Engineering Highly Accessed Web Sites for Performance

Jim Challenger, Arun Iyengar, Paul Dantzig, Daniel Dias, and Nathaniel Mills

IBM Research, T. J. Watson Research Center, P. O. Box 704,
Yorktown Heights, NY 10598, USA,
{challngr,aruni,pauldant,dias,wnm3}@us.ibm.com

Abstract. This paper describes techniques for improving performance at Web sites which receive significant traffic. Poor performance can be caused by dynamic data, insufficient network bandwidth, and poor Web page design. Dynamic data overheads can often be reduced by caching dynamic pages and using fast interfaces to invoke server programs. Web server acceleration can significantly improve performance and reduce the hardware needed at a Web site. We discuss techniques for balancing load among multiple servers at a Web site. We also show how Web pages can be designed to minimize traffic to the site.

1 Introduction

Performance is a critical factor at many Web sites. Popular Web sites can receive hundreds of thousands of hits per minute. A Web site which only receives a moderate amount of traffic can also suffer from slow response times if a significant percent of the requests are for dynamic data which are expensive to create since dynamic data can consume orders of magnitude more CPU time to create than static data.

There are a number of techniques which can be used to improve performance at a Web site. Multiple processors can be used to scale the CPU capacity of the system. We will describe techniques for balancing load among multiple processors. In order to reduce the overhead for generating dynamic pages, it is often possible to cache the dynamic pages and re-use cached copies instead of generating a new copy each time. The cache must be explicitly managed so that it is kept consistent. It is also important to use an efficient interface for invoking server programs to create dynamic data.

Web pages should also be properly designed to optimize performance. Web pages can often be redesigned to provide more information close to the home page so that less navigation is required to obtain critical information. Encryption consumes significant CPU cycles and should only be used for confidential information; many Web sites use encryption for nonessential information such as all of the image files included in a Web page.

The remainder of the paper is organized as follows. Section 2 describes how multiple Web servers can be deployed to service high request rates and techniques

for routing requests to such servers. Section 3 describes how performance can be improved with Web server acceleration. Section 4 describes techniques for efficiently serving dynamic data. Finally, Section 5 describes other techniques affecting performance such as page design.

2 Multiple Web Servers

In order to handle significant traffic, Web servers must use multiple servers running on different computers. Some sites use mirroring in which different Web sites contain the same information. Clients are responsible for selecting one of the mirrored Web sites. In some cases, the mirrored sites are geographically dispersed, and clients are supposed to select the closest Web site.

There are a number of problems with mirroring. Clients must select an appropriate Web site. This puts an extra burden on clients. There could be considerable load imbalances if some sites are selected more than others. Mirroring doesn't solve the problem of routing requests from one site to another when one of the sites fail. There is also administrative work in maintaining multiple Web sites and providing consistent content across the Web sites.

Because of the problems with mirroring, it is often preferable to have a single Web site being serviced by multiple servers running on different computers. The servers might share information using a shared file system such as the Andrew File System (AFS) or Distributed File System (DFS). Information can also be shared via a shared database or replicated across independent file systems running on the servers.

2.1 Routing Requests to Multiple Web Servers

One method for distributing requests to the various servers is by using the round-robin Domain Name Server [1, 16] (RR-DNS). RR-DNS allows a single domain name to be associated with several IP addresses which could each represent different Web servers. Client requests specifying the domain name will be mapped to Web servers in a round-robin fashion.

There are several problems that arise with this method. First, caching of name-to-IP address mappings at name servers can cause load imbalances. There are typically several name servers between clients and the RR-DNS that cache the resolved name-to-IP address mapping. In order to force a mapping to different server IP addresses, the RR-DNS can specify a *time-to-live* (TTL) for a resolved name, such that requests made after the specified TTL are not resolved in the local name server, but are forwarded to the authoritative RR-DNS to be re-mapped to the IP address of a different HTTP server. Multiple name requests made during the TTL period will be mapped to the same HTTP server. If the TTL is made very small, there is a significant increase in network traffic for name resolution. Therefore, name servers often impose their own minimum TTL, and ignore small TTL's given by the RR-DNS. There is thus no way to prevent intermediate name servers from caching the resolved name-to-IP address

mapping, even by using small TTL's. Many clients, for instance those served by the same Internet service provider, may share a name server, and may therefore be pointed to a specific Web server.

A second problem is that client caching of resolved name-to-IP address mappings can cause load imbalances. Since the clients may make future requests at any time, the load on the HTTP servers cannot be controlled subsequently and will vary due to statistical variations in client access patterns. Further, clients make requests in bursts as each Web page typically involves fetching several objects including text and images, and this burst is directed to a single server node, increasing the skew. It is shown in [8] that these effects can lead to significant dynamic load imbalance, requiring that the cluster be operated at lower mean loads in order to be able to handle peak loads.

Another problem with RR-DNS is that round-robin is often too simplistic a method for providing good load balancing. It is desirable to consider other factors such as the load on individual servers. For instance, a particular Web server may become overloaded due to requests for dynamic data, which is constructed from many database accesses at a server node.

