Chapter 2

A Web Caching Primer

2.1 Introduction

At this time educated readers are familiar with the spectacular growth of the Internet and with the World Wide Web. More content provides incentive for more people to go online, which provides incentive to produce more content. This growth, both in usage and content, has introduced significant challenges to the scalability of the Web as an effective means to access popular content.

When the Web was new, a single entity could (and did) list and index all of the Web pages available, and searching was just an application of the Unix egrep command over an index of 110,000 documents [McB94]. Today, even though the larger search engines index billions of documents, any one engine is likely to see only a fraction of the content available to users [LG99].

Likewise, when the Web was new, page retrieval was not a significant issue — the important thing was that it worked! Now with the widespread commercialization of the Web, exceeding the 'eight-second rule' for download times may mean the loss of significant revenue [Zon99] as increasing numbers of users will go on to a different site if they are unsatisfied with the performance of the current one.

Finally, as increasing use of the Web has necessitated faster and more expensive connections to the Internet, the concern for efficient use of those connections has similarly increased. Why should an organization (either a net consumer or producer of content) pay for more bandwidth than is necessary?

This chapter provides a primer on one technology used to make the Web scalable: Web resource caching. Web caching can reduce bandwidth usage, decrease user-perceived latencies, and reduce Web server loads. Moreover, this is generally accomplished in a manner that is transparent to the end user. As a result, caching has become a significant part of the infrastructure of the Web, and thus represents a strong commercial market (more than $2B in caching sales expected in 2003 [Int99]). Caching has even spawned a new industry: *content delivery networks*, which are likewise growing at a fantastic rate (expecting over $3B in sales and services by 2006 [CK02]). In subsequent sections we will introduce caching, discuss how it differs from traditional caching applications, describe where Web caching is used, and when and why it is useful.

Those readers familiar with relatively advanced Web caching topics such as the Internet Cache Protocol (ICP) [WC97b], invalidation, or interception proxies are not likely to learn much here. Instead, this chapter is designed for a more general audience that is familiar only as a user of the Web. Likewise, this is not a how-to guide to deploy caching technologies, but rather argues at a higher level for the value of caching on the Web to both consumers and producers of content. Furthermore, while this chapter will provide references to many other published papers, it is not intended as a survey. Instead, the reader may wish to examine some surveys [Wan99, BO00, Mog00] for a more detailed treatment of current Web caching trends and techniques.

2.2 Caching

A simple example of caching is found in the use of an address book that one keeps close at hand. By keeping oft-used information nearby, one can save time and effort. While the desired phone number may be listed in a phone book, it takes a longer and may first require the retrieval of the phone book from across the room. This is the general principle used by caching systems in many contexts.

2.2.1 Caching in memory systems

A long-standing use of caching is in memory architectures [Smi82, HP87, Man82, MH99]. The central processing unit (CPU) of modern computers operate at very high speeds. Memory systems, typically are unable to work at the same speeds, and so when the CPU needs to access information in memory, it must wait for the slower memory to provide the required data. If a CPU always had to slow down for memory access, it would operate at the effective speed of a much slower CPU. To help minimize such slowdowns, the CPU designer provides one or more levels of cache — a small amount of (expensive) memory that operates at, or close to, the speed of the CPU. Thus, when the information that the CPU needs is in the cache, it doesn't have to slow down. When the cache contains the requested object, this is called a *hit*. When the requested object cannot be found in the cache, there is a cache *miss*, and so the object must be fetched directly (incurring any requisite costs for doing so).

Typically, when a cache miss occurs, the object retrieved is then placed in the cache so that the next time the object is needed, it can be accessed quickly. An assumption of *temporal locality* — the idea that a recently requested object is more likely than others to be needed in the future — is at work here. A second kind of locality, *spatial locality*, in which nearby objects are likely to be needed, is also typically assumed. For this reason (and to take advantage of other efficiencies), memory systems retrieve multiple consecutive memory addresses and place them in the cache in a single operation, rather than just one object at a time.

