Architecting Web Sites for High Performance

Arun Iyengar and Daniela Rosu

IBM Research

T.J. Watson Research Center

P.O. Box 704

Yorktown Heights, NY 10598

Corresponding author:

Arun Iyengar
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
914-784-6468
914-784-7455 (fax)
aruni@us.ibm.com

Abstract

Web site applications are some of the most challenging high-performance applications currently being developed and deployed. The challenges emerge from the specific combination of high variability in workload characteristics and of high performance demands regarding the service level, scalability, availability, and costs. In recent years, a large body of research has addressed the Web site application domain, and a host of innovative software and hardware solutions have been proposed and deployed. This paper is an overview of recent solutions concerning the architectures and the software infrastructures used in building Web site applications. The presentation emphasizes three of the main functions in a complex Web site: the processing of client requests, the control of service levels, and the interaction with remote network caches.

1 Introduction

Web site applications are some of the most challenging high-performance applications currently being developed and deployed. This class of applications spans a wide range of activities including commercial sites, like on-line retailing and auctions, financial services, like on-line banking and security trading, information sites, like news and sport events, and educational sites, like digital libraries.

The challenges that should be addressed by high-performance Web site applications emerge from the characteristics of workloads and the complexity of performance constraints that these applications have to support. In general, a Web site application provides one or more types of services developed on top of an HTTP-based infrastructure. These services may span a large range of functionalities, from delivery of static content, to execution of site-specific computations and database queries, and to streaming content. Consequently, the services offered by a site may differ with respect to request rates, response time constraints, and computation and bandwidth needs.

Numerous studies demonstrate that Web traffic is increasing at a high rate. Moreover, the throughput that Web sites need to sustain continues to increase dramatically. Figure 1 illustrates this trend by presenting the increase of (a) daily Web traffic and of (b) peak hits per minute at major sporting and event Web sites hosted by IBM from February 1998 through September 2000. This trend raises important capacity planning problems, given that having sufficient capacity to handle traffic at a given point in time might not be sufficient several months in the future.

Another challenging characteristic of Web traffic is its burstiness, given that request rates during peak intervals may be several times larger than their average rates [37]. Cost-effective solutions to this challenge multiplex several independent Web applications over the same computing and networking infrastructure in order to appropriately balance performance and cost parameters [5]. However, this raises the need for complex mechanisms for service level control.

Practically anyone who has used the Web is aware of the fact that requests can incur long response delays. On the path of a client's interaction with a Web site application, there are many components of the Web infrastructure where performance problems can occur. The Web site itself can be a major bottleneck. Particular attention is required for Web sites generating significant dynamic or encrypted content. Previous research has shown that generating dynamic content can consume significant CPU cycles [36, 35], while serving encrypted content via SSL can be more than an order of magnitude more expensive than serving unencrypted content [4].

While predictable response time is not the most critical performance demand a Web site is designed for, failing to address it may have significant business consequences. Long waits can deter users who are not willing to spend significant amounts of time accessing a Web site in order to retrieve content of interest. Moreover, unpredictable response times combined with availability gaps can make a Web site unsuitable for mission-critical applications. These problems might be due to failures of systems along the path between clients and Web sites, but often their cause lies in the Web site itself. The site's architecture and software infrastructure are likely to not scale well with high peak request rates or highly variable per-request resource needs.

Overall, the high growth rate of client population and the increased complexity of interactions and content models result in demands for high throughput, high availability, low response times, and reduced costs. To address these demands, innovative software and hardware solutions are required. Recently, a large body of research has addressed Web site-related problems. This paper is an overview of recent solutions concerning the architectures, software infrastructure services, and operating system services used in building high performance Web site applications.

The remainder of this paper is organized as follows. Section 2 presents the architecture of a high performance Web site, identifying the main components and briefly discussing implementation challenges and recent solutions. Sections 3-5 review the challenges and solutions proposed for several Web site functions which have a significant impact on performance – the processing of client requests (Section 3), the control of service levels (Section 4), and the interactions with remote network caches (Section 5). Finally, Section 6 concludes the paper highlighting several problems for future research.

2 Web Site Architecture

In this section, we introduce the main elements of a high-volume Web site architecture. Focusing on their role in processing client requests, we categorize the components of a Web site's architecture in three layers: the request distribution layer, the content delivery layer, and the content generation layer. The *request distribution layer* performs the routing of requests to the Web site's nodes or subsystems that can appropriately process them. The goal of request distribution is to ensure the targeted levels of throughput, response time, and availability. This task is particularly challenging when the site is offering several services, each with specific resource needs, request patterns, throughput goals, and business values. The request distribution layer includes components such as Domain Name System (DNS) servers, access switches and routers, and infrastructure for performance monitoring and request (re)routing.

The *content delivery layer* replies to client requests by transmitting content available in disk and memory-based stores, or acquired by interaction with content generation components. A goal of the content delivery layer is to maximize the ratio of requests that can be serviced from local stores, thus minimizing response times and server loads. This layer includes components such as caching proxies, HTTP servers, application servers, and content transformation engines.