Finally, another important problem with RR-DNS is client and name server caching of resolved name-to-IP address mappings make it difficult to provide high availability if Web server nodes fail. Since clients and name servers are unaware of Web servers going down, they may continue to make requests to failed servers. Similarly, it may be desirable to bring down a specific Web server node of a cluster for maintenance purposes. Again, making IP addresses of individual Web servers visible to the client and name servers makes it more difficult to achieve this. It is possible to configure back-up servers and perform IP address take-overs when Web server node failures are detected, or when a node is to be brought down for maintenance. However, not only is this hard to manage, but if the back-up node is active, it may get twice the load after failure of a primary node.

Another method for achieving load balancing is based on routing at the TCP level (rather than standard IP routing), and is illustrated in Figure 1. A node of the cluster serves as a so-called *TCP router(s)*, forwarding client requests to the different Web server nodes in the cluster in a round-robin (or other) order. The name and IP address of the router is public, while the addresses of the other nodes in the cluster are hidden from clients. The client sends requests to the TCP router node which in turn forwards all packets belonging to a particular TCP connection to one of the server nodes. The TCP router can use different algorithms based on load to select which node to route to, or use a simple round-robin scheme. The server nodes directly send the response back to the client, bypassing the TCP router. Note that the response packets are large compared to the request packets; these bypass the router. Thus, the overhead added by the TCP router is small.

One advantage of the router scheme over the DNS-based solutions is that good load balancing can be achieved and there is no problem of client or name server caching. It is shown in [8] that the use of a TCP router results in better
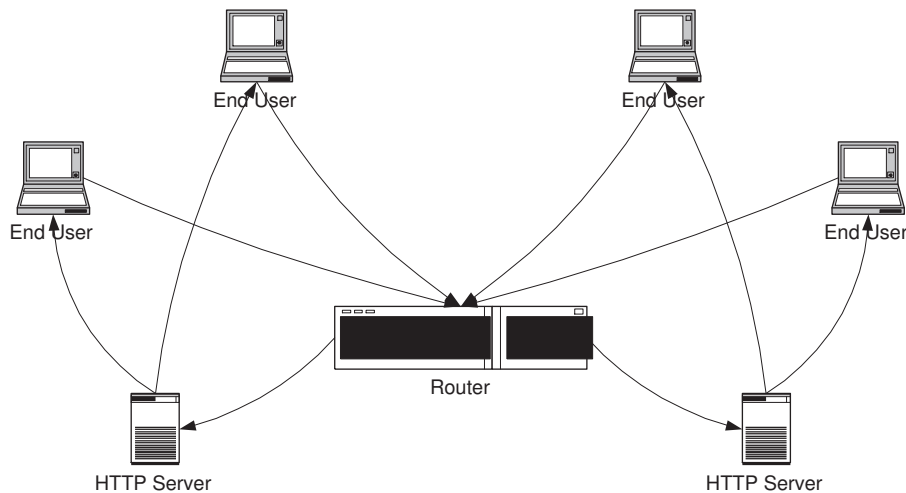
Fig. 1. A TCP router can load balance requests to multiple Web servers. Responses from the server go directly to clients, bypassing the router.

load balancing than RR-DNS. Another advantage of using TCP routers is that the router can use sophisticated load balancing algorithms which take the load on individual servers into account as opposed to simple round-robin. Finally, the TCP router provides high availability by detecting failure of Web server nodes, and routing user requests to only the available Web server nodes. In addition, for easy maintenance of the Web server cluster, the TCP router configuration can be changed to remove or add Web server nodes. Failure of the TCP router node itself is handled by configuring a back-up TCP router [8]. The back-up TCP router can operate as a Web server during normal operation; on detecting failure of the primary TCP router, the back-up TCP router would route client requests to the remaining Web server nodes, possibly excluding itself.

There are a number of commercially available TCP routers. One example is IBM's Network Dispatcher (ND) [10] which runs on stock hardware under several Operating Systems (OS), including Unix, Sun Solaris, Windows NT, and an embedded OS optimized for communication. The advantage of using an embedded OS is that router performance is improved by optimizing the TCP communications stack, and eliminating the scheduler and interrupt processing overheads of a general-purpose operating system. ND can route up to 10,000 HTTP requests per second (when running under an embedded OS on a uniprocessor machine). Other commercially available TCP routers are the Web Server Director by Radware [17] and the Resonate Central Dispatch [18]. Cisco Systems' LocalDirector [7] differs from the TCP router approach because packets returned from servers go through the LocalDirector before being returned to clients. A comparison of different load balancing approaches is contained in [2].

If a single TCP router has insufficient capacity to route requests to a site without becoming a bottleneck, the TCP router and DNS schemes can be used together. For example, a number of router nodes can be used, and the RR-DNS method can be used to map different clients to different router nodes. This hybrid scheme can tolerate the load imbalance achieved using RR-DNS because the corresponding router will route any burst of requests that were mapped by the RR-DNS to the same router to different server nodes. It achieves good scalability because (i) a long TTL can be used so that the node running the RR-DNS does not become a bottleneck, and (ii) several router nodes can be used achieving scaling beyond that of a single router.

