World Wide Web Caching: Trends and Techniques

Greg Barish and Katia Obraczka, USC Information Sciences Institute

ABSTRACT

Academic and corporate communities have been dedicating considerable effort to World Wide Web caching. When correctly deployed, Web caching systems can lead to significant bandwidth savings, server load balancing, perceived network latency reduction, and higher content availability. In this article we survey the state of the art in caching designs, presenting a taxonomy of architectures and describing a variety of specific trends and techniques.

INTRODUCTION

As the Internet continues to grow in popularity and size, so do the scalability demands on its infrastructure. Exponential growth without scalability solutions will eventually result in prohibitive network load and unacceptable service response times.

It is well known that the primary reason Internet traffic has increased is the surge in popularity of the World Wide Web, specifically the high percentage of HTTP traffic. To better understand this growth rate, consider that studies conducted in the early '90s revealed that 44 percent of Internet traffic originated from FTP requests [1]. However, within the last year it was shown that HTTP activity had grown to account for somewhere between 75–80 percent of all Internet traffic [2].

THE APPEAL OF WEB CACHING

The unparalleled growth of the Internet in terms of total bytes transferred among hosts, coupled with the sudden dominance of the HTTP protocol, suggest much can be leveraged through Web caching technology. Specifically, Web caching becomes an attractive solution because it represents an effective means for reducing bandwidth demands, improving Web server availability, and reducing network latencies.

Deploying caches close to clients can reduce overall backbone traffic considerably. Cache hits eliminate the need to contact the originating server. Thus, additional network communication can be avoided.

To improve Web server availability, caching systems can also be deployed in a "reverse" fashion (e.g., the reverse proxy server approach), where caches are managed on behalf of content providers who want to improve the scalability of their site under existing or anticipated demand. In these cases, caches not only improve the availability and fault tolerance of their associated Web servers, but also act as load balancers.

Caching can improve user perceptions about network performance in two ways. First, when serving clients locally, caches hide wide area network latencies. On a local cache miss, the original content provider will serve client requests. However, in this case, server-side (reverse) caches can still play a role by reducing actual request serving time, as described above.

Second, temporary unavailability of the network can be hidden from users, thus making the network appear to be more reliable. Network outages will typically not be as severe to clients of a caching system, since local caches can be leveraged regardless of network availability. This will especially hold true for objects[1] that are temporal in nature (e.g., the isochronous delivery of multimedia data such as video and/or audio), where consistent bandwidth and response times are particularly important.

Although Web caching offers much hope for better performance and improved scalability, there remain a number of ongoing issues. Perhaps primary among these is object integrity: if an object is cached, is the user guaranteed that the cached copy is up-to-date with the version on the originating server? Other issues include using HTTP "cookies" to personalize versions of identically named URLs, content security, and the legal ethics of caching. Open issues are not the focus of this article; rather, we simply provide a taxonomy of designs and techniques that are available today.

THE NEED FOR A SURVEY

This survey is motivated by the rapid evolution of Web caching technologies. It is difficult to keep up with recent advances, since there are a number of ongoing efforts (both industrial and academic), many containing solutions based on emerging Web standards (e.g., persistent connections [3] and XML). Web caching is also a young industry, with a number of commercial vendors pushing new solutions either with direct ties to research systems or architected by individuals with notable research backgrounds. Thus, commercial solutions often contain aggressive and unique architectures.

The survey presented here serves to capture a snapshot of current design trends and techniques. Its purpose is not to compare one solution with another, but to identify common designs and put them in context.

[1] *Web pages can consist of multiple "objects": the HTML for the page itself, an embedded picture or video, a Java applet, etc.*

0163-6804/00/$10.00 © 2000 IEEE

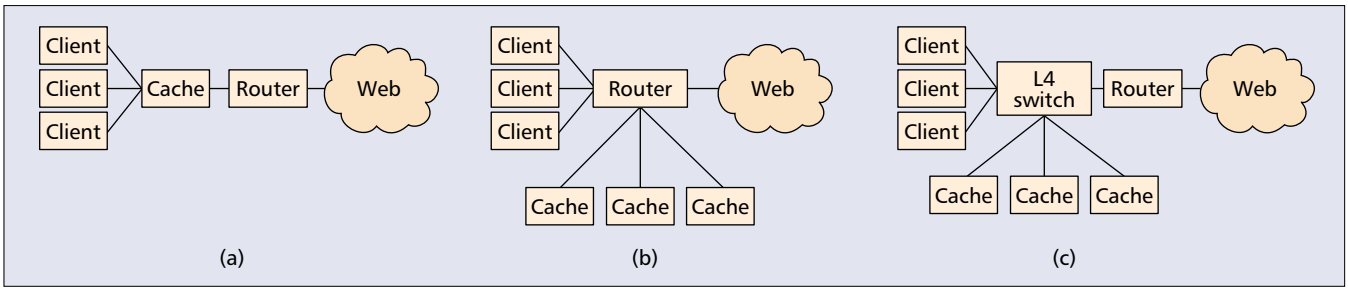

Figure 1. *a) Standalone, b) router-transparent, and c) switch-transparent proxy caching.*

OUTLINE

This article is organized as follows. The next section describes several distinct Web caching architectures. We summarize various cache deployment options; some deployments go hand in hand with the caching system architecture, whereas some architectures allow for a variety of deployment options. We also focus on common design techniques found in many existing architectures. In Appendix A, we briefly discuss the difficulty in identifying metrics of cache effectiveness and specifically address recent work on benchmarking tools. Finally, in Appendix B, we elaborate on related networking components that can augment existing cache deployment strategies to improve scalability.