Finally, the *content generation layer* handles content materialization triggered by client requests and database updates. This layer includes database servers, caches of query results, engines for tracking updates and triggering off-line re-computation, and infrastructure for Web site management.

The three layers of a Web site application may be mapped to a variety of physical configurations ranging from a single node to a worldwide distributed infrastructure. Figure 2 illustrates the architecture of a complex Web site which would be similar to those at sites such as IBM's sporting and event Web sites [34]. The content generation layer is at the central point of the site configuration and typically spans a relatively small geographical area. Content delivery and request routing layers may be widely distributed.

Multiple Web servers would typically be needed to handle high request rates. In Figure 2, multiple Web servers satisfy requests dispatched by the connection router according to some load balancing policy. Web sites may be mirrored in geographically distributed locations. A client can then obtain content with less latency by accessing the Web site replica that is closest to it. For some mirrored Web sites, clients must pick the closest site themselves, while for other sites, requests are routed by the network to the closest site [16]. Mirrored Web sites also provide higher availability. If one replica site is down, others are likely to be available.

In the remainder of this section, we briefly discuss the role of the main components in a Web site architecture including DNS servers, caching proxies, connection routers, HTTP servers, query engines, and dynamic content management frameworks. The presentation order follows the flow of a client's interaction with the Web site application.

DNS Servers. DNS servers provide client hosts with the IP address of one of the site's content delivery nodes. When a request is made to a Web site such as *http://www.research.ibm.com/hvws*, "www.research.ibm.com" must be translated to an IP address. DNS servers perform this translation.

Besides enabling clients to reach the targeted Web sites, DNS servers provide a means for scaling Web sites to handle high request rates. Namely, a name associated with a Web site might map to several IP addresses, each associated with a different set of server nodes. DNS servers select one of these addresses based on a site-specific request distribution policy. This policy may be as simple as Round Robin, but typically it falls into the 'Least Loaded' paradigm, attempting to minimize maximum node utilization [19, 15]. In the latter case, load estimations are based on observations local to the DNS server (e.g., number of forwarded clients), and on monitoring information provided by each of the nodes (e.g., number of serviced requests, CPU utilization). In either case, the load estimators are likely to have low accuracy, and this is due to several factors.

First, not all requests trigger identical server loads, with load variability likely to increase as the web site complexity increases. Second, DNS servers are not on the path of all of the requests that reach the site, as the results of DNS lookups can be cached. Cached name-to-IP address mappings have lifetimes, called "Time-To-Live" (TTL) attributes, which are provided by the DNS servers and which indicate when these mappings are no longer valid. If a mapping is cached, a subsequent request to the same Web site can obtain the IP address from a cache, obviating the need to contact the DNS server. Thus, the caching of name-to-IP address mappings allows requests to reach the site's nodes without being load balanced by the DNS server; this increases the risk of load imbalance and limits the effectiveness of DNS-based load balancing [22].

Overcoming these drawbacks by increasing the rate of monitoring reports may be prohibitively expensive. A typical solution is to limit the number of requests that reach the site without the control of the DNS server. This is achieved by setting very short, possibly zero, TTL's, causing most of the requests to trigger DNS lookups. However, recent research provides evidence that this approach has negative effects on client performance as it may result in significant increase of per-page response times [51]. Consequently, for a scalable Web site, DNS-based request distribution should be coupled with local solutions for load redistributions [8].

Connection Routers. Clusters dedicated to content delivery typically have a *connection routing* front end, a component of the request distribution layer which routes requests to multiple back-end servers. The connection router hides the IP addresses of the back-end servers. This contrasts with typical DNS-based routing in which clients can obtain the addresses of individual back-end servers. However, for sites with multiple connection routers, DNS lookups provide the address of one of these routers.

A connection router includes a router or a switch that may be stand-alone [46, 61, 22, 31], may work with dedicated resource management nodes [62, 54], or may be backed by request re-routing systems distributed across the back-end nodes [8, 63, 7]. The request distribution policies implemented by the connection router are based on system load parameters and rules for mapping content and service types to resources.

The performance enabled by a connection router is better than that achieved with DNS-based routing [22]. This is mainly because, in comparison to a DNS server, the connection router handles all requests addressed to the site, and thus requests do not reach back-end nodes without first going through the router. In addition, a connection router can have more accurate system load information (e.g., number of active connections, transferred payload), can monitor resource utilization with a much finer granularity, and can determine and exploit the type of the requested service and content.

Besides enabling effective load distribution, connection routers simplify system management. Namely, by hiding the identities of back-end servers, a connection router enables transparent additions or removals of servers. In addition, when a server fails, a connection router can quickly identify the event and stop directing new requests to the node.