One characteristic of many load balancing algorithms is that they spread requests from the same client across the Web server nodes; while this is often desirable, some applications need requests routed to specific servers. To support such applications, ND allows requests to be routed with an affinity towards specific servers. An example is the manner in which ND handles requests encrypted using SSL (Secure Sockets Layer). SSL generates a session key which is used for encrypting information passed between a client and server. Session keys are expensive to generate. In order to avoid regenerating a session key for every SSL request, session keys typically have a lifetime of about 100 seconds. After a client and server have established a session key, all requests within the session key lifetime between the specific client and server will use the same session key.

In a system with multiple Web servers, however, one Web server will not know about session keys generated by another Web server. If a simple load balancing scheme like round-robin is used, there is a high probability that two SSL requests from the same client within the lifetime of a session key will be sent to different servers resulting in unnecessary generation of session keys. ND avoids this problem by routing two SSL requests received from the same client within 100 seconds of each other to the same server.

2.2 Geographically Distributed Web Servers

In addition to cluster of nodes at a single site, geographically distributed Web servers supporting a single Web site provide for higher availability in the face of catastrophic failures at a location, and also can provide better response time by routing client requests to the lowest latency site for that client. There are a number of methods for providing load balancing of client requests among geographically distributed Web sites including: (i) Manual; (ii) DNS-based; (iii) Open Shortest Path First (OSPF)-based; (iv) Geographical Dispatching.

(i) Manual (naive) load balancing: This is the obvious method, where the client is given a choice of location, often based on a country selection, or geographical area. The obvious advantage is the simplicity of implementation. The drawbacks include that it places the burden on the client, and the load across the servers is not system controlled, leading to poor load balancing.

(ii) DNS-based: Geographical load balancing based on extending the basic Domain Name Server techniques are gaining increasing popularity. As discussed, in Section 2.1, DNS can be used to map incoming name resolution requests

for a Web server name to different IP addresses, which in the geographically distributed case is for the (single) IP address of each site. Each site, in turn, could be hosted on multiple computers, as discussed in Section 2.1.

Various algorithms can be used by the extended DNS to map different clients to different sites. The RR-DNS, outlined in Section 2.1, is the simplest, and maps clients to sites in a round-robin manner. One drawback is that RR-DNS does not take into account the current load on the sites, or the proximity of the client to servers. Several other DNS site selection algorithms have been devised. In one technique, the sites send load information to the DNS periodically; the DNS maps incoming requests to the sites based on last known load, such as the least loaded site, or round-robin among sites below a threshold load. The DNS could also add an estimated load for all mapping requests since the last known load, so as to attempt to prevent overloading the site(s) which was/were previously lightly loaded till the next load information is received. As explained in Section 2.1, the DNS technique is moderate at best in balancing the load, because of caching in name servers and at the client. Nevertheless, site-level load balancing using these and other similar techniques works reasonably well [2].

In another extended DNS technique, some source (i.e. requestor) IP addresses can be associated with certain geographies, such as country of origin; in this case, the DNS can map the request to the nearest geographically located sites. However, several IP address origins, such as those from multi-national companies, cannot be identified by country or geographical proximity. For such cases, another known technique is referred to as WOMbat [9]. In this technique, either the DNS and/or another set of sites ping the source IP address, and the site to serve the request is based on the response times to the pings measured. Measuring the "ping triangulation" delays is not possible for each request, because of the overheads and delays this incurs. Tables can be maintained for the best site to serve certain sets of source IP addresses.

Combinations of the above techniques can also be used. For example, the closest site if it can be identified as such can be selected unless the load at this site is above a threshold; if so, the next closest, or other site can be selected.

(iii) Open Shortest Path First (OSPF)-based: OSPF is a routing protocol supported by (most) routers on the Internet. OSPF determines the lowest cost route to a destination IP address from a specific router. In using OSPF to balance load across multiple Web sites, the Web servers are in a single subnet, and each advertises the same IP address. Thus, the routers near the requestor route the request along the lowest cost path to a selected Web server node.

This technique has been used for a number of IBM sports sites, including the Olympic Games Web site, to balance the load among multiple Web sites.

(iv) Geographical TCP routing: One of the above techniques can be used to achieve coarse load balancing across a set of geographically distributed sites used to support a single Web site. However, once a Web client is directed to a Web site in this manner, especially with the DNS or manual techniques, the Web server loses control, and the clients, or many clients behind a gateway can make requests to one site, potentially overloading it. In such a case, a TCP router at

that site can detect the overload situation, and re-route Web requests to one of the other sites. We refer to this as geographical TCP routing; this technique is supported by IBM's Network Dispatcher.

The TCP router at each site periodically get load information from the dispatchers at the other sites. When the load at one site is above a threshold, and the load at a set of other sites is below a threshold, incoming Web requests are re-directed to the other sites. This redirection can use various algorithms, such as round-robin among other qualifying sites or weighted round robin based on load or other criteria.