Regardless of how objects get placed in a cache, at some point, the cache will become full. Some algorithm will be used to remove items from the cache to make room for new objects. Various methods have been proposed and implemented, including variations of simple ideas like first-in/first-out (FIFO), least recently used (LRU), or least frequently used (LFU). When selecting a *replacement algorithm* such as one of these, the goal is to optimize performance of the cache (e.g., maximizing the likelihood of a cache hit for typical memory architectures).

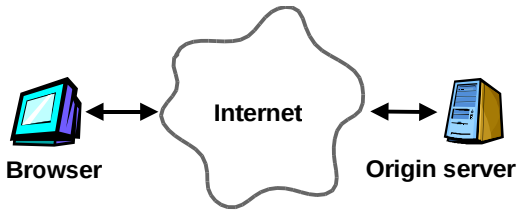

Figure 2.1: A simplistic view of the Web in which the Web consists of a set of Web servers and clients and the Internet that connects them.

2.2.2 Mechanics of a Web request

Before getting into caching on the Web, we first need to address how the World-Wide Web works. In its simplest form, the Web can be thought of as a set of Web clients (such as Web browsers, or any other software used to make a request of a Web server) and Web servers. In order to retrieve a particular Web resource, the client attempts to communicate over the Internet to the *origin Web server* (as depicted in Figure 2.1). Given a URL such as `http://www.web-caching.com/`, a client can retrieve the content (the home page) by establishing a connection with the server and making a request. However, in order to connect to the server, the client needs the numerical identifier for the host. It queries the domain name system (DNS) to translate the hostname (`www.web-caching.com`) to its Internet Protocol (IP) address (`209.182.1.122`). Once it has the IP address, the client can establish a connection to the server. Once the Web server has received and examined the client's request, it can generate and transmit the response. Each request and response include headers (such as shown in Figure 2.2) and possibly content (e.g., the HTML of the page requested, or the data for a requested image). Each step in this process takes some time (see Figure 2.3 for estimates of the time cost to establish a connection, request content, and receive the content).

The HyperText Transfer Protocol (HTTP) specifies the interaction between Web clients, servers, and intermediaries. The current version is HTTP/1.1 [FGM+99]. Requests and responses are encoded in headers that precede an optional body containing content. Figure 2.2 shows one set of request and response headers. The initial line of the request shows the method used (GET), the resource requested ("/"), and the version

Request Headers:

```
GET / HTTP/1.1
Host: www.web-caching.com
Referer: http://vancouver-webpages.com/CacheNow/
User-Agent: Mozilla/4.04 [en] (X11; I; SunOS 5.5.1 sun4u)
Accept: */*
Connection: close
```

Response Headers:

```
HTTP/1.1 200 OK
Date: Mon, 18 Dec 2000 21:25:23 GMT
Server: Apache/1.3.12 (Unix) mod_perl/1.18 PHP/4.0B2
Cache-Control: max-age=86400
Expires: Tue, 19 Dec 2000 21:25:23 GMT
Last-Modified: Mon, 18 Dec 2000 14:54:21 GMT
ETag: "838b2-4147-3a3e251d"
Accept-Ranges: bytes
Content-Length: 16711
Connection: close
Content-Type: text/html
```

Figure 2.2: Sample HTTP request and response headers. Headers identify client and server capabilities as well as describe the response content.

of HTTP supported (1.1). GET is just one of the methods defined for HTTP; another commonly used method is POST, which allows for content to be sent with the request (e.g., to carry variables from an HTML form). Likewise, the first line of the response headers shows the HTTP version supported, and a response code with standard values (e.g., 200 is OK, 404 is Not Found, etc.).

