Design Considerations for Distributed Caching on the Internet*

Renu Tewari, Michael Dahlin, Harrick M. Vin
Department of Computer Sciences, University of Texas at Austin
{tewari,dahlin,vin}@cs.utexas.edu

Jonathan S. Kay
Cephalapod Proliferationists, Inc.
jkay@cs.utexas.edu

Technical Report UTCS TR98-04
February 1998 (Revised October 1998)

Abstract

In this paper, we describe the design and implementation of an integrated architecture for cache systems that scale to hundreds or thousands of caches with thousands to millions of users. Rather than simply try to maximize hit rates, we take an end-to-end approach to improving response time by also considering hit times and miss times. We begin by studying several Internet caches and workloads, and we derive three core design principles for large scale distributed caches: (1) minimize the number of hops to locate and access data on both hits and misses, (2) share data among many users and scale to many caches, and (3) cache data close to clients. Our strategies for addressing these issues are built around a scalable, high-performance data-location service that tracks where objects are replicated. We describe how to construct such a service and how to use this service to provide direct access to remote data and push-based data replication. We evaluate our system through trace-driven simulation and find that these strategies together provide response time speedups of 1.27 to 2.43 compared to a traditional three-level cache hierarchy for a range of trace workloads and simulated environments.

1 Introduction

The growth of the Internet and the World Wide Web allow increasing number of users to access vast amounts of information stored at geographically distributed sites. However, long round-trip propagation delays between client and server sites, as well as hot spots of network and server load yield high latencies for information access.

Caching provides an opportunity to combat this latency by allowing users to fetch data from a nearby cache rather than from a distant server. But because users tend to access many sites, each for a short period of time, hit rates of per-user caches are low. Thus, some organizations have begun to utilize shared proxy caches [19] or hierarchical caches [8] so that each user can benefit from data fetched by others. Current shared cache architectures face a dilemma. On one hand, they wish to share data among a large number of clients to achieve good hit rates. On the other hand, as a shared cache system services more clients, the response time it provides to any one client worsens due to the increased distance between client and cache, the increased load on the cache, or the increased number of levels in the cache hierarchy. Thus, these hierarchies of data caches achieve modest hit rates [2, 14, 19, 21], can yield poor response times on a cache hit [30, 36], and can slow down cache misses.

This paper examines techniques for building systems of shared, distributed caches that scale to hundreds or thousands of caches with tens of thousands to millions of users. We believe our techniques will be of interest to system designers building large-scale, distributed cache infrastructures in a range of environments including network service providers, independent service providers, cache service providers [9, 33, 40, 45], collections of caches linked by formal service agreements [39], and large intra-nets.

Using measurements of several caches on the Internet and analysis of several traces of web traffic, we first attempt to understand the factors that limit the performance of current web caches. We find that to provide good performance to the end user, it is important not only to maximize hit rates, but also to improve hit times and miss times. Based on these measurements, we derive three basic design principles for large-scale caches: (1) minimize the number of hops to locate and access data on both hits and misses, (2) share data among many users and scale to many caches, and (3) cache data closed to clients. Although these principles may seem obvious in retrospect, current cache architectures routinely violate them at a significant performance cost. For example, hierarchical caches in the United States are often seen as a way to reduce bandwidth consumption rather than as a way to improve response time.

To address these principles, we design a scalable, high-

*This work was supported in part by an NSF Research Infrastructure Award (CDA-9624082) and grants from IBM, Intel, Lucent Bell Laboratories, Mitsubishi Electronic Research Laboratories (MERL), NASA, Novell, and Sun Microsystems. Dahlin was also supported by an NSF CAREER grant (CCR-9733842), and Vin was also supported by an NSF CAREER grant (CCR-9624757).

performance data-location service that tracks where objects are replicated. We describe how to construct such a service and how to use this service to meet our design principles via direct access to remote data and push-based data replication. Through simulation using a range of workloads and network environments, we find that direct access to remote data can achieve speedups of 1.3 to 2.3 compared to a standard hierarchy. We also find that pushing additional replicas of data provides additional speedups of 1.12 to 1.25.