A combination of these techniques, using OSPF-based or DNS-based techniques for coarse-grained load balancing, and geographical TCP routing for fine-grained load balancing, works well in practice.

3   Web Server Accelerators

The performance of Web servers is limited by several factors. In satisfying a request, the requested data is often copied several times across layers of software, for example between the file system and the application and again during transmission to the operating system kernel, and often again at the device driver level. Other overheads, such as operating system scheduler and interrupt processing, can add further inefficiencies. One technique for improving the performance of Web sites is to cache data at the site so that frequently requested pages are served from a cache which has significantly less overhead than a Web server. Such caches are known as httpd accelerators [6] or Web server accelerators.

We have developed a Web server accelerator [13] which runs under an embedded operating system and can serve up to 5000 pages/second from its cache on a uniprocessor 200 MHz PowerPC. This throughput is up to an order of magnitude higher than that which would typically be achieved by a high-performance Web server running on similar hardware under a conventional operating system. The superior performance of our system results largely from the embedded operating system, by optimizing the TCP communications stack, and by largely eliminating scheduler and interrupt processing. Buffer copying is kept to a minimum. The operating system is unsuitable for implementing general-purpose software applications (like database applications or on-line transaction processing) because of its limited functionality. However, it is well-suited to specialized network applications such as Web server acceleration because of its optimized support for communications.

In order to maximize hit rates and maintain updated caches, our accelerator provides an API which allows application programs to explicitly add, delete, and update cached data. Consequently, we allow dynamic Web pages to be cached as well as static ones, since applications can explicitly invalidate any page whenever the page becomes obsolete. Caching of dynamic Web pages is important for improving the performance of many Web sites containing significant dynamic content.
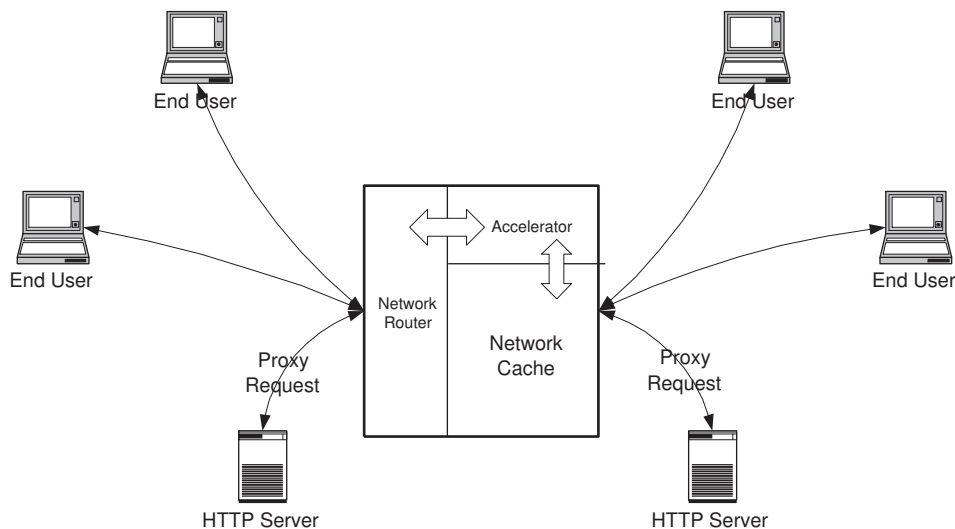
Fig. 2. Web server acceleration.

As illustrated in Figure 2, the accelerator can be placed in front of a set of Web server nodes. A TCP router runs on the same node as the accelerator (although it could also run on a separate node). If the requested page is contained in the cache, the page is returned to the client. Otherwise, the TCP router selects a Web server node to service the request, and the request is sent to the selected Web server node. Our accelerator can significantly reduce the number of Web servers needed at a Web site since a large fraction of the Web requests can be handled by the accelerator cache. A nice feature of our accelerator is that it can be used in conjunction with any server platform; special support in the operating system for serving platforms is not required.

There are a number of Web server accelerators which are implemented either in network routers, or as kernel-mode caches on the serving platform. Kernel-mode accelerators generally require special operating system support. Examples are IBM's Adaptive Fast Path Architecture (AFPA) cache, Microsoft's Scalable Web Cache (SWC) [15], and kHTTPd for Linux (http://www.fenrus.demon.nl/). Novell sells an httpd accelerator as part of its BorderManager product [12].

Web server accelerators can also be geographically distributed. In this case, they differ from proxy caches in that they cache content for specific sites, rather than caching data from all Web servers. IBM has used geographically distributed Web server accelerators for hosting a number of highly accessed sports sites.

Another example of a distributed Web server accelerator is the service offered by Akamai (www.akamai.com). Akamai has a large number of distributed Web caches, on the order of a few thousand. Web servers utilize Akamai's caching service by running a utility at the Web server which modifies the URLs of embedded objects in pages to point to Akamai's caches. The base Web page is

fetched from the server, while the embedded objects, such as images, are obtained from the Akamai caches. Akamai uses a DNS-based scheme to distribute requests for cached objects among their caches.