CACHING ARCHITECTURES

PROXY CACHING

A proxy cache server intercepts HTTP requests from clients, and if it finds the requested object in its cache, it returns the object to the user. If the object is not found, the cache goes to the object's home server, the originating server, on behalf of the user, gets the object, possibly deposits it in its cache, and finally returns the object to the user.

Proxy caches are usually deployed at the edges of a network (i.e., at company or institutional gateway or firewall hosts) so that they can serve a large number of internal users. The use of proxy caches typically results in wide-area bandwidth savings, improved response time, and increased availability of static Web-based data and objects.

Standalone proxy configuration is shown in Fig. 1a. Notice that one disadvantage to this design is that the cache represents a single point of failure in the network. When the cache is unavailable, the network also appears unavailable to users. Also, this approach requires that all user Web browsers be manually configured to use the appropriate proxy cache. Subsequently, if the server is unavailable (due to a-long term outage or another administrative reason), all users must reconfigure their browsers in order to use a different cache.

Browser auto-configuration is a recent trend that may alleviate this problem. The Internet Engineering Task Force (IETF) recently reviewed a proposal for a Web Proxy Auto-Discovery Protocol (WPAD),[2] a means of locating nearby proxy caches. WPAD relies on resource discovery mechanisms, including domain name service (DNS) records and Dynamic Host Configuration Protocol (DHCP), to locate an automatic proxy configuration (APC) file.

A final issue related to the standalone approach has to do with scalability. As demand rises, one cache must continue to handle all requests. There is no way to dynamically add more caches when needed, as is possible with transparent proxy caching.

Reverse Proxy Caching — An interesting twist to the proxy cache approach is the notion of reverse proxy caching, in which caches are deployed near the origin of the content instead of near clients. This is an attractive solution for servers that expect a high number of requests and want to ensure a high level of quality of service. Reverse proxy caching is also a useful mechanism when supporting Web hosting farms (virtual domains mapped to a single physical site), an increasingly common service for many Internet service providers (ISPs).

Note that reverse proxy caching is totally independent of client-side proxy caching. In fact, they may coexist and collectively improve overall performance.

Transparent Caching — Transparent proxy caching eliminates one of the big drawbacks of the proxy server approach: the requirement to configure Web browsers. Transparent caches work by intercepting HTTP requests and redirecting them to Web cache servers or cache clusters. This style of caching establishes a point at which different kinds of administrative control are possible; for example, deciding how to load balance requests across multiple caches.

The strength of transparent caching is also its main weakness: it violates the end-to-end argument by not maintaining constant endpoints of the connection. This is a problem when an application requires that state be maintained throughout successive requests or during a logical request involving multiple objects.

The filtering of HTTP requests from all outbound Internet traffic adds additional latency. For example, caches deployed in conjunction with layer 4 (L4) switches rely on the fact that these switches intercept TCP traffic directed at port 80 and send all other traffic directly to the WAN router.

There are two ways to deploy transparent proxy caching: at the switch level and at the router level. Router-based transparent proxy caching (Fig. 1b) uses policy-based routing to direct requests to the appropriate cache(s). For example, requests from certain clients can be associated with a particular cache.

[2] *http://search.ietf.org/internet-drafts/draft-ietf-wrec-wpad-01.txt*

Adaptive Web caching views the caching problem as one of optimizing global data dissemination. A key problem adaptive caching targets is the "hot spot" phenomenon, where various, short-lived Internet content can, overnight, become massively popular and in high demand.

In switch-based transparent proxy caching (Fig. 1c), the switch acts as a dedicated load balancer. This approach is attractive because it reduces the overhead normally incurred by policy-based routing. Although it adds extra cost to the deployment, switches are generally less expensive than routers.

Using L4 switches for transparent caching is an example of how other network components play a role in the effectiveness of a Web caching solution. For transparent caching, these switches provide a form of local load balancing. There are other switches and solutions for local load balancing as well as solutions for global load balancing. The use of related network components for this purpose is covered further in Appendix B.

Adaptive Web Caching

Adaptive Web caching [4] views the caching problem as one of optimizing global data dissemination. A key problem adaptive caching targets is the "hot spot" phenomenon, where various short-lived Internet content can, overnight, become massively popular and in high demand.