Reverse Proxies. A Web site's *reverse proxies*, also called "Web server accelerators", are components in the content delivery layer that attempt to service client requests from a high-performance cache. When this is not possible, requests are forwarded to the Web site. A key difference between a reverse proxy and a proxy cache that serves an institution or a region is the mix of cached content. An institution or region cache, also called a forward proxy, includes content from a large set of Web sites and therefore has a large working-set size, requiring an extensive amount of storage to obtain reasonable hit rates. In contrast, a reverse proxy cache includes content from a single site. Therefore, its working set is relatively small. Less storage is thus required for a reverse proxy to achieve good hit rates.

A reverse proxy may be located both within and outside the physical confines of the Web site. Its physical configuration varies from a cache running on a high-performance connection router [41], to a stand-alone node [39, 32], to a cluster [26, 52, 53, 30]. In cluster-based reverse proxies, nodes operate independently or as cooperative caches [52, 30]. In typical implementations, reverse proxies do not have disk stores, relying only on main memory caches. Reverse proxies can include the functionality of a content-based router[32], a particular type of connection router in which the target node is selected based on the type of the requested content.

By offloading many of the requests addressed to the regular Web servers at a site, reverse proxies contribute to an increase in the site's overall throughput. Moreover, their restricted functionality is amenable to efficient implementations, which can boost throughput and reduce response times. Reverse proxies may comprise a cluster of cooperative caches [53, 30].

In order to achieve consistency and prevent stale data from being served, a reverse proxy cache should have mechanisms that allow the server to invalidate and possibly push data. The API used in [41] allows dynamic data to be cached in addition to static data. More recently, a significant body of work has addressed the problem of consistency protocols for proxy caches. Much of this work is relevant to reverse proxies. New proposals aim to reduce Web site overhead and client-observed response times by extending traditional pull-based protocols with push-based methods [35, 57, 21]. Namely, the Web site keeps track of the proxies caching its content and appropriately pushes invalidation messages when new versions are created.

HTTP Servers. HTTP servers process requests for static and pre-computed dynamic content [48] and provide the underlying framework for execution of application servers, such as IBM's WebSphere [33], in response to client requests. The main characteristics that distinguish HTTP servers are their software architectures and execution modes. The software architectures of HTTP servers vary from event-based with a single thread handling all client connections (e.g., Zeus[60], NWSA[32], kHTTPd[55]), to a combination of event-based processing and thread-based handling of disk I/O operations (TUX[44]), to pure thread-based processing with connection-to-thread mappings that last for the entire connection lifetime (e.g., Apache[3]). With respect to execution mode, HTTP servers may execute in user mode (e.g., Zeus, Apache) or in kernel mode (e.g., kHTTPd, TUX, NWSA). Both software architecture and execution mode influence the achievable performance levels; significant performance advantages result from event-based kernel-mode servers. Recent research has focused on optimizing the operating system functions on the critical path of request processing. For instance, [11] proposes scalable solutions to the UNIX methods for select and allocation of file descriptors, which are critical for user-mode, event-based servers. [9] proposes a network subsystem architecture that allows for incoming network traffic to be processed at the priority of the receiving process, thus achieving stability under overload and integrating the network subsystem into the node's resource management system. [47] proposes a unified I/O buffering and caching system that eliminates all data copies involved in serving client requests and eliminates multiple buffering of I/O data, thus benefiting servers with numerous WAN connections, for which TCP retransmission buffers have to be maintained for long time intervals.

Caching Query Results. In the process of serving requests for dynamic content, application servers often query databases. These queries can have significant overhead. Recent research indicates that significant benefits can result from caching query results. Cached results can be used as a whole [20, 48] or as a base for sub-queries [42]. Proposed solutions allow query caching to be controlled by the application [20, 48] or by the database itself [25]. Application-controlled caching may benefit from exploiting application semantics but may be less effective for complex applications with many relations and a variety of queries. In contrast, query engine-controlled caching can exploit the engine's detailed understanding of the complete site schema. For instance, a query engine can automatically implement query reformulation in order to produce content that is likely to have higher cache utility.

Management of Dynamic Content. The framework for *dynamic content management* is a Web site component that tracks information updates and controls the pro-active recomputation of the related dynamic content. Recent research has proved the significant benefits that can result from pro-active content recomputation (and caching) versus the traditional per-request content generation [64, 17].

For instance, the approach proposed in [17] is to monitor underlying data which affect Web pages, such as database tables or files. When changes are detected, new Web pages are generated to reflect these changes. The underlying mechanism, called "trigger monitor", uses a graph to maintain relationships between the underlying data and the Web pages affected by the data. The trigger monitor uses graph traversal algorithms to determine which Web pages need to be updated as a result of changes to underlying data.

To summarize, this section has reviewed the major components of a Web site's architecture, highlighting related challenges and proposed solutions. In the remainder of this paper, we focus on several functions of a Web site application that have strong implications on the overall performance, namely the processing of client requests, the control of service levels, and the interaction with network caches.