Web server acceleration services are also provided by other providers, such as Digital Island (http://www.digisle.net/), Mirror Image Internet (http://www.mirror-image.com/), and epicRealm (www.epicrealm.com).

4 Efficiently Serving Dynamic Data

Web servers provide two types of data: static data from files stored at a server and dynamic data which are constructed by programs that execute at the time a request is made. Dynamic pages can seriously reduce Web server performance. High-performance Web servers can typically deliver several hundred static files per second. By contrast, the rate at which dynamic pages are delivered is often orders of magnitude slower; it is not uncommon for a program to consume over a second of CPU time in order to generate a single dynamic page. For Web sites with a high proportion of dynamic pages, the performance bottleneck is often the CPU overhead associated with generating dynamic pages.

Dynamic pages are essential at Web sites which provide data that change frequently. If pages are generated dynamically by a server program, the server program can return the most recent version of the data. If, on the other hand, the data are stored in files and served from a file system, it may not be feasible to keep the files current. This is particularly true if the number of files which need to be updated frequently is large.

One technique for improving performance of dynamic data is to cache dynamic pages the first time they are created. That way, subsequent requests for the same dynamic page can access the page from a cache instead of repeatedly invoking a program to generate the same page. This was a key technique for improving performance at the Web sites for the 1996 and 1998 Olympic Games Web sites [11,3]. These Web sites served significant amounts of dynamic data, and caching was a critical component in reducing the amount of hardware need for the Web sites.

In order to keep the cached data consistent, the server should explicitly manage the cache contents instead of relying on expiration times. Algorithms for keeping cached dynamic data consistent are described in [4].

Dynamic data cannot always be cached. Some requests cause updates to occur at the server and thus must invoke a server program. If a Web site is generating pages which are personalized to individual clients, specific pages might not be accessed by multiple clients which makes caching ineffective. A technique we have developed to reduce dynamic overhead in this situation is to generate Web pages from fragments. Personalized parts of the page can be confined to specific fragments. In order to generate a personalized page, a personalized fragment is added to a template containing the rest of the page, a process which incurs significantly less overhead than regenerating the entire page from scratch [5].

Many Web sites create dynamic data from databases. Inefficient implementations will make a new connection to the database for each access. Connecting to a database can incur significant CPU overhead. It is more efficient to maintain open connections to the database via long-running processes so that new connections are not required for each access [14].

The next section discusses techniques for getting more out of servers. We examine one aspect of what distinguishes a static page from a dynamic one and try and determine if pages once thought uncachable might, in fact, be cachable. We discuss the use of database and application triggers to automatically generate HTML pages. Some of the consequences of the use of server-side processing and inefficient HTML are demonstrated. Finally we discuss pregeneration of HTML pages in more detail, with an eye toward separating server functions from page generation functions to allow better tuning of servers.

4.1   Myths About the Cachability of Dynamic Data

There has long been a misconception that dynamic pages are not cachable. It is true that some pages must be recreated on each fetch of the page. However, a great deal of data commonly thought of as *dynamic* is, in fact, highly cachable.

The difficulty with managing changing pages is controlling cache coherence. If a page changes at intervals close to, or greater than the natural lifetime of objects in caches, it is easier to manage. If the page changes more frequently, cache management becomes more difficult. But if frequently changing pages can be cached and updated properly, the savings can be considerable. For example pages with current scores of sporting events are often requested hundreds of times a second while the event is in progress, making it quite profitable to invest in the overhead of managing short lifetimes.

While an argument can be made that, given sufficient disk space and computing power, nearly every page is cachable, the cost of caching some pages can exceed the cost of generating the page on each request. Such pages can validly be considered non-cachable. For instance, attempting to cache airline reservation information is difficult because the data changes too frequently and the number of pages affected is very large. Current stock quotes are another example where the update rate may be significantly higher than the request rate.

The choice of where and when an object should be cached can also influence server design. Private data such as bank balances might be cachable in a well-controlled web server accelerator but not in public proxy caches. Server performance can be improved this way, but traffic at the proxy remains unchanged. Highly public data, however, can benefit from proxy caching because the more general appeal of such data increases the frequency of access. The nature of the data is also a consideration. Some data can tolerably be somewhat out of date, significantly increasing its cachability. Stock quotes, for instance, are often served with 20-minute delays and disclaimers regarding their use for buy/sell decisions. This relaxation of requirements could well be sufficient to permit successful caching in Web accelerators and even in proxies.

Consider a bank account where the bank's server has a Web server accelerator that is managed directly by the server itself. When the balance is first requested, the server places a copy of the returned page in its accelerator cache and notes that the user is active. This also causes a trigger to be set in the database that results in the the page being regenerated and recached if any change occurs. After a period of inactivity, or if the user explicitly logs out, the trigger and cached copies are then removed. This scheme can be optimized if a login process is required to establish a session. Part of session initialization would include prefetching commonly accessed information such as the current account summary and establishing appropriate triggers. The key idea here is that users often view shopping carts or account summaries many times before making a change that affects the underlying data. This technique can be used for many applications traditionally considered uncachable such as shopping carts and online banking.

