheads to store and update the map replicas by about 70%, with only a 6-point drop in the achievable hit ratio for the 25-day trace. Almost all of these hits can be recovered by a simple optimization where proxies probe the global directory on a miss in their local submap replicas, resulting in only a 0.31-point drop in hit ratio.

Acknowledgments

We extend our sincere appreciation to Jeff Mogul, Tom Kroeger, Carlos Maltzahn, and Digital Equipment Corporation for providing sanitized access traces from DEC proxies. We also thank Pei Cao, Fred Douglis, and Mike Feeley for their valuable insights and feedback on this paper. Thanks also go to Peter Danzig for providing details on commercial Harvest and NetCache.

References

[1] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. Serverless network file systems. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, December 1995.

[2] Netscape Communications Corporation. Netscape Navigator 3.0 — Automatic Proxy Configurations. Available at <URL:`http://home.netscape.com/comprod/products/navigator/version_3.0/management/proxy/index.html`>, 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] Michael D. Dahlin, Clifford J. Mather, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. A quantitative analysis of cache policies for scalable network file systems. In *Proceedings of the 1994 SIGMETRICS*, pages 150–160, May 1994.

[6] S. Deering. Host extensions for IP multicasting. RFC 1112, August 1989.

[7] Fred Douglis, Antonio Haro, and Michael Rabinovich. HPP: HTML macro-preprocessing to support dynamic document caching. In *Proceedings of The USENIX Symposium on Internet Technologies and Systems (USITS)*, December 1997. To appear.

[8] Michael J. Feeley, William E. Morgan, Frederic H. Pighin, Anna R. Karlin, Henry M. Levy, and Chandramohan A. Thekkath. Implementing global memory management in a workstation cluster. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995 1995.

[9] 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.

[10] James Gwertzman and Margo Seltzer. World-wide web cache consistency. In *The 1996 Usenix Technical Conference*. USENIX, January 1996.

[11] Barron C. Housel and David B. Lindquist. WebExpress: A system for optimizing Web browsing in a wireless environment. In *Proceedings of The Second Annual International Conference on Mobile Computing and Networking (MOBICOM)*, pages 108–116, November 1996.

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

[13] Chengjie Liu and Pei Cao. Maintaining strong cache consistency in the world-wide web. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 1997.

[14] Ari Luotonen and Kevin Altis. World-wide Web proxies. In *First International Conference on the World-wide Web*, 1994. Available at <URL:`http://www.cern.ch/PapersWWW94/luotonen.ps`>.

[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 16th ACM Symposium on Operating Systems Principles.*, 1997. To appear.

[16] Margo Seltzer and Ozan Yigit. A new hashing package for unix. In *USENIX Conference Proceedings*, pages 173–84, January 1991.

[17] Vinod Valloppillil. Cache Array Routing Protocol v1.0. Internet-Draft, June 1997. Available at <URL:`http://www.io.com/ ftp/mirror/internic/internet-drafts/draft-vinod-carp-v1-00.txt`>.

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

[19] Duane Wessels et al. Squid Internet Object Cache. <URL:`http://squid.nlanr.net/`>.