The headers of an HTTP transaction also specify aspects relevant to the cacheability of an object. The headers from the example in Figure 2.2 that are meaningful for caching include `Date`, `Last-Modified`, `ETag`, `Cache-Control`, and `Expires`. For example, in HTTP GET requests that include an `If-Modified-Since` header, Web servers use the `Last-Modified` date on the current content to return the object only if the object had changed after the date of the cached copy. An accurate clock is essential for the origin server to be able to calculate and present modification times as well as expiration times in the other tags. An `ETag` (entity tag) represents a signature for the object and allows for a stronger test than `If-Modified-Since` — if the signature of the current object at this URL matches the cached one, the objects are considered equivalent. The `Expires`

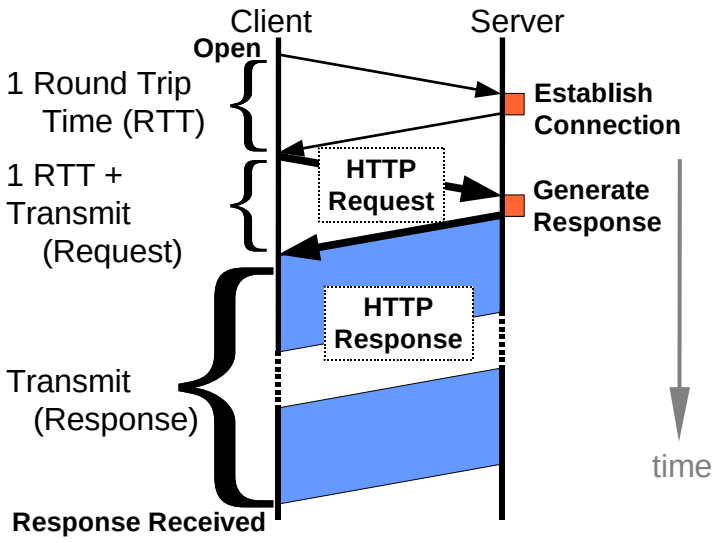

Figure 2.3: HTTP transfer timing cost for a new connection. The amount of time to retrieve a resource when a new connection is required can be approximated by two round-trip times plus the time to transmit the response (plus DNS resolution delays, if necessary).

and `Cache-Control: max-age` headers specify how long the object can be considered valid. For slowly or never changing resources, setting an explicit expiration date is the way to tell caches how long they can keep the object (without requiring the cache to contact the origin server to validate it).

2.2.3 Web caching

Now that we have discussed the basics of caching (from memory systems) and seen some of the details of an individual Web request and response, we can introduce some of the characteristics of caching on the Web and then discuss why caching is successful. Caching in the Web works in a similar manner to that of caching in memory systems — a Web cache stores Web resources that are expected to be requested again. However, it also differs in a number of significant aspects. Unlike the caching performed in many other domains, the objects stored by Web caches vary in size. As a result, cache operators and designers track both the overall *object hit rate* (percentage of requests

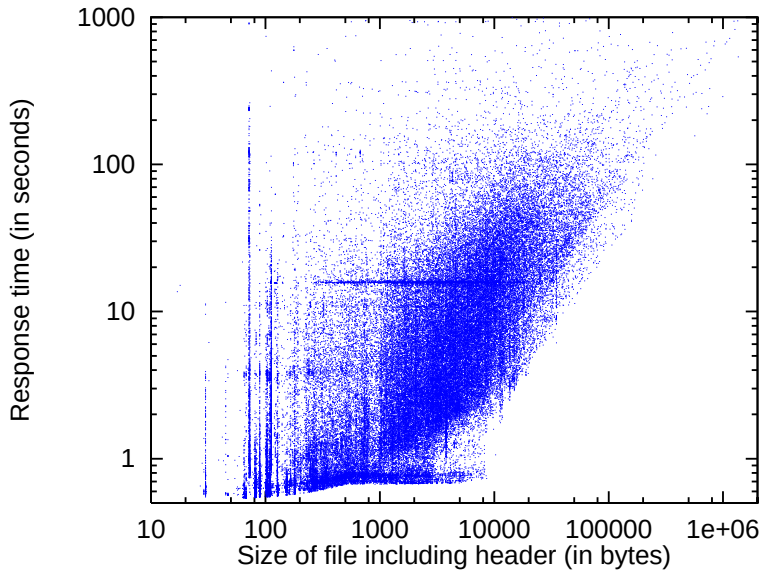

Figure 2.4: In this scatter-plot of real-world responses (the first 100,000 from the UC Berkeley Home IP HTTP Trace of dialup and wireless modem users [Gri97]), the response time varies significantly, even for resources of a single size.