Pages whose content can be classified as *news* are potentially quite cachable regardless of the update rate, because a great deal is known about both the update rate and the request rate. In general, news classified as current is the news most people fetch: this morning's headlines, the current sports scores, today's Doonesbury. News servers can therefore benefit greatly by forcing pages into cache as soon as they are generated, and without waiting for an explicit request. Cache misses against such pages can be very costly, because of their high request rate. When such a page is allowed to expire from cache, each subsequent request generally requires a new fetch from the server, thus causing disruptive request *spikes* at the server. Some proxies and web accelerators do not queue subsequent requests for a page if a fetch for the page is already pending to the host because of the difficulty of error recovery. Rather, if the requested object is not in cache, the request is forwarded to the host regardless of whether a similar request is already in progress. This is not a problem for infrequently accessed pages, but for popular pages, this can cause a great number of redundant host accesses until the object finally arrives in cache. This entire problem can be sidestepped for pages known to be popular by simply prefetching as soon as they are available.

Ideally, caches permit authenticated applications to directly manipulate their contents by adding, removing, and updating items. Unfortunately there is no standardization of cache protocols and APIs among vendors, so few commercial caches provide this ability yet. Older protocols such as ICP were designed to solve cache-to-cache communication problems and do not provide sufficient function for host-to-cache control.

Some caches allow authorized servers to fetch items via specially configured addresses. These addresses can be configured to bypass the cache for the fetch, but to add/replace the item in the cache when it arrives, thus providing a sort of "proxy-client" update mechanism. Unfortunately there is no comparable way to delete items, so if explicit object deletion is required rather than object replacement, a different mechanism such as time-based expiration is required.

4.2   Overuse of HTTP Services

Another significant cause of overhead on http servers is the use of server extensions. These extensions have become very popular and come in many forms: server side includes (SSI), Java Servlets, Java Server Pages (JSP), the mod_perl extensions to Apache, Microsoft's Active Server Pages (ASP), as well as more low-level interfaces such as Netscape's NSAPI and Apache's modules. The Common Gateway Interface (CGI) is still used but due to its very high overhead, is being replaced by the other, more efficient, approaches.

These extensions can provide extremely useful services, particularly when a large number of similar pages can be generated from a site. For example, Microsoft's Terraserver (http://terraserver.microsoft.com) uses ASP to implement point-and-click navigation over aerial photographs of most of the continental United States.

However, these extensions use cycles that may be better consumed serving pages. One common example is the "factoid", or "thought of the day", a short snippet of trivia or wisdom chosen at random and appended to each page of a site. These are usually included by some form of server-side parser such as SSI. The cost of doing this is very large: the server must parse every page as it serves it. However, we have found that statically choosing a random factoid only at *initial page generation time* produces significant savings at the server. Many users don't even notice the factoid, and those that do don't generally reload the page repeatedly just to see how it changes. If the site has any elements of change (all news sites, for example), the factoid tends to change fairly frequently anyway.

The per-page overhead of simple features such as factoids may seem small. However, the server has to serve those pages over and over, and at very high rates on a busy site. The accumulation of small overheads such as serve-time factoids can be significant.

Suppose a server extension is used to add a factoid to a page. Let us assume that the average number of bytes of HTML per page is 10,000 bytes. This could easily require an extra 20,000 instructions to parse the page, find the SSI directive, fetch the factoid text, insert it into the page, and finally be ready to serve the page. The IBM sports sites are starting to experience extended peak request rates approaching 1,000,000 pages per minute. These peak rates are no longer the spikes seen in the early days of the web, but are extended, flat peaks lasting as long as several hours. To insert a factoid at serve time, given these assumptions, requires additional processing power sufficient to serve (1,000,000/60) * 20,000 = 333,333,333 additional instructions per second. It is for this reason that the IBM sports servers have switched to more static factoids, generated only once at page composition time, rather than at page serve time.

Inefficient HTML is another significant performance drain. Excessive use of scripting, deeply nested tables, long hypertext references, and embedded comments and blanks can significantly increase the size of a page. One site recently analyzed has over 6,000 bytes of Javascript served on *every* page, whether the script is needed or not, as a result of the design of the common headers and foot-

ers for each page. Suppose one of the sports sites mentioned above was serving these pages. During peak periods of 1,000,000 pages per minute the site would have to serve an additional (1,000,000 / 60) * 6,000 = 100,000,000 bytes per second, requiring very substantial increase in network infrastructure as well as additional CPU horse power to feed the network.

4.3   Pre-Generation of Pages

Disk space is inexpensive these days. Several 40 GB disks are significantly less expensive than a new server. If a site is heavily visited, pre-generating all pages and serving them as flat files can result in significant savings. This is simple when the database is static or mostly static. If the database changes often, it may still be easy to "flatten" the pages. Industrial-strength databases usually implement the concept of database triggers. Through the use of database triggers, one can easily associate HTML page generation with database updates. This technique has allowed the IBM Olympic Games and Sports sites to actually *decrease* the amount of hardware, despite extremely high growth of traffic. Techniques for doing this are described in [4].