Adaptive caching consists of multiple distributed caches which dynamically join and leave cache groups (referred to as *cache meshes*) based on content demand. Adaptivity and the self-organizing property of meshes are a response to those scenarios where demand for objects gradually evolves and those where demand spikes, or is otherwise unpredictably high or low.

Adaptive caching uses the Cache Group Management Protocol (CGMP) and Content Routing Protocol (CRP). CGMP specifies how meshes are formed, and how individual caches join and leave those meshes. In general, caches are organized into overlapping multicast groups which use voting and feedback techniques to estimate the usefulness of admitting or excluding members from that group. The ongoing negotiation of mesh formation and membership results in a virtual topology.

CRP is used to locate cached content from within the existing meshes. It can be said that CRP is a more deterministic form of hierarchical caching, which takes advantage of the overlapping nature of the meshes as a means of propagating object queries between groups, as well as propagating popular objects throughout the mesh. This technique relies on multicast communication between cache group members and makes use of URL tables to intelligently determine to which overlapping meshes requests should be forwarded.

One of the key assumptions of the adaptive caching approach is that the deployment of cache clusters across administrative boundaries is not an issue. If the virtual topologies are to be most flexible and have the highest chance of optimizing content access, administrative boundaries must be relaxed so that groups form naturally at proper points in the network.

Push Caching

As described in [5], the key idea behind push caching is to keep cached data close to the clients requesting that information. Data is dynamically mirrored as the originating server identifies where requests originate. For example, if traffic to a west coast site started to rise because of increasing requests from the east coast (typical of what happens on weekdays at 9 a.m. EST), the west coast site would respond by initiating an east coast based cache.

As with adaptive caching, one main assumption of push caching is the ability to launch caches that may cross administrative boundaries. However, push caching is targeted mostly at content providers, which will most likely control the potential sites at which the caches could be deployed. Unlike adaptive caching, it does not attempt to provide a general solution for improving content access for all types of content from all providers.

One study [6] found that well-constructed push-based algorithms can lead to speedups of between 1.27 and 2.43 as compared to traditional cache hierarchies. This study also notes the general dilemma that push caching encounters: forwarding local copies of objects incurs costs (storage, transmission), while overall performance and scalability are only seen as improved if those objects are indeed accessed.

Active Caching

The WisWeb project at the University of Wisconsin explored how caching can be applied to dynamic documents [7]. Their motivation is that the increasing amount of personalized content makes caching such information difficult and not practical with current proxy designs.

Indeed, a recent study [8] of a large ISP trace revealed that over 30 percent of client HTTP requests contained cookies, which are HTTP header elements typically indicating that a request be personalized. As Web servers become more sophisticated and customizable, and as one-to-one marketing e-commerce strategies proliferate the Internet, the level of personalization is anticipated to rise.

Active caching uses applets, located in the cache, to customize objects that could otherwise not be cached. When a request for personalized content is first issued, the originating server provides the objects and any associated cache applets. When subsequent requests are made for that same content, the cache applets perform functions locally (at the cache) which would otherwise (more expensively) be performed at the originating server. Thus, applets enable customization while retaining the benefits of caching.

Cache Deployment Options

There are three main cache deployment choices: near the content consumer (consumer-oriented), near the content provider (provider-oriented), and at strategic points in the network, based on user access patterns and network topology. Below we discuss each option's advantages and disadvantages.

Positioning caches near the client, as in proxy caching (including transparent proxy caching), has the advantage of leveraging one or more caches to a user community. If those users tend to access the same kind of content, this placement strategy improves response time by being able to serve requests locally.

Caches positioned near or maintained by the content provider, on the other hand, as in reverse proxy and push caching, improve access to a logical set of content. This type of cache deployment can be critical to delay-sensitive content such as audio or video. Positioning caches near or on behalf of the content provider allows the provider to improve the scalability and availability of content, but is obviously only useful for that specific provider. Any other content provider must do the same thing.

Of course, there are compromises between provider-oriented and consumer-oriented cache deployments. For example, resource sharing and security constraints permitting, it may be possible for multiple corporations to share the same client-side caches through a common ISP. Also, one can envision media hubs such as Broadcast.com providing content-based caching on behalf of the actual media providers. Finally, the use of both consumer-oriented and provider-oriented caching techniques is perhaps the most powerful and effective approach, since it combines the advantages of both while lowering the disadvantages of each.

The last approach, dynamic deployment of caches at network choke points, is a strategy embraced by the adaptive caching approach. Although it would seem to provide the most flexible type of cache coverage, it is still a work in progress and, to the best of the authors' knowledge, there have not been any performance studies demonstrating its benefits. The dynamic deployment technique also raises important questions about the administrative control of these caches, such as what impact network boundaries would have on cache mesh formation.

Finally, a discussion about cache deployment would not be complete without noting the capabilities of browsers to do caching on a per-user basis using the local file system. Obviously, while browser caching is useful for a given user, it does not aid in the global reduction of bandwidth or decline in average network latency for common Web objects.