served from cache) as well as the overall *byte hit rate* (percentage of bytes served from cache). Traditional replacement algorithms often assume the use of a fixed object size, so variable sizes may affect their performance. Additionally, non-uniform object size is one contributor to a second difference, that of non-uniform retrieval costs. Web resources that are larger are likely to have a higher cost (that is, higher response times) for retrieval, but even for equally-sized resources, the retrieval cost may vary as a result of distance traveled, network congestion, or server load (see Figure 2.4 for one example of real usage in which retrieval times for the same size object vary over multiple orders of magnitude). Finally, some Web resources cannot or should not be cached, e.g., because the resource is personalized to a particular client or is constantly updated (as in *dynamic pages*, discussed further in Section 2.2.8).

2.2.4 Where is Web caching used?

Caching is performed in various locations throughout the Web, including the two endpoints about which a typical user is aware. Figure 2.5 shows one possible chain of caches through which a request and response may flow. The internal nodes of this chain are

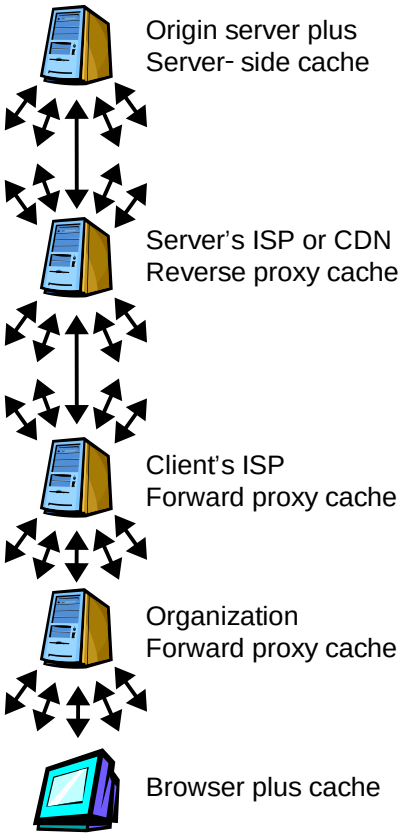

Figure 2.5: Caches in the World Wide Web. Starting in the browser, a Web request may potentially travel through multiple caching systems on its way to the origin server. At any point in the sequence a response may be served if the request matches a valid response in the cache.

termed *proxies*, since they will generate a new request on behalf of the user if the proxy cannot satisfy the request itself. A user request may be satisfied by a response captured in a browser cache, but if not, it may be passed to a department– or organization–wide proxy cache. If a valid response is not present there, the request may be received by a proxy cache operated by the client's ISP. If the ISP cache does not contain the requested response, it will likely attempt to contact the origin server. However, *reverse proxy caches* operated by the content provider's ISP or content delivery network (CDN) may instead respond to the request. If they do not have the requested information, the request may ultimately arrive at the origin server. Even at the origin server, content,

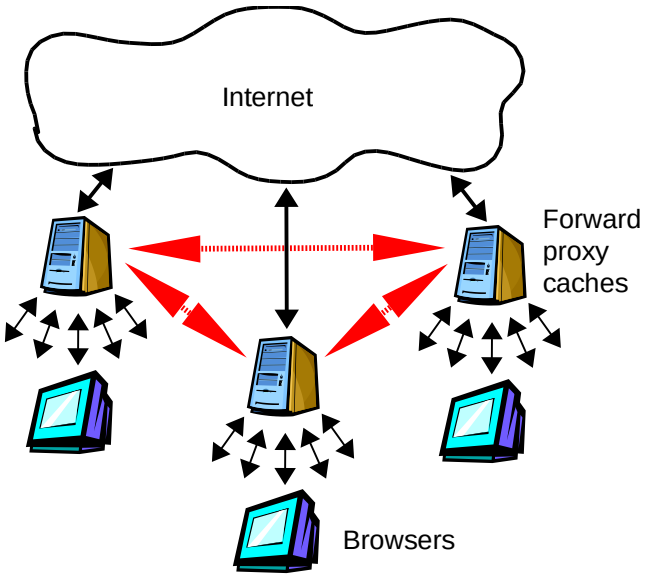

Figure 2.6: Cooperative caching in which caches communicate with peers before making a request over the Web.

or portions of content can be stored in a *server-side cache* so that the server load can be reduced (e.g., by reducing the need for redundant computations or database retrievals).