3 Processing Client Requests

Highly accessed Web sites may need to handle peak request rates of over a million hits per minute. Web serving lends itself well to concurrency because transactions from different clients can be handled in parallel. A single Web server can achieve parallelism by multithreading or multitasking among different requests. Additional parallelism and higher throughputs can be achieved by using multiple servers and load balancing requests among the servers. Sophisticated restructuring by the programmer or compiler to achieve high degrees of parallelism is not necessary.

Web servers satisfy two types of requests, static and dynamic. *Static requests* are for files that exist at the time a request is made. *Dynamic requests* are for content that has to be generated by a server program executed at request time. A key difference between satisfying static versus dynamic requests is the processing overhead. The overhead of serving static pages is relatively low. A Web server running on a uniprocessor can typically serve several hundred static requests per second. This number is highly dependent on the data being served; for large files, the throughput is lower.

The overhead for satisfying a dynamic request may be orders of magnitude more than the overhead for satisfying a static request. Dynamic requests often involve extensive back-end processing. Many Web sites make use of databases, and a dynamic request may invoke several database accesses; these database accesses can consume significant CPU cycles. The back-end software for creating dynamic pages may be complex. While the functionality performed by such software may not appear to be compute-intensive, such middleware systems are often not designed efficiently. Many commercial products for generating dynamic data are highly inefficient.

One source of overhead in accessing databases is connecting to the database. In order to perform a transaction on many databases, a client must first establish a connection with a database in which it typically provides authentication information. Establishing a connection is often quite expensive. A naive implementation of a Web site would establish a new connection for each database access. This approach could overload the database with relatively low traffic levels.

A significantly more efficient approach is to maintain one or more long-running processes with open connections to the database. Accesses to the database are then made with one of these long-running processes. That way, multiple accesses to the database can be made over a single connection.

Another source of overhead is the interface for invoking a server program in order to generate dynamic data. The traditional method for invoking server programs for Web requests is via the Common Gateway Interface (CGI). CGI works by forking off a new process to handle each dynamic request; this incurs significant overhead. There are a number of faster interfaces available for invoking server programs [34]. These faster interfaces use one of two approaches. The first approach is for the Web server to provide an interface to allow a program for generating dynamic data to be invoked as part of the Web server process itself. IBM's GO Web server API (GWAPI) is an example of such an interface. The second approach is to establish long-running processes to which a Web server passes requests. While this approach incurs some interprocess communication overhead, the overhead is considerably less than that incurred by CGI. FastCGI is an example of the second approach [45].

In order to reduce the overhead for generating dynamic data, it is often feasible to generate the data corresponding to a dynamic page once, store the page in a cache, and to subsequently serve requests to the page from cache instead of invoking the server program again [35, 48, 64]. Using this approach, dynamic data can be served at the same rate as static data.

However, there are types of dynamic data that cannot be pre-computed and serviced from the cache. For instance, dynamic requests that cause a side effect at the server such as a database update cannot be satisfied merely by returning a cached page. For example, consider a Web site that allows clients to purchase items using credit cards. At the point at which a client commits to buying something, that information has to be recorded at the Web site; the request cannot be solely serviced from the cache.

Personalized Web pages can also present problems for caches. A personalized Web page would contain content specific to a client such as the client's name. Such a Web page could not be used for another client. Therefore, caching the page is of limited utility since only a single client can use it. Each client would need a different version of a page.

One method which can reduce the overhead for generating dynamic pages and enable caching of some parts of personalized pages is to define these pages as a collection of fragments [18]. In this approach, a complex Web page is constructed from several simpler fragments that may be recursively embedded. This is efficient because the overhead for composing an object from simpler fragments is usually minor compared to the overhead for constructing the object from scratch, which can be quite high.

The fragment-based approach also makes it easier to design Web sites. Common information that needs to be included on multiple Web pages can be created as a fragment. In order to change the information on all pages, only the fragment needs to be changed.

In order to use fragments to allow partial caching of personalized pages, the personalized information on a Web page is encapsulated by one or more fragments that are not cacheable, but the other fragments in the page are. When serving a request, a cache composes pages from its constituent fragments, many of which can be locally available. Only personalized fragments have to be created by the server. As personalized fragments typically constitute a small fraction of the entire page, generating them would require lower overhead than generating all of the fragments in the page.

Generating Web pages from fragments provides other caching benefits as well. Fragments can be constructed to represent entities that have similar lifetimes. When a particular fragment changes but the rest of the Web page stays the same, only the fragment needs to be invalidated or updated in the cache, not the entire page. Fragments can also reduce the amount of cache space taken by a collection of pages. Suppose that a particular fragment f_1 is contained in 2000 popular Web pages which should be cached. Using the conventional approach, the cache would contain a separate version of f_1 for each page resulting in as many as 2000 copies. By contrast, if the fragment-based method of page composition is used, only a single copy of f_1 needs to be maintained.