Design Techniques

In addition to object hit rate, caching systems are generally evaluated according to three metrics: speed, scalability, and reliability. There are a variety of design techniques on which many commercial and academic systems rely in order to improve performance in these respects.

Hierarchical Caching

Hierarchical caching was pioneered in the Harvest Cache [9]. The basic idea is to have a series of caches hierarchically arranged in a tree-like structure and to allow those caches to leverage from each other when an object request arrives and the receiving cache does not have that object.

Usually, in hierarchical designs child caches can query parent caches and children can query each other, but parents never query children. This promotes an architecture where information gradually filters down to the leaves of the hierarchy. In a sense, the adaptive caching approach also uses cache hierarchies (in the form of cache groups) to diffuse information from dynamic hot spots to the outlying cache clusters, but these hierarchies are peer-based: the parent/child relationships are established per information object. Thus, in one case a cache group might act as a parent for a set of information object X, but also as a child (or intermediary) node for information object Y.

With hierarchical caches, it has been observed that parent nodes can become heavily swamped during child query processing. Commercial caches such as Network Appliances NetCache employ clustering to avoid this swamping effect.

Intercache Communication

Web caching systems tend to be composed of multiple distributed caches to improve system scalability and availability, or to leverage physical locality. In terms of scalability and availability, the existence of multiple distributed caches permits a system to deal with a high degree of concurrent client requests as well as survive the failure of some caches during normal operation. In terms of physical locality, assuming that bandwidth is constant, simply having caches closer in proximity to certain groups of users may be an effective way to reduce average network latencies, since there is often a correlation between the location of a user and the objects requested.

Regardless of why a logical cache system is composed of multiple distributed caches, it is often desirable to allow these caches to query each other. Distributing objects among caches also allows load balancing, and permitting subsequent intercache communication allows the overall logical system to efficiently resolve requests internally.

There are five well-known protocols [10] for intercache communication: ICP, cache digests, CRP, CARP, and WCCP. Among these, ICP has the longest history and is the most mature. It evolved from the cache communication in Harvest and was explored in more detail within Squid. With ICP, caches issue queries to other caches to determine the best location from which to retrieve the requested object. The ICP model consists of a request-reply paradigm, and is commonly implemented with unicast over UDP.

Although it was mentioned earlier that multiple distributed caches can improve scalability, they can also impede it, as ICP later revealed. One issue was raised by [11], which identified desirable limits to the depth of the cache hierarchy. For example, trees deeper than four levels provided noticeable delays. Another scalability concern was the number of ICP messages that could be generated as the number of cache peers increased. As noted in [12], there is a direct relationship between the number of peer caches and the number of ICP messages sent.

That same study of ICP also raised the issue of multicast, which is a key enabling technology of the adaptive caching design. Its CRP protocol uses multicast to query cache meshes. Overlapping nodes are used to forward unresolved queries to other meshes in the topology. To optimize the path of meshes to query, adaptive caching uses CRP to determine the likely direction of origin for the content. Although multicast was a proposed solution suggested for use with ICP when querying peer caches, the scala-

In addition to object hit rate, caching systems are generally evaluated according to three metrics: speed, scalability, and reliability. There are a variety of design techniques on which many commercial and academic systems rely in order to improve performance in these respects.

Hash-based routing is used to perform load balancing in cache clusters. It uses a hash function to map a key, such as the URL or domain name of originating server, to a cache within a cluster. Good hash functions are critical in partitioning workload evenly among caches or clusters of caches.

bility implications of this approach are still unclear.

Another technique related to cache-to-cache communication is the notion of cache digests, such as those implemented by Squid [12] and the Summary Cache [13]. Digests can be used to reduce intercache communication by summarizing the objects contained in peer caches. Thus, request forwarding can be more intelligent and more efficient. This approach is similar to the use of URL routing tables in adaptive caching as a more intelligent way to forward requests.

Finally, there are two other proprietary protocols: Cisco's WCCP and Microsoft's CARP method. When used with devices such as the Cisco Cache Engine,[3] WCCP enables HTTP traffic to be transparently redirected to the Cache Engine from network routers. WCCP has gradually been integrated into Cisco's Internet Operating System (IOS) which Cisco ships with its routers. Recently, Cisco announced it was licensing WCCP to vendors such as Network Appliance and Inktomi.

In contrast to ICP-based approaches, where caches can communicate with each other to locate requested content, Microsoft's CARP[4] is deterministic: it uses a hashing scheme to identify which proxy has the requested information. When a request comes in from a client, a proxy evaluates the hash of the requested URL with the name of the proxies it knows about, and the one with the highest value is realized to be the owner of that content.

The CARP hash-routing scheme is proposed as a means of avoiding the overhead and scalability issues associated with intercache communication. In that sense it is very similar to the CRP protocol used by the adaptive caching project, as well as the cache digest approach championed by both Summary Cache and Squid-2. All of these efforts attempt to reduce intercache communication.