Wherever the response is generated, it will likewise flow through the reverse path back to the client. Each step in Figure 2.5 has multiple arrows, signifying relationships with multiple entities at that level. For example, the reverse proxy (sometimes called an *http accelerator*) operated by the content provider's ISP or CDN can serve as a proxy cache for the content from multiple origin servers, and can receive requests from multiple downstream clients (including forward caches operated by others, as shown in the diagram).

In general, a cache need not talk only to the clients below it and the server above it. In fact, in order to scale to large numbers of clients it may be necessary to consider using multiple caches. A hierarchical caching structure [CDN+96] is common, and is partially illustrated in Figure 2.5. In this form, each cache serves many clients, which may be users or other caches. When the request is not served by a local cache, the request is passed on to a higher level in the hierarchy, until reaching a root cache, which, having no parent, would request the object from the origin server.

An alternative approach is to use a cooperative caching architecture (see Figure 2.6), in which caches communicate with each other as peers (e.g., using an inter-cache protocol such as ICP [WC97b]). In this form, on a miss, a cache would ask a predetermined set of peers whether they already have the missing object. If so, the request would be routed to the first responding cache. Otherwise, the cache attempts to retrieve the object directly from the origin server. This approach can prevent storage of multiple copies and reduce origin server retrievals, but pays a penalty in the form of increased inter-cache communication. Instead of asking peers directly about each request, another variation periodically gives peers a summary of the contents of the cache [FCAB98, RW98], which can significantly reduce communication overhead.

Variations and combinations of these approaches have been proposed; current thinking [WVS+99] limits cooperative caching to smaller client populations, but both cooperative caching and combinations are in use in real-world sites, particularly where network access is very expensive.

Technically, a client may or may not know about intermediate proxy caches. If the client is configured directly to use a proxy cache, it will send to the proxy all requests not satisfied by a built-in cache. Otherwise, the client has to look up the IP address of the origin host. If the content provider is using a CDN, the DNS servers may be customized to return the IP address of the server (or proxy cache) closest to the client (where closest likely means not only network distance, but may also take into consideration additional information to avoid overloaded servers or networks). In this way, a reverse proxy server can operate as if it were the origin server, answering immediately any cached requests, and forwarding the rest.

Even when the client has the IP address of the server that it should contact, it might not end up doing so. Along the network path between the client and the server, there may be a network switch or router that will 'transparently' direct all Web requests to a proxy cache. This approach may be utilized at any of the proxy cache locations shown in Figure 2.5. Under this scenario, the client believes it has contacted the origin server, but instead has reached an *interception proxy* which will serve the content either from cache, or by first fetching it from the origin server.

2.2.5 Utility of Web caching

Web caching has three primary benefits. The first is that it can reduce network bandwidth usage, which can save money for both clients (consumers of content) and servers (content creators) through more efficient usage of network resources. The second is that it can reduce user-perceived delays. This is important as the longer it takes to retrieve content, the lower the value as perceived by the user [RBP98, BBK00]. Finally, caching can also reduce loads on the origin servers. While this effect may not seem significant to end users, it can allow significant savings in hardware and support costs for content providers as well as a shorter response time for non-cached resources.

The combination of these features makes caching technology attractive to all Web participants, including end-users, network managers, and content creators. There are, of course, some potential drawbacks, which we will explore shortly. First, we need to discuss how caching achieves the benefits we have described.

2.2.6 How caching saves time and bandwidth

When a request is satisfied by a cache (whether it is a browser cache, or a proxy cache run by your organization or ISP, or an edge proxy operated by a CDN), the content no longer has to travel across the Internet from the origin Web server to the cache, saving bandwidth for the cache owner as well as the origin server.