One problem with database triggers is that most real-world events cause multiple database updates. Triggering on each database update results in expensive redundant triggers that can be difficult to manage. If the transaction has to be rolled back before it is complete, some triggers would have been erroneously delivered, resulting in the generation of inconsistent pages. This problem can be solved with the use of a "commit table". The database loader is modified so that after committing the actual transaction updates, a single update to the commit table is made summarizing the transaction. The database trigger is now attached only to the commit table. This technique solves the problem of rollback, and reduces the number of trigger activations to a minimum.

Database triggers are not always available or practical to use. In these cases, simple log-following programs that parse logs produced by database loaders or content generating programs can be used in lieu of database triggers. If the publishing tools have APIs, appropriate hooks can sometimes be inserted into the tools to trigger page generation.

These examples illustrate the idea of separating page serving from page composition. Web servers generate a unique load on the system. Page composition creates a significantly different type of load. It can be difficult to tune a system that performs both tasks well, and it can be even more difficult to tune the system to do both tasks well in the same process (such as a threaded http server). By partitioning two or more systems so that some are dedicated solely to the task of generating pages, and dedicating the rest to serving, systems can be tuned for maximum throughput.

5   Other Factors Affecting Performance

Web page design can have a significant impact on performance. Web pages should be designed to convey useful information in a limited number of pages so that

clients don't navigate through too many pages to obtain the information that they are looking for. For example, Web page design for the 1998 Olympic Games Web site was considerably improved over the design for the 1996 Olympic Games Web site in order to reduce the number of intermediate pages viewed for accessing information. Figure 3 shows a home page from the 1996 Olympic Games Web site. The page doesn't contain much useful information. Clients must navigate to other pages in order to obtain the information they are looking for. By contrast, the home page for the 1998 Olympic Games Web site shown in Figure 4 contains important information on it. Fewer pages must be navigated through in order to obtain useful information. We estimate that moving from the 1996 page design to the 1998 page design may have decreased hits by more than a factor of two.

In order to reduce the number of hits to a Web site, static content can be cached remotely in proxy caches. A file cached at a remote proxy can have an expiration time associated with it indicating when the object should no longer be served. One problem with caching content in proxy caches is that standard protocols don't allow a server to pre-emptively contact proxy caches in order to notify the caches that a file has changed. Since expiration times are often difficult to guess precisely, a proxy cache may continue to serve stale data if an object changes before its assigned expiration time. Alternatively, if a server assigns expiration times conservatively so that objects usually change long after their assigned expiration times have expired, the server is likely to receive more requests for updates from proxy caches than are needed.

Encrypting Web pages via SSL can consume significant CPU cycles. In order to reduce overhead, only essential information should be encrypted. A mistake sights will often make is to encrypt not only text containing confidential information but embedded images as well. This can increase the CPU overhead significantly since a Web page might contain several embedded images, and perhaps none of them contain confidential information.

Although data encrypted via SSL is generally not cached within the network, such data can be cached in browsers if an expiration time is provided. Many sites have common navigation bars, buttons, logos, etc. which are used on several different pages. If these entities need to be encrypted, expiration times should be included in order to allow them to be cached in browsers. This will reduce the number of SSL requests to the site for the cached objects. While browser caching of nonencrypted objects also can improve performance, caching of encrypted objects can reduce server load more significantly because of the high overhead of SSL requests.

The organization of content and the design of Web pages can adversely affect performance. The HTTP protocol used to retrieve Web page content supports keep-alive sockets, a method allowing the client requesting content to reuse the same connection to the Web server for subsequent content requests once the initial request has been satisfied. Reusing sockets can avoid the overhead of negotiating SSL keys and network connections. If a page is designed to draw content from more than one server, this can cause the client to have to perform a DNS lookup for the various servers referenced, close previously opened connections

Fig. 3. Home page for the 1996 Olympic Games Web site. Clients must navigate to other Web pages in order to obtain useful information.

Fig. 4. Home page for the 1998 Olympic Games Web site containing significant useful information.

and establish new connections to other servers. If the content can be located on the same server, all of this overhead can be avoided. Keep-alive sockets can also be closed prematurely by improperly tuned servers, where the session time out is set too low causing the server to close the socket unnecessarily and forcing the client to reestablish a connection.

Because the client (e.g., browser) is able to provide content caching, page designers should ensure the same content (e.g., menu GIFs) are requested using the same URL's, thereby allowing the request to be satisfied locally from the cache. We have seen cases where the main page and search page presented the same images (e.g., menus, logos), but each was drawn from different locations on the server forcing the client to re-request the same content. Menus often comprise multiple images, one for each selection. Therefore, to populate the menu requires the client to make multiple requests to receive the individual menu images. Even if the content is able to be served from its cache, a request may be sent from the client to the server to test to see that the cached copy is still viable. By consolidating multiple menu images into one larger image and employing an image map to determine which portion of the menu bar has been selected, only one request is required to retrieve the menu (or validate the cached menu is viable), saving on network transmissions and relieving the server by reducing the number of requests it must satisfy.