Hash-Based Request Routing

Hash-based routing [14] is used to perform load balancing in cache clusters. It uses a hash function to map a key, such as the URL or domain name of the originating server, to a cache within a cluster. Good hash functions are critical for partitioning workload evenly among caches or clusters of caches. For example, NetCache uses MD5-indexed URL hash routing to access clusters of peer child caches that do not have overlapping URLs.

Since hashing can be used as the basis for cache selection during object retrieval, hash-based routing is seen as an intercache communication solution. Its use can reduce (or eliminate) the need for caches to query each other. For example, in the Microsoft CARP method, caches never query each other. Instead, requests are made to caches as a function of the hashing the URL key.

There are also scenarios in which hash-based routing is used only to point the caller in the direction of the content. This can be the case for very large caching infrastructures, such as the type described by the adaptive caching project. When locating remote content, hash-based routing can be used as a means to point the local cache in the direction of other caches (or cache meshes) which either have the object or can get it (from other caches or the originating server).

Optimized Disk I/O

Many systems, especially commercial ones, have spent substantial time tuning their disk I/O, treating the object cache as one does a high-performance database. NetCache and Novell's Internet Caching System (ICS), for example, use either application programming interfaces (APIs) provided by their own custom microkernel or low-overhead APIs provided by the host operating system.

Other disk I/O optimizations include improving the spatial locality of objects and using in-memory data structures to avoid disk I/O altogether, as in [15]. The former technique leverages knowledge about logically related content in order to determine where on the disk to place that content. In-memory data structures (i.e., hash tables) can be used to quickly determine if an object has been cached; if querying these data structures finds that the object has indeed been cached, disk operations can begin to actually locate the object (if not already in RAM). Otherwise, disk access can be avoided altogether and the object fetched from the originating server. Thus, these data structures summarize the contents of a cache so that, when possible, costly I/O operations can be avoided.

Microkernel Operating Systems

Microkernel architectures have emerged as a technique for optimizing cache performance, specifically in terms of improving resource allocation, task execution, and disk access and transfer times. The general motivation for microkernel-based approaches has been that general-purpose operating systems, such as UNIX and Windows NT, are not suitable for the specialized needs of optimized Web caching. These general-purpose operating systems handle resources such as processes and file handles in a cooperative way, whereas caching systems have specific needs related to how these resources are managed.

At least five notable vendors[5] have embraced microkernel architectures, and their systems share a number of common features. In some, caches are modeled as finite state machines. They are event-driven, allowing them to scale better than the thread-based approaches commonly used for deployment on general-purpose operating systems. Microkernels can also optimize access to disk resources by increasing the number of file handles for each process as well as creating very fast APIs for disk hardware access.

Content Prefetching

Prefetching refers to the retrieving of data from remote servers in anticipation of client requests. Prefetching can be classified as local-based or server-hint-based [16]. The former relies on data such as reference patterns to choose what to prefetch next. The latter uses data accumulated by the server, such as historical information about objects frequently requested by other clients. One disadvantage of the server-hint approach is that integration between client and server is more complicated, since there needs to be some coordination related to the communication of the hints from the server to the client. Also, as noted by [7], there are three distinct

[3] *http://www.cisco.com/univercd/cc/td/doc/pcat/206.htm*

[4] *http://www.microsoft.com/Proxy/Guide/carp-wp.asp*

[5] *CacheFlow, Cisco, InfoLibria, Network Appliance, and Novell.*

prefetching scenarios: prefetching between clients and servers, between clients and proxies, and between proxies and servers. Several studies [3, 7, 16] have attempted to quantify the benefits of prefetching in these various scenarios.

Kroeger *et al.* [16] looked at prefetching between proxies and servers. They found that without prefetching, proxy caching with an unlimited cache size resulted in a 26 percent reduction in latency. With a basic prefetching strategy in place, the reduction in latency could be improved to 41 percent, and with a more sophisticated prefetching strategy (server hints), that number could be further improved to 57 percent. As might be expected, a good prefetching algorithm and higher prefetch lead times play an important role in optimizing the profits associated with this technique.

Padmanabhan and Mogul [3] examined predictive prefetching between clients and servers, distinguishing between file access times and network latencies. In their experiment, they considered a case where a prefetching unit would be on the client side, communicating with a Web server that was integrated with a predictive engine. They also explored two models of prefetching: individual prefetching requests and pipelined requests. The found that a predictive prefetching approach could indeed shift the distribution of object access times such that Web cache hits were more frequent. However, they also correlated the gain in object access time with the trade-off in increased network bandwidth required. For example, the more easily prefetching was induced, the lower the average access time (especially in the pipelined requests case), but the greater the network bandwidth demand.

A study by Cao *et al.* [7] explored scenarios in which proxy caches pushed content out to clients connected through low-bandwidth links (i.e., modem users). Thus, this explored the issues of prefetching between clients and proxies. The study showed that the average latencies encountered by these types of clients could be reduced by over 23 percent. Their design was based on a proxy that predicted which documents a user might access in the near future and then pushed those documents out to the user's local cache during periods of idle network activity. This study acknowledged that their results were based on simulations constructed from actual modem trace logs, where the average bandwidth between clients and proxies was 21 kb/s.