TCP, the network protocol used by HTTP, has a fairly high overhead for connection establishment and sends data slowly at first. This, combined with the knowledge that most requests on the Web are for relatively small resources, means that reducing the number of connections that are necessary and holding them open (making them persistent) so that future requests can use them, has a significant positive effect on client performance [PM94, NGBS+97]. In particular, for clients of forward proxies, time can be saved; the client can retain a persistent connection to the proxy, instead of taking the time to establish new connections with each origin server that a user might visit during a session. Use of persistent connections is particularly beneficial to clients suffering from high-latency network service such as via dial-up modems [CDF+98, FCD+99].

Furthermore, busy proxies may be able to use persistent connections to send requests for multiple clients to the same server, reducing connection establishment times to servers as well. Therefore, a proxy cache supporting persistent connections can cache connections on both client and server sides (avoiding the initial round trip time for a new HTTP connection shown in Figure 2.3).

Caching on the Web works because of various forms of temporal and spatial locality — people request popular objects from popular servers. In one study spanning more than a month, out of all the objects requested by individual users, close to 60% of those objects on average were requested more than once by the same user [TG97]. So, individual users request distinct Web objects repeatedly. Likewise, there is plenty of content that is of value to more than one user. In fact, of the hits provided in another caching study, up to 85% were the result of multiple users requesting the same objects [DMF97].

In summary, Web caching works because of popularity — the more popular a resource is, the more likely it is to be requested in the future — either by the same client, or by another. However, caching is not a panacea; it can introduce problems as well.

2.2.7 Potential problems

There are a number of potential problems associated with Web caching. Most significant from the views of both the content consumer and the content provider is the possibility of the end user seeing *stale* (e.g., old or out-of-date) content, compared to what is available on the origin server (i.e., *fresh* content). HTTP does not ensure *strong consistency* and thus the potential for data to be cached too long is real. The likelihood of this occurrence is a trade-off that the content provider can explicitly manage (as we discuss in the next section).

Caching tends to improve the response times only for cached responses that are subsequently requested (i.e., hits). Misses that are processed by a cache generally have decreased speed, as each system through which the transaction occurs will increase the latency experienced (by a small amount). Thus, a cache only benefits requests for content already stored in it. Caching is also limited by a high rate of change for

popular Web resources, and importantly, that many resources will be requested only once [DFKM97]. Finally, some responses cannot or should not be cached, as described below.

2.2.8 Content cacheability

Not every resource on the Web is cacheable. Of those that are, some will be cacheable for long periods by any cache, and others may have restrictions such as caching for short periods or to certain kinds of caches (e.g., non-proxy caches). This flexibility maximizes the opportunity for caching of individual resources. The cacheability of a Web site affects both its user-perceived performance, and the scalability of a particular hosting solution. Instead of taking seconds or minutes to load, a cached object can appear almost instantaneously. Regardless of how much the hosting costs, a cache-friendly design will allow more pages to be served before needing to upgrade to a more expensive solution. Fortunately, the cacheability of a resource is determined by the content provider. In particular, it is the Web server software that will set and send the HTTP headers that determine cacheability,[1] and the mechanisms for doing so vary for each Web server. To maximize the cacheability of a Web site, typically all static content (buttons, graphics, audio and video files, and pages that rarely change) are given an expiration date far in the future so that they can be cached for weeks or months at a time.

By setting an expiration date far into the future, the content provider is trading off the potential of caching stale data with reduced bandwidth usage and improved response time for the user. A shorter expiration date reduces the chance of the user seeing stale content, but increases the number of times that caches will need to validate the resource. Currently, most caches use a client polling approach that favors re-validation for objects that have changed recently [GS96]. Since this can generate significant overhead (especially for popular content), a better approach might be for

[1]Note that HTML META tags are not valid ways to specify caching properties and are ignored by most proxy caching products since they do not examine the contents of an object (i.e., they do not see the HTML source of a Web page).

the origin server to invalidate the cached objects [LC98], but only recently has this approach been considered for non-proprietary networks. Work is underway to develop the Web Cache Invalidation Protocol (WCIP) [LCD01], in which Web caches subscribe to invalidation channel(s) corresponding to content in which they are interested. This protocol is intended to allow the caching and distribution of large numbers of frequently changing Web objects with freshness guarantees.