A key problem with caching dynamic content is maintaining consistent caches. The cache requires a mechanism, such as an API, allowing the server to explicitly invalidate or update cached objects that have become obsolete. Web objects may be assigned expiration times that indicate when they should be considered obsolete. Such expiration times are generally not sufficient for allowing dynamic data to be cached properly because it is often not possible to predict accurately when a dynamic page will change. This is why a mechanism is needed to allow the server to explicitly keep the cache updated.

Server performance can also be adversely affected by encryption. Many Web sites need to provide secure data to their clients via encryption. The Secure Sockets Layer protocol (SSL) [27] is the most common method for passing information securely over the Web. SSL causes serious performance degradation [4]. In addition, the overhead of providing secured interactions may be unnecessarily increased if embedded images that do not include private content are specified by the Web content designer as requiring secure transmission.

While objects encrypted via SSL generally cannot be cached within the network, they can be cached within browsers if they have expiration times. Web sites can thus considerably reduce their encryption workloads by properly setting expiration times for objects that need to be encrypted.

To summarize, this section has addressed the elements involved in the actual processing of HTTP requests that have significant overhead. Among these, the generation of dynamic content is one of the most critical, particularly for Web sites that generate significant dynamic content.

4 Control of Service Levels

The control of service levels is a major concern for many deployed Web site applications. This is motivated by business reasons, including the need to maximize resource utilization and the need to keep clients motivated to access the site by delivering responses within reasonable time. Challenges stem from the characteristics of Web site workloads that typically exhibit high burstiness of arrivals and high variance of per-request resource usage.

The problem of controlling the service levels in a Web site has been addressed in different formulations by a large body of research. The most frequently considered formulations include:

- *Single Service - Maximal Performance.* [61, 22, 63, 46, 8, 29] In the simplest formulation, a Web site provides a single type of service. The goal of service-level control is to maximize throughput and minimize response times across all of the requests received by the system.
- *Multiple Services - Differentiated Performance.* [62, 2] In a more general formulation, a Web site provides several types of services, each characterized by a relative importance coefficient. The goal of service-level control is to ensure that requests for more important services receive better service quality while less important services do not starve.
- *Multiple Services - Isolated Performance.* [6, 54, 5, 1, 10] In an alternative formulation for Web sites providing multiple services, each service is associated with a minimum performance level, know as a Service Level Agreement (SLA), and defined by bounds on performance metrics like sustainable request rate, response time, and availability. The goal of service-level control is to ensure that each service achieves the

SLA levels while the system resources are effectively exploited.

Research has addressed the problem of service level control for Web sites based on host and cluster systems. For host-based sites, solutions are defined by methods for dispatching requests to the available execution threads [2, 1] and for enforcing that these threads do not use more than service-specific shares of the CPU and disk resources [10]. For cluster-based sites, solutions are defined by methods for dispatching client requests to content delivery hosts and for performance monitoring. For both types of sites, request-dispatching solutions may include rules for discarding requests in order to prevent the system from reaching overload. In the remainder of this section, we discuss the solutions proposed for each of the three formulations of the service-level control problem, focusing on the solutions for cluster-based sites, the most relevant configuration for high performance Web sites.

Single Service - Maximum Performance. In the most frequently addressed model of service level control, all requests reaching the Web site have equal importance. The goal is to maximize the site's throughput by ensuring that the load is well balanced across all of the site's processing nodes. A secondary goal is to minimize the average response time. Sample approaches include the increase of per-node hit ratios in the memory cache and reduction of wait times in node service queues.

All solutions are based on the existence of a control component that is invoked for each new connection or request to decide to which node it should be dispatched or whether it should be dropped. The decision is based on per-node information, such as estimates of current CPU and disk loads, accessible content, and memory cache content. In addition, the decision might consider request characteristics, such as target content group, actual URL, and expected resource needs. When the identity of the requested content is a parameter of the dispatching decision, the controller includes a request analysis component.

The feature of the control mechanism that has the strongest impact on performance and scalability is the placement of its modules for dispatching decision and request analysis. The proposed solutions include the following: (1) both dispatching decision and analysis are performed by the connection router, (2) dispatching decision is made by a specialized node, while analysis is done by processing nodes, and (3) both dispatching decision and analysis are performed by each of the processing nodes.

Solutions based on connection router decisions and focusing only on throughput maximization have been the first to appear[1]. In this group, the most common request distribution policy is Weighted Round-Robin (WRR), in which the number of requests directed to a node is inversely proportional to its load [22]. Sample of per-node load estimators include the number of established connections, response times to probe requests sent by the controller, CPU utilization, and a combination of CPU and disk utilizations. Alternative policies include "Least Loaded First" and weighted "least connections" - with weights reflecting the relative capacities of the nodes [50].