Cache Consistency Methods

Cache consistency is concerned with ensuring that the cached object does not reflect stale or defunct data. For purposes of our discussion, stale or defunct data refers to locally cached objects which are either:

- No longer equivalent to the object on the originating server (a phenomenon discovered through comparison of object checksums)
- Obsolete

As identified by Dingle and Partl [17], there are four well-known cache consistency maintenance techniques to deal with detecting such instances: client polling, invalidation callbacks, time to live, and if-modified-since.

With client polling, caches timestamp each cached object and periodically compare the cached object timestamp with that of the original object (at the originating server). Out-of-date objects are discarded and newer versions fetched when necessary. This approach is very similar to the timestamp-based file system cache consistency approach used by the Sun Network File System (NFS).

Instead of having clients periodically check for inconsistency, invalidation callbacks rely on the originating server to identify stale objects. The server must then keep track of the proxies that are caching its objects and contact these proxies when objects change. On one hand, callbacks improve cache consistency as well as save network bandwidth by not requiring clients to periodically poll servers. On the other hand, there are clear scalability issues and privacy/security concerns regarding this approach, since servers need to track caches for each cached object.

Similar to the limited life of packets in transit on communication networks, cached objects can also have a time to live (TTL). When expired, these objects are dumped and new copies fetched. The TTL approach does not guarantee that an object which never changes will not be continually refetched, but it does significantly limit repeated retrieval. Adaptive TTL is a technique whereby the TTL of an object is updated within the cache when a cache hit occurs.

If-modified-since is a recent demand-driven variation on TTL-based consistency (Squid migrated to this approach). In this scenario caches only invalidate objects when they are requested and their expiration date has been reached.

Despite the concerns listed above, [18] reported that invalidation and adaptive TTL are comparable in terms of network traffic, client response times, and server CPU workload. These methods were found to be preferable over the other two approaches, given situations where consistency is important. Furthermore, it was also found that choosing to support strong consistency over weak consistency does not necessarily result in increased network bandwidth.

On one hand, callbacks improve cache consistency as well as save network bandwidth by not requiring clients to periodically poll servers. On the other hand, there are clear scalability issues and privacy/security concerns regarding this approach, since servers need to track caches for each cached object.

Conclusions

Web caching is an important technology which can improve content availability, reduce network latencies, and address increasing bandwidth demands. In this article we present several caching architectures, deployment options, and specific design techniques. We have shown that while there are different approaches to designing and deploying caches, some issues (e.g., intercache communication) remain common among them.

There remain a number of open issues in Web caching. Among the technical issues are content security, the practicality of handling dynamic and real-time data, and dealing with complex functional objects (e.g., Java programs). The next few years will likely be exciting as both researchers and vendors address these challenges.

Acknowledgments

We are grateful to Gary Tomlinson for making his ICS report quickly available to us, and to Peter Danzig for his helpful discussions.

REFERENCES

[1] R. Cáceres *et al.*, "Characteristics of Wide-Area TCP/IP Conversations," *Proc. SIGCOMM*, 1991.
[2] K. Thompson, G. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics," *Proc. 3rd Int'l. Conf. Web Caching*, 1998.
[3] V. Padmanabhan and J. C. Mogul, "Improving http Latency," *2nd World Wide Web Conf.*, 1994.
[4] S. Michel *et al.*, "Adaptive Web Caching: Towards A New Global Caching Architecture," *Comp. Networks & ISDN Syst.*, 1998.
[5] J. Gwertzman and M. Seltzer, "The Case for Geographical Push Caching," *5th Annual Wksp. Hot Op. Sys.*, 1995.
[6] R. Tewari *et al.*, "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet," Tech rep. TR98-04, Univ. of Texas at Austin, 1998.
[7] P. Cao, J. Zhang, and Kevin Beach, "Active Cache: Caching Dynamic Contents on the Web," *Proc. IFIP Int'l. Conf. Dist. Sys. Platforms and Open Dist. Processing*, 1998.
[8] R. Cáceres *et al.*, "Web Proxy Caching: The Devil Is in the Details," *Proc. Wksp. Internet Server Perf.*, 1998.
[9] A. Chankhunthod *et al.*, "A Hierarchical Internet Object Cache," *Proc. USENIX Tech. Conf.*, 1996.
[10] I. Melve, "Inter-Cache Communication Protocols," IETF WREC Working Group Draft, 1999.
[11] M. Baentsch *et al.*, "World Wide Web Caching: The Application Level View of the Internet," *IEEE Commun. Mag.*, vol. 35 June 1997.
[12] K. Claffy and D. Wessels, "ICP and the Squid Web Cache," 1997.
[13] L. Fan *et al.*, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," Comp. Sci. Dept., Univ. of Wisconsin-Madison, 1998.
[14] K. Ross, "Hash Routing for Collections of Shared Web Caches," *IEEE Network*, Nov./Dec. 1997.
[15] G. Tomlinson, D. Major, and R. Lee, "High-Capacity Internet Middleware: Internet Caching System Architectural Overview," *2nd Wksp. Internet Server Perf.*, 1999.
[16] T. M. Kroeger, D. E. Long, and J. C. Mogul, "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," *Proc. Symp. Internet Tech. and Sys.*, 1997.
[17] A. Dingle and T. Partl, "Web Cache Coherence," *5th Int'l. World Wide Web Conf.*, 1996.
[18] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World Wide Web," *Proc. 3rd Int'l. Conf. Web Caching*, 1998.
[19] J. Almeida and P. Cao, "Measuring Proxy Performance with the Wisconsin Proxy Benchmark," Tech. rep. 1373, Comp. Sci. Dept., Univ. of Wisconsin-Madison, 1998.
[20] L. Breslau *et al.*, "The Implications of Zipfs Law for Web Caching," *3rd Int'l. Conf. Web Caching*, 1998.