Dynamically generated objects are typically considered uncacheable, although they are not necessarily so. Examples of dynamically generated objects include fast-changing content like stock quotes, personalized pages, real-time photos, results of queries to databases (e.g., search engines), page access counters, and e-commerce shopping carts. Rarely would it be desirable for any of these to be cacheable at an intermediate proxy, although some may be cacheable in the client browser cache (such as personalized resources that don't change often).

However, dynamically generated objects constitute an increasing fraction of the Web. One proposed approach to allow for the caching of dynamic content is to cache programs that generate or modify the content, such as an applet [CZB99]. Another is to focus on the server, and to enable caching of portions of documents so that server-side operations are optimized (e.g., server-side products from companies like SpiderCache [Spi02], Persistence [Per02] and XCache [XCa02]). Since most dynamic pages are mostly static, it might make sense to just send the differences between pages or between versions of a page [HL96, MDFK97] or to break those documents into separately cacheable pieces and reassemble them at the client (e.g., [DHR97]). The latter is also essentially what is proposed in Edge Side Includes [OA01] (e.g., Akamai's EdgeSuite service [Aka02]), except that the Web pages are assembled at edge servers rather than at the client.

2.2.9 Improving caching latencies

Caching can provide only a limited benefit (typically reaching object hit rates of 40-50% with sufficient traffic), as a cache can only provide objects that have been previously requested. If future requests can be anticipated, those objects can be obtained in advance. Once available in a local cache, those objects can be retrieved with minimal

delay, enhancing the user experience. *Prefetching* is a well-known approach to decrease access times in the memory hierarchy of modern computer architectures (e.g., [Smi82, Mow94, KCK+96, VL97, KW98a, LM99, JG99, SSS99]), and has been proposed by many as a mechanism for the same in the World Wide Web (e.g., [PM96, KLM97]).

While the idea is promising, prefetching is not straightforward to evaluate (as we point out later in this thesis and then propose an alternative approach in Chapter 10) and has not been widely implemented in commercial systems. It can be found in some browser add-ons and workgroup proxy caches that will prefetch the links of the current page, or periodically prefetch the pages in a user's bookmarks. One significant difficulty is to accurately predict which resources will be needed next so that mistakes which result in wasted bandwidth and increased server loads are minimized. Another concern is the need to determine what content may be prefetched safely, as some Web requests have (potentially undesirable) side-effects, such as adding items to an online shopping cart (which we discuss further in Chapter 12).

An alternative to true prefetching has been suggested by Cohen and Kaplan when they proposed techniques that do everything but prefetch [CK00]. That is, they demonstrated the usefulness of 1) pre-resolving (performing DNS lookup in advance); 2) pre-connecting (opening a TCP connection in advance); and 3) pre-warming (sending a dummy HTTP request to the origin server). These techniques can be implemented in both proxies and browsers, and could significantly reduce the response times experienced by users without large increases in bandwidth usage nor the retrieval of content with side effects.

2.3 Why do research in Web Caching?

There are a number of reasons to do research in Web caching.[2] Motivations in Web caching research can be distinguished by their outlook. In the short term, there is the desire to improve upon Web caching systems as they can affect the performance of the

[2]Many of these issues were brought up in a discussion on the `ircache` mailing list (`http://www.ircache.net/`). For the original posts, see the thread on 'free bandwidth' in the ircache archives at `http://www.roads.lut.ac.uk/lists/ircache/` for September 1998.

Web today. This includes:

- **Reducing the cost of connecting to the Internet.** Traffic on the Web consumes more bandwidth than any other Internet service, and so any method that can reduce the bandwidth requirements is particularly welcome, especially in parts of the world in which telecommunication services are much more expensive than in the U.S., and in which service is often provided on a cost per bit basis. Even when telecommunication costs are reasonable, a large percentage of traffic is destined to or from the U.S. and so must use expensive trans-oceanic network links.