While characterized by a relatively low overhead, decisions based only on load information cannot exploit

[1] refer to [50] for a survey and taxonomy of router-based solutions.

content distribution across the disk stores and memory caches of the individual servers. To address this problem, content-based routing is used. A content-based router examines each request and dispatches it to a node that has access to the necessary content and is likely to deliver it with the least overhead. This method can also be used to route certain types of requests to designated servers. For example, a content-based router could send all dynamic requests to a particular set of servers.

The drawback to content-based routing is that it introduces considerable overhead. Besides the actual overhead of reading and parsing the request, a connection management overhead is incurred. Namely, in order to examine the request content, the router must terminate the connection with the client, establish a connection to the server, and transfer messages between client and server. The transfer overhead is higher than for a regular (i.e., layer 4) router, which forwards messages between client and server without terminating the connection. In a straightforward implementation of content-based routing, the router is involved in the transfer of all of the data exchanged by client and server [61]. Better performance is achieved by using a TCP handoff protocol which transfers the client connection from the router to the server in a client-transparent manner [46, 22]. In this case, the router is only involved with the transfer of content sent by the client (mostly ACKs), which results in a much lower overhead than the transfer of all of the content sent by both server and client.

For content-based dispatching, a straightforward method is to use a fixed partition of content to nodes. However, this may lead to low throughput performance because of the high propensity for load imbalances. A viable solution considered in [46, 61] is to have the controller direct each request to a node that has recently serviced the same content if this is not heavily loaded relative to other nodes in the system. This leads to higher locality and hit rates in server caches. Load can be estimated by the number of active connections [46] or by the weighted sum of CPU and disk utilizations, with fixed weights, defined according to site characteristics [61]. The experimental evaluation presented in [46] demonstrates that a connection router using this locality-aware content-based redirection can achieve more then twice the performance of a router using WRR.

However, recent studies provide experimental evidence that content-based routing implemented by the connection router does not scale well with cluster size and processing power and is limited by the capacity of the router. Addressing this drawback, [7] propose a distributed architecture in which the request analysis is performed by the processing nodes and the dispatching decisions are made by a dedicated node. Given the relatively low overhead of decision making and load monitoring, a dispatcher node based on a 300MHz PIII machine can sustain a peak of 50,000 connections/sec, which is about an order of magnitude larger than can be achieved with a content-based front-end router that performs both request analysis and decision.

An alternative solution to the access router bottleneck is to have the request distribution decisions made by the processing nodes. In this paradigm, after analyzing a request, a node decides by itself whether to process it locally or to forward it to a peer [8, 63]. Typically, the selected peer is the one that is the least loaded, according to the current node's knowledge of its peers' loads. For instance, the proposal in [8] uses periodic broadcast to disseminate load information such as CPU load or the number of locally opened and/or redirected connections.

The solution proposed in [63] uses more complex load models and acquisition protocols targeted at maximizing the accuracy of the information used in the dispatching decision. Namely, a request is forwarded to the node with the least load provided this is significantly lower than that of the current node. Load is expressed by the *stretch factor* of the average response time, i.e., the expected increase in response time with respect to execution on an idle system. The load information may be requested at decision time, or it may be derived from possibly outdated information received previously through periodic multicast; the choice of method depends on the node's load index (i.e., weighted sum of CPU and disk utilization). This method is suitable for heterogeneous clusters, particularly for workloads with large and highly variable per-request overheads (about 10 sec average and one order of magnitude variance) and relatively low arrival rates (1-3 sec between requests), such as for a digital library.

Both [8] and [63] present experimental results that demonstrate that inter-node request redirection not only solves the scalability problem but also enables a Web site application to effectively accommodate inappropriate load distributions that may result from DNS-based routing or from a high variance of per-request overheads. For instance, [8] presents experiments run on a 3-node server, with two thirds of requests going to one node and one sixth to each of the other two. The request forwarding mechanism is based on connection transfer [12]. The study shows that redirection based on load information (sampled at 1-sec intervals) results in significant improvements relative to the no-redirection approach. Mean response time was reduced by about 70%, variance was reduced by more than an order of magnitude, and throughput increased by about 25%.

To conclude this section, we mention an important theoretical result. [29] demonstrates that the *size-based* request dispatching policy is optimal with respect to average response time. The size-based policy ensures that the requests directed to the same node have comparable sizes. The optimality result is due to the heavy-tailed distribution of Web content. [29] proposes a method for partitioning the content among the processing nodes and demonstrates by simulation that this scheme results in waiting times at least one order of magnitude lower than round-robin or least-load-first. Unfortunately, for most Web site applications, the applicability of this result is restricted by the difficulty of determining the (approximate) request sizes at the dispatcher.

Multiple Services - Differentiated Performance. After extensively addressing the problem of differentiated Web services in the context of a single host, research has recently addressed this problem in a cluster context. The method proposed in [62] aims at ensuring that the various classes of service that a Web site provides receive prioritized access to resources in accordance with their relative importance to the system. More important services should be guaranteed better service quality, in particular, under heavy load situations. In addition, under light loads, requests for less important services should not starve. Service quality is quantified by the *stretch factor* defined earlier in this section.