BIOGRAPHIES

KATIA OBRACZKA [M] (katia@isi.edu) received B.S. and M.S. degrees in electrical and computer engineering from the Federal University of Rio de Janeiro, Brazil, and M.S. and Ph.D. degrees in computer science from the University of Southern California (USC), Los Angeles. She is currently a research scientist at USC's Information Sciences Institute and a research assistant professor at USC's Computer Science Department. Her research interests include computer networks, distributed systems, and Internet information systems. She is a member of the ACM.

GREG BARISH (barish@isi.edu) is a Ph.D. student at USC. His research interests include autonomous agents, distributed systems, and databases. Prior to graduate study, he received a B.S. degree from the University of California at Los Angeles, and was an engineer at both Oracle Corporation and Healtheon Corporation.

APPENDIX A: MEASURING PERFORMANCE

Given all the existing caching solutions, how does one decide which is best? Ultimately, the effectiveness of a Web caching system will largely depend on the need and constraints of the deployer. A given architecture might inherently be more suitable (and thus perform better) for certain scenarios.

For example, corporations may have significant success with basic proxy serving, and may not view browser configuration as an issue (perhaps the company enforces use of a particular browser which supports auto-configuration). Other deployers, such as ISPs, may profit more from router- or switch-based transparent caching approaches, deployed at the edges of their networks. Finally, there are content providers who have no control over how their content is cached at the client side they merely want to deliver their content in a scalable manner. Thus, reverse proxying or push caching might be most suitable, given those constraints.

Regardless of deployment needs, there is still a demand to quantify the performance of various caching systems. Similar to the standard SPEC benchmarks used to measure the performance of multiprocessor architectures and the TPC family of benchmarks used with database systems, the cache developer and user community recently recognized the need for standard benchmarking tools to evaluate the performance of cache systems. Cache-specific performance metrics include amount of bandwidth saved, response time, hit rate, and various scalability metrics.

In this section, we briefly describe two well-known cache benchmarks: the Wisconsin Proxy Benchmark and Polygraph.

THE WISCONSIN PROXY BENCHMARK

The Wisconsin Proxy Benchmark (WPB) [19] was one of the first publicly available cache benchmarking tools. Its distinguishing features include support for studying the effects of adding disk arms and handling low-bandwidth (modem-based) clients. One interesting finding through the use of WPB was that latency advantages due to caching were essentially erased when considering the overall profit to modem-based clients.

While WPB addresses a number of important benchmarking requirements, such as initial support for temporal locality and some ability to generate load on Web server processes, it has some limitations. These include lack of support for modeling spatial locality, persistent HTTP 1.1 connections, DNS lookups, and realistic URLs. The WPB has been used to benchmark several proxy caches, including some of the research and commercial systems mentioned in [19].

POLYGRAPH

Polygraph[6] is a recently developed, publicly available cache benchmarking tool developed by NLANR. It can simulate Web clients and servers as well as generate workloads that model typical Web access patterns.

Polygraph has a client and a server component, each of which uses multiple threads to generate or process requests. This allows Polygraph to simulate concurrent requests from multiple clients to multiple servers. Polygraph can generate different types of workloads to simulate various models of content popularity. For example, Zipf-based [20] workloads (largely believed to model average Web access patterns) can be created.

More recently, Polygraph has been playing an increasing role in holding open benchmark Web Caching Bakeoffs as a way of inspiring the development community and encouraging competition toward good caching solutions. A summary of their study comparing a number of commercial and academic systems can be found online at http://bakeoff.ircache.net/N01/.

[6] *http://polygraph.ircache.net*

APPENDIX B: RELATED NETWORK COMPONENTS

The effectiveness of Web caching is, in part, affected by the related network components deployed in conjunction with a cache, cache cluster, or Web server. We now summarize a few types of local and global load balancing solutions which augment existing cache systems.