- **Reducing the latency of today's WWW.** One of the most common end-user desires is for more speed. Web caching can help reduce the "World Wide Wait", and so research (such as in Web prefetching) that improves upon user-perceived response times is quite welcome. Latency improvements are most noticeable, again, in areas of the world in which data must travel long distances, accumulating significant latency as a result of speed-of-light constraints, or by accumulating processing time by each system over many network hops, and likewise increasing the chance of experiencing congestion as more networks are traversed to cover such distances. High latency as a result of speed-of-light constraints is particularly taxing in satellite communications.

In the long term one might argue that research in Web caching is not necessary, as the cost of bandwidth continues to drop. However, research in Web caching will continue to reap benefits as:

- **Bandwidth will always have some cost.** The cost of bandwidth will never reach zero, even though end-user costs are currently going down as competition increases, the market grows, and economies of scale contribute. At the same time, the cost of bandwidth at the core has stayed relatively stable, requiring ISPs to implement methods such as caching to stay competitive and reduce core bandwidth usage so that edge bandwidth costs can be low. Regardless of the

cost of bandwidth, one will always want to maximize the return on investment, and caching will often help. If applied indiscriminately, prefetching can make bandwidth needs larger, but when bandwidth is purchased at a flat rate (i.e., a T1 without per-bit charges), intelligent prefetching can be applied during idle periods to maximize the utility of the fixed cost.

- **Non-uniform bandwidth and latencies.** Because of physical limitations such as environment and location (including speed-of-light constraints mentioned above), as well as financial limitations, there will always be variations in bandwidth and latencies. Caching can help to smooth these effects.

- **Network distances are increasing.** More users are sending requests through firewalls and other proxies for security and privacy, which slows Web response times. Likewise, the use of virtual private networks for telecommuters and remote offices means that Web requests must first be routed through the VPN and suffer the associated latency costs. Both of these increase the number of hops through which content must travel, directly increasing the cost of a Web retrieval. While access to some objects served by CDNs are getting better, these still represent only a relatively small fraction of a typical user's content [KV01, BV02].

- **Bandwidth demands continue to increase.** New users are still getting connected to the Internet in large numbers. Even as growth in the user base slows, demand for increased bandwidth will continue as high-bandwidth media such as audio and video increase in popularity. If the price is low enough, demand will always outstrip supply. Additionally, as the availability of bandwidth increases, user expectations of faster performance correspondingly increase.

- **Hot spots in the Web will continue.** While some user demand can be predicted (such as for the latest version of a free Web browser), and thus have the potential for intelligent load balancing by distributing content among many systems and networks, other popular Web destinations come as a surprise, sometimes as a result of current events, but also potentially just as a result of desirable content and word of mouth. These 'hot spots' resulting from flash traffic loads will

continue to affect availability and response time and can be alleviated through distributed Web caching.

- **Communication costs exceed computational costs.** Communication is likely to always be more expensive (to some extent) than computation. We can build CPUs that are much faster than main memory, and so memory caches are utilized. Likewise, caches will continue to be used as computer systems and network connectivity both get faster.

There are a number of parallels to the persistence of caching in the Web — one is that of main memory in computer systems. The cost of RAM has decreased tremendously over the past decades. Yet relatively few people would claim to have enough, and in fact, demand for additional memory to handle larger applications continues unabated. Therefore, virtual memory (caching) continues to be used strategically instead of purchasing additional physical memory.

2.4 Summary

The benefits of caching are apparent. Given the choices of caching products on the market today, how can one be selected? The process involves determining requirements like features (such as manageability, failure tolerance, and scalability), and performance (such as requests per second, bandwidth saved, and mean response times — often measured in cache benchmarks [RW00, RWW01]). Likewise, as more Web traffic is served by caches and content delivery services instead of origin servers, it is prudent for content developers to consider how to maximize the usefulness of this technology.

By reducing network latencies and bandwidth demands in a way that is functionally transparent to the end points, caching has become a significant part of the infrastructure of the Web. This chapter has provided an overview of Web caching technology, including where Web caching is utilized, and discussed how and why it is beneficial to end users and content providers. To examine this topic in more depth, take a look at books from Addison-Wesley [KR01, RS02] and O'Reilly [Wes01], or visit `www.web-caching.com`.