Using a simple queuing theory model, the authors derive basic relationships among per-class stretch factors, node assignments, resource utilization, and importance coefficients. These formulas are evaluated periodically using information on the current per-class arrival rates and CPU utilizations to determine the number of nodes to be allocated to a class and the associated request admission rates. This information is used by an access router to limit the number of requests serviced in each class and to appropriately distribute these requests among the nodes assigned to the class. The request-to-node mapping is based on a Least Loaded policy in which the relative CPU and I/O loads of the available nodes are considered.

Multiple Services - Isolated Performance. Numerous solutions for Web site service-level control enable each service in the system to perform at the levels specified by the SLA, as predictable as if the service is running on an isolated system. Toward this end, the performance parameters specified in the SLA are translated into resource reservations (e.g., CPU, disk, and network) that are enforced through system-specific mechanisms. Typical solutions are based on two-layer resource infrastructures. One layer of the infrastructure performs system-level resource management, deciding how the system-level resource reservation of a service is split into per-node reservations. The other layer of the infrastructure performs node-level resource management, enforcing the per-node service reservations.

For node-level resource managers, typical solutions are based on proportional-share resource allocation methods, like the SFQ CPU scheduler considered in [54] and the Lottery Scheduling-based "resource containers" [10] considered in [6]. Each service is assigned a CPU reservation as a percentage of the node's capacity. The resource manager schedules the thread processing in each service and appropriately charges the resource usage to the corresponding service reservations. Unallocated or unused resources on a node are fairly shared among the active services with reservations on that node.

For the system-level resource manager, the goal is to ensure that each node has enough resources to process the assigned requests. The existence of this component is necessary when client requests directly hit the server nodes based on DNS routing or when the connection router policy is based on criteria other than system load, such as content locality [46].

An effective solution for the system-level component is to start with a default distribution of each service reservation to the available nodes and to periodically adjust the per-node service reservations to accommodate the actual load distribution. A per-node service reservation is increased when the service performs at the maximum of its allocation on the node. However, adjustments do not change the system-level reservation of a service; the increase in reservation on one node is accompanied by equivalent decrease on other nodes [6, 54]. The adjustment decision is based on monitoring information regarding current per-node, per-service resource usage.

In [6], the reservation is increased by a small amount proportional to the relative difference between the default per-node allocation and the actual usage up to a bound (e.g., $\min\{5, 500 \cdot \frac{D-u}{D}\}$, where D is the default allocation and equal on all the nodes, u is the current utilization, and 5 is a sample bound on the reallocated amount). The actual decision is made by solving an optimization problem that minimizes the distance between the solution and target reservation levels.

The solution in [54], based on similar principles, explicitly accounts for the unallocated resources. The reservation increment is a percentage of the current usage. Also, this solution does not assume that the default per-node reservation is equal for all of the nodes in the system.

Both studies provide evidence that in the presence of bursty arrivals, which are typical of many Web sites, a system with dynamically adjustable per-node service reservations can achieve better resource utilization and performance than a system with fixed per-node service reservations. In addition, these solutions enable high scalability. Experimental results in [54] demonstrate that the control infrastructure scales up to 50,000 (node, service)-pairs for decision periods of 15 sec.

The proposal in [5] is similar but addresses a more complex, two-tier Web site architecture. The first tier includes the content delivery servers, which can be dynamically assigned to particular services depending on load variations. The second tier includes the content generation servers that have a static assignment to services. In addition, the site has network links shared by all the services. A connection router implements request throttling and load-based dispatching. The solution is based on an elaborate and flexible infrastructure for performance control and system management. A high-level resource manager analyzes the monitoring events. For each service, it adjusts the limits of request throttling and the allocation of first-tier servers. The limits of request throttling are determined by the load of the second-tier servers and by the utilization of network resources.

To summarize, this section has presented the main system models considered in service-level control and reviewed relevant solutions in each category of models. Despite the variety of solutions to enforcing system-specific performance goals, all of the related studies provide experimental evidence that dynamic resource allocation decisions significantly outperform the static solutions in the context of typical Web site workloads. Consequently, the run-time mechanism for service-level control is a critical component in a highly efficient, high performance Web site application.

5 Caching Within the Network

Web site performance can be further improved by caching data within the network. Cached data are moved closer to clients, reducing the latency for obtaining the data. Network bandwidth is saved because cached data has to travel over fewer links. Since aggregate network consumption is reduced, this means that latency could also be reduced for uncached as well as cached data. Web site server CPU cycles are saved because fewer requests reach the servers at the Web site.

A recent trend in architecting high-volume Web site applications is the outsourcing of static content distribution. The enabling mechanism is called a *content distribution network* (CDN) and represents an infrastructure of caches within the network that replicates content as needed to meet the demands of clients across a wide area (Figure 3). CDNs are provided by companies such as Akamai and Digital Island and are maintained in sites belonging to commercial hosting services such as Exodus.