We construct our data-location service using a scalable hint hierarchy in which each node tracks the nearest location of each object. Scalability and performance of the hint hierarchy comes from four sources. First, we use simple, compact data structures to allow each node's view of the hint hierarchy to track the location many objects. Second, the location system satisfies all on-line requests locally using the hint cache; the system only sends network messages through the hierarchy to propagate information in the background—off the critical path for end-user requests. Third, the hierarchy prunes updates so that updates are propagated only to the affected nodes. Fourth, we adapt Plaxton's algorithm [35] to build a scalable, fault tolerant hierarchy for distributing information.

We have implemented a prototype of our system by augmenting the widely-deployed Squid proxy cache [45].[1] It implements hint caches, push-on-write, and self-configuring dynamic hierarchies.

The rest of the paper is organized as follows. Section 2 evaluates the performance of traditional cache hierarchies and examines the characteristics of several large workloads and then derives a set of basic design principles for large-scale, distributed caches. Section 3 provides an overview of our design, and Section 4 discusses implementation details and evaluates system performance. Section 5 surveys related work, and Section 6 summarizes our conclusions and outlines areas for future research.

2 Evaluating traditional cache hierarchies

In this section, we evaluate the performance of traditional cache hierarchies using measurements of several caches on the Internet and trace-driven simulations, with the goal of understanding the factors that limit cache performance.

2.1 Workload characteristics

We examine how characteristics of web workloads stress different aspects of shared cache systems. We find that:

- Cache systems should *share data among many clients* to reduce compulsory misses (misses due to the first references to objects by clients) and *scale to large numbers of caches*.

[1] The simulator and prototype are available at http://www.cs.utexas.edu/users/tewari/cuttlefish.

Trace	# of Clients	Accesses (millions)	Distinct URLs (millions)	Dates	# of Days
DEC [13]	16,660	22.1	4.15	Sep96	21
Berkeley [21]	8,372	8.8	1.8	Nov96	19
Prodigy	35,354	4.2	1.2	Jan98	2

Table 1: Trace workloads. Note: for the DEC and Berkeley traces, each client has a unique ID throughout the trace; for the Prodigy trace, clients are dynamically bound to IDs when they log onto the system.

- Cache hit time constitutes a significant fraction of the total information access latency. Hence, cache architectures *should minimize the cost to access a cache.*

- Even an ideal cache will have a significant number of compulsory and communication misses (misses to objects that have changed since they were last referenced.) Thus, *cache systems should not slow down misses.*

We also find that capacity misses (misses to objects that have been replaced due to limited cache capacity) are a secondary consideration for large-scale cache architectures because it is economical to build shared caches with small numbers of capacity misses. If more aggressive techniques for using cache space are used (for example, pre-fetching and push caching), capacity may again be a significant consideration.

2.1.1 Methodology

Our simulation experiments use three multi-day traces taken at proxies serving thousands of clients. Table 1 summarizes key parameters. In analyzing the cache behavior of these traces, we use the first two days of each trace to warm our caches before gathering statistics.

To determine when objects are modified and should not be serviced from the cache, we use the last-modified-time information provided in the DEC traces. For the other traces or when the DEC trace does not contain the last-modified-time information, we infer modifications from document sizes and return values to if-modified-since requests. Both of these strategies will miss some of the modifications in these traces.

Current web cache implementations generally provide weak cache consistency via ad hoc algorithms. For example, current Squid caches discard any data older than two days. In our simulations, we assume that the system approximates strong cache consistency by invalidating all cached copies whenever data change. We do this for two reasons. First, techniques for approximating or providing strong cache consistency in this environment are improving [24, 29, 47], so we expect this assumption to be a good reflection of achievable future cache technology. Second, weak cache consistency distorts cache performance either by increasing apparent hit rates by counting "hits" to stale data or by reducing apparent hit rates by discarding perfectly good data from caches. In either case, this would add a potentially significant source of "noise" to our results.

These traces have two primary limitations that affect our results. First, although we use traces with thousands of clients, it still represents only a small fraction of the client population on the web. Several studies [6, 14, 21] suggest that will improve as more clients are included in a cache system The second limitation of these traces is that they are gathered at proxies rather than at clients. Thus, all of these traces will display less locality and lower total hit rates than would be seen by clients using such a system.