We have seen many Web pages comprising 20 or more components, often resulting in downloads of more than 100,000 bytes. While these pages perform well for the developers connected by 100MB LANs to local servers, consider the typical customer attaching remotely through the Internet via a 28,800 modem and able to retrieve less than 4000 bytes per second. Often, changing the size or number of colors employed by images can have dramatic impact on improving Web page performance. By paying more attention to the ink to information ratios, one can often improve performance and simplify the presentation of information without sacrificing its beauty.

Designing Web pages to be effective is an art requiring the authors to strike a balance between information, aesthetics and performance considerations. The application or purpose of the Web page being served plays a large part in selecting the "best" design for a Web page. For example, the introductory page for a Web site provides the jumping off point to access the sites features. Is there really a need to have more information than can be viewed without scrolling? If not carefully coded, retrieval of information unable to be viewed in the visible portion of the browser can cause the page to be held pending retrieval of size and placement information before the browser can render the page. This leaves customers staring at a blank or partially formed page while they wait to use the Web page. Tags labeling page components with their dimensions allow the browser to begin page rendering as other components are retrieved. By using text in combination with graphic buttons, a page can be made navigable before all the images are retrieved. Even in cases where lots of information needs to be retrieved for a Web page, the design can make the Web page useful before all information is retrieved. Consider auction sites that are effective by presenting

the most commonly sought information (e.g., current bid, auction close, brief item description, seller) in the visible portion of the browser while the images of the items being auctioned and supporting bid histories are retrieved below the visible portion. By presenting customers with the information they seek while the remainder is retrieved, the customer switches from waiting to comprehending (effectively allowing use of think time to retrieve the supporting information). Though it still takes time to completely retrieve the page, the customers will not perceive poor performance because they can begin using the page soon after making their retrieval requests.

When engineering highly accessed Web sites for performance, it is important to understand the end-to-end dynamics of Web page retrievals. Though components are often able to be retrieved concurrently, the schedule by which they are retrieved and the ability of the receiving application to render the content plays a large part in customer-perceived performance.

References

1. T. Brisco. DNS Support for Load Balancing. Technical Report RFC 1974, Rutgers University, April 1995.
2. V. Cardellini, M. Colajanni, and P. Yu. Dynamic Load Balancing on Web-Server Systems. *IEEE Internet Computing*, pages 28–39, May/June 1999.
3. J. Challenger, P. Dantzig, and A. Iyengar. A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites. In *Proceedings of ACM/IEEE SC98*, November 1998.
4. J. Challenger, A. Iyengar, and P. Dantzig. A Scalable System for Consistently Caching Dynamic Web Data. In *Proceedings of IEEE INFOCOM'99*, March 1999.
5. J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. A Publishing System for Efficiently Creating Dynamic Web Content. In *Proceedings of IEEE INFOCOM 2000*, March 2000.
6. A. Chankhunthod et al. A Hierarchical Internet Object Cache. In *Proceedings of the 1996 USENIX Technical Conference*, pages 153–163, January 1996.
7. Cisco Systems. Inc. Cisco LocalDirector. http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/index.shtml.
8. D. Dias, W. Kish, R. Mukherjee, and R. Tewari. A Scalable and Highly Available Web Server. In *Proceedings of the 1996 IEEE Computer Conference (COMPCON)*, February 1996.
9. G. Goldszmidt and A. Stanford-Clark. Load Distribution for Scalable Web Servers: Summer Olympics 1996 - A Case Study. In *Proceedings of the 8th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, October 1997.
10. G. Hunt, G. Goldszmidt, R. King, and R. Mukherjee. Network Dispatcher: A Connection Router for Scalable Internet Services. In *Proceedings of the 7th International World Wide Web Conference*, April 1998.
11. A. Iyengar and J. Challenger. Improving Web Server Performance by Caching Dynamic Data. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.
12. R. Lee. A Quick Guide to Web Server Acceleration. http://www.novell.com /bordermanager/accel.html.

13. E. Levy, A. Iyengar, J. Song, and D. Dias. Design and Performance of a Web Server Accelerator. In *Proceedings of IEEE INFOCOM '99*, March 1999.

14. Y. H. Liu, P. Dantzig, C. E. Wu, and L. M. Ni. A Distributed Connection Manager Interface for Web Services on SP Systems. In *Proceedings of the International Conference for Parallel and Distributed Systems*, June 1996.

15. Microsoft Corporation. Installation and Performance Tuning of Microsoft Scalable Web Cache (SWC 2.0). http://www.microsoft.com/technet/iis/swc2.asp.

16. P. Mockapetris. Domain Names - Implementation and Specification. Technical Report RFC 1035, USC Information Sciences Institute, November 1987.

17. Radware Ltd. Complete IP Load balancing Solutions from RADWARE. http://www.radware.co.il/.

18. Resonate Inc. Central Dispatch - Data Sheets. http://www.resonate.com /products/central_dispatch/data_sheets.html.