LOCAL LOAD BALANCERS

Local load balancing is concerned with improving the scalability, availability, and reliability of Web servers or Web caches. Incoming requests are intercepted and "redirected" to one member of a group of servers or caches, all of which exist in a common geographic location. By splitting the requests among members of a group, one server or cache is not forced to handle all incoming requests. Thus, such services can scale better and are more robust (no single point of failure). Local load balancing usually involves distributing requests according to some load balancing algorithm, such as round-robin or least connections.

One way to achieve local load balancing is to use L4 switches in a transparent caching environment. In that case, client outbound requests can be intercepted and redirected toward members of a cache cluster. Another type of local load balancing can be achieved by using L5 switches, which perform a more semantic style of load balancing.

The Alteon ACE-Director and CACHE-Director switches represent examples of how L4 switching can improve server and cache load balancing. Another example is the LocalDirector from Cisco, which is typically deployed to distribute incoming load on multiple Web servers or reverse proxy caches.

Recent alternatives to L4-based switching include L7 switching and L2 switches that support L4 and L7 processing. The former type provides the same functionality as an L4 switch but also includes sophisticated partitioning support and comes with IP filters that normally need to be added to L4 switches.

More recently, L5 switching has emerged as another load balancing alternative. This type of switching approaches load balancing from a more semantic level. One example is the Arrowpoint Content SmartSwitch (CSS). Like a standard L4 switch, the CSS performs load balancing for both client-based or server-based deployment strategies. However, CSS operates above the transport layer and balances based on requested content type.

For instance, on the client side CSS can be configured to redirect static HTTP requests to a local cache cluster, and bypass caching for dynamic HTTP requests. When balancing load among content servers, CSS can distinguish among different "higher-level" protocols like HTTP, SSH, and NTTP, and divert them to the appropriate server or group of servers. Furthermore, requests can be redirected based on content-based quality of service. For example, as Fig. 2 shows, administrators can redirect SSH request files to different servers. Fulfillment of those requests may be per request type (i.e., requests for video files may be slowly filled, whereas text files are handled much quicker, since the latter is easier to transmit than the former).

Distributed Web serving solutions may combine caching, L4-, and L5 switching. Take, for example, the case of an ISP that also provides Web hosting services. An L4 switch can be placed in front of each group of replicated servers, each serving different content. An L5 switch can be placed at the entrance of the ISP to divert different content to the appropriate L4-replicated server group.

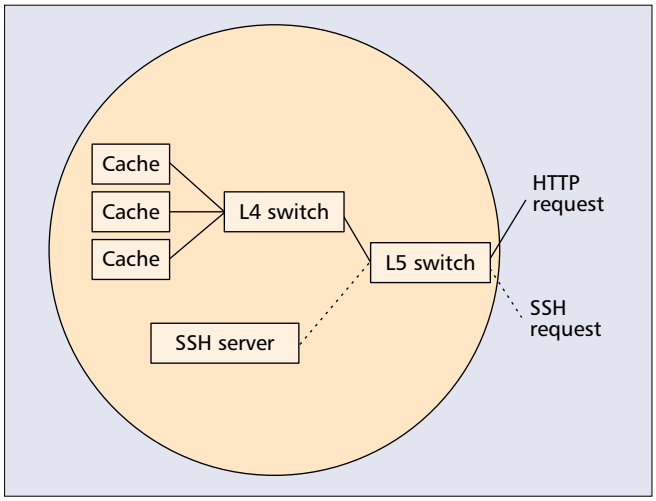

Figure 2. *Request-based routing via L5 switching.*

GLOBAL LOAD BALANCERS

Global load balancers are similar to local load balancers in that they have the goal of improving performance and scalability However, instead of distributing requests among members of a group which are geographically local to one another, global load balancers usually distribute requests to servers or caches which are near the client, to achieve lower average network latencies as well as improve scalability.

For example, if an organization has Web servers deployed throughout the world, a global load balancer can determine the appropriate host (based on physical location) for a given client request and return the IP address of that server to the client. Since browsers often cache DNS entries (local POPs also do this, etc.), repeated requests by a client to a given logical Internet address will continue to result in those requests being forwarded to the most local server.

Cisco's Distributed Director is one example of a global load balancer. When a client request is issued, it is initially directed at a primary DNS server. This request eventually reaches the local DNS server for that logical site. At this point, the local DNS server contacts the Distributed Director and determines the IP address of the geographically appropriate server for handling the request. This IP address is then sent back to the client, and will most likely be cached for further use.

Radware's Cache Server Director (CSD) is another global load balancing solution, dedicated toward improving cache fault tolerance. Consider the case of a content provider whose replicated servers are located at different Internet sites. A CSD is placed at each site as a front-end to the local set of replicated servers. Each CSD runs a local DNS server and exchanges network proximity as well as server load information with the other CSDs. A CSD maps a client request's host name to one of the ISP site's IP address. It makes its decision based on where the request originated, and its current network and server load information (about itself and the other server clusters).