2.1.2 Sources of cache misses

Figure 1 shows the breakdown of cache miss rates and byte miss rates for a global cache shared by all clients in the system as cache size is varied. The cache uses LRU replacement. Misses fall into four categories:

1. Compulsory misses. These misses correspond to the first access to an object. The two key strategies for reducing compulsory misses are increasing the number of clients sharing a cache system and prefetching. Facilitating sharing is an important factor in designing large scale caches, and we discuss sharing in detail in the next subsection. We do not address prefetching in this paper.

2. Capacity misses. These misses occur when the system references an object that it has previously discarded from the cache to make space for another object. Our original intuition had been that it would be important to coordinate the contents of different caches to minimize capacity misses. However, the data suggest that for shared caches, capacity misses are a relatively minor problem that can be adequately addressed by building cache nodes with a reasonable amount of disk space. This study, therefore, does not focus on coordinated cache replacement [11, 16].

3. Communication/consistency. These misses occur when a cache holds a stale copy of data that has been modified since it was read into the cache. Providing efficient cache consistency in large systems is a current research topic [24, 29, 47], and we do not focus on that problem here. We do note, however, that the data location abstraction we construct could also be a building block for a scalable consistency system [10].

4. Uncachable/error. Objects are marked "uncachable" or encounter errors for a number of reasons, some of which might be addressed by more sophisticated cache protocols that support better cache consistency, caching dynamically generated results [41], dynamically replicating servers [43], negative result caching [31, 8], and caching programs along with data [7, 42]. We do not address such protocol extensions here. Also, because we are interested in studying the effectiveness of caching strategies, for the remainder of this study, we do not include "Uncachable" or "Error" requests in our results.

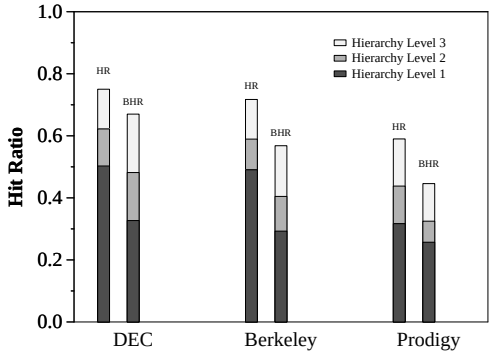

Figure 2: Overall per-read hit rate (HR) and per-byte hit rate (BHR) within infinite L1 caches shared by 256 clients, infinite L2 caches shared by 2048, and infinite L3 caches shared by all clients in the trace. As sharing increases, so does the achievable hit rate.

For all of the traces, even an ideal cache will suffer a significant number of misses. Thus, one key design principle is that in addition to having good hit rates and good hit times, cache systems should not slow down misses.

2.1.3 Sharing

Figure 2 illustrates the importance of enabling widespread sharing in large cache systems. In this experiment, we configure the system as a three-level hierarchy with 256 clients sharing a L1 proxy, eight L1 proxies (2048 clients) sharing a L2 proxy, and all L2 proxies sharing an L3 proxy. As more clients share a cache, the compulsory miss rate for that cache falls because it becomes less likely that any given access to an object is the first access to that object. For example, in the DEC traces going from a 256-client shared cache to a 16,336-client shared cache improves the byte hit rate by nearly a factor of two. Prior studies [21, 14, 6] have also reached similar conclusions. This characteristic of the workload suggests that cache systems should accommodate large numbers of clients and thereby reduce compulsory miss rates.

For caches with different degrees of sharing, Figure 3 depicts the variation in the request response times with increase in the distance between clients and the shared cache. It illustrates a dilemma faced by the designers of shared proxy caches: although it is important to share a cache among many clients, it is also important that the shared cache be close to clients. For example, a 256-client cache with an average hit time of 50 ms can outperform a 16,336-client cache that averages 300 ms per access.

Another limitation of large, monolithic caches is load. Duska et. al [14] point out that shared caches require thousands of clients and that Maltzahn and Richardson [30] found that peak processor throughput of the Squid v1.1 proxy was less than 35 req/s on a Digital AlphaStation 250 4/266. Gribble and Brewer observe that request rates are bursty at the time scale of seconds [21]. In our traces, we found peak request rates of 2850 req/s (139 MB/s) over periods of 1 second and peak rates of 294 req/s (14.4 MB/s) over periods of 10 seconds. Although processor performance will improve, we