Practically, the interaction between a Web site, its clients, and a CDN occurs as follows. A Web site pushes (or the CDN prefetches) some of its content into the CDN's caches. For instance, the content selected for replication in a CDN includes frequently requested image files. When a client is accessing the Web site for an HTML page, it is sent a customized version of the page in which some of the links to embedded objects point to object replicas in the CDN's caches rather than to the primary copies of these objects at the Web site. The replicas are selected such that the client can achieve the fastest response times. The decision is based on static information, such as geographic locations and network connectivity, and on dynamic information, such as network and cache loads.

The cache infrastructure of CDNs is distributed across wide areas to enable the delivery of Web content from locations closer to clients. Therefore, CDNs contribute to reductions in client-observed response times and in network traffic. Furthermore, CDNs benefit the service quality delivered to clients by dynamically adjusting content placement and per-site resource allocation in response to demand and network conditions.

In addition, a CDN service releases internal Web site resources that can be used to provide better service for requests for non-cacheable content. Moreover, CDNs protect a Web site from unpredictable request bursts, obviating at reasonable cost the need for excess capacity at the Web site. This is because CDNs maintain the necessary network and computation resources that are used to service several Web sites.

CDNs represent the Web site-biased alternative to the caching within the network, while Web proxy caches represent the client-biased alternative (Figure 3). For Web proxy servers, which are typically deployed to service clients within administrative and regional domains, the performance benefits from caching result from the overlap in Web content accesses across the client population. Documents accessed by multiple clients can be cached at the proxy server, reducing the need to repeatedly fetch the documents from a remote server. The performance of two commercial CDNs (Akamai and Digital Island) are discussed in [38], and an overview of Web proxy caching is contained in [56].

The remote caching provided by CDNs has several advantages over Web proxy caching. First, content is prefetched into caches directly, whereas proxy caches are populated in response to client requests. Therefore, the Web site's overhead of offloading the content is incurred only once, and not repetitively, for all proxy caches that access the site.

Second, a Web site has direct control of the content cached by CDNs. It can easily keep track of cached content and can use appropriately suited cache consistency protocols. As proxy caches are transparent to Web sites, they can only rely on object expiration times to prevent serving stale data. There is no standard protocol for a server to indicate to the proxy cache that an object has become obsolete.

Third, a CDN provides a Web site with accurate statistics about the access patterns (e.g. number of hits) to the content serviced from its caches. In contrast, the Web site has no information about the content serviced by Web proxy caches. One technique for getting around this problem is for the each cacheable page to include a small uncacheable image file. Since Web pages typically embed several image files, the relative Web site overhead for serving the uncacheable image is relatively low. It is then possible to determine the total requests for content from the site, including requests to data cached by proxy servers, by analyzing log requests for such uncacheable image files.

One advantage that proxy caching has over CDN's is that any Web site can take advantage of proxy caching with no extra cost just by setting expiration times and HTTP headers appropriately. By contrast, people running a Web site have to pay money to a CDN service in order to use one.

A possible concern for the Web site in its selection of a CDN service is the CDN's ability to cope with unexpected request rates. The quality that a CDN can provide under overwhelming requests bursts or changing network conditions depends on the amount and the distribution of its own resources. Recent research and standardization efforts have focused on enabling CDNs to interoperate towards improving their scalability, fault tolerance, and performance. The solution proposed in [13] is based on DNS-level routing among CDNs. The DNS router, more elaborate than in the case of a single site, maps client DNS server IP addresses to a geographical region and then returns the address of the CDN serving the region. The selection is based on CDN load. Upon receiving a client request, a CDN without a direct agreement with the target Web site acts as a regular Web proxy, retrieving content from the site and appropriately billing the site's CDN.

Consistency of Network Caches. A major problem with both CDN and proxy caches is maintaining cache consistency. The ideal is to achieve *strong consistency.* Strong consistency means that a cache consistency mechanism always returns the results of the latest write at the server. Due to network delays, it is generally not possible to achieve this literal notion of strong consistency in Web caches. Therefore, a more relaxed notion of strong consistency is usually used with regard to the Web, e.g. a cache consistency method is strongly consistent if it never returns data that is outdated by more than t time units, where t is the delay for sending a message from a server to a cache [23].

With current standards, strong consistency can be achieved by polling. Namely, the cache polls the server on *each* client request to see if the cached data is current. This approach results in significant message traffic sent between caches and servers. It also increases response times at caches because a request cannot be satisfied from a cache until the cache has polled the server to ensure that it is sending updated content.

Alternatively, strong consistency can be achieved by server-driven methods in which servers send update messages to caches when changes occur. This approach can minimize update traffic because update messages are only sent when data actually change. However, servers need to maintain state for cached objects indicating which caches store the objects. In addition, problems arise if a server cannot contact a cache due to network failure or the cache being down.

One approach which can reduce the overhead for server-driven cache consistency is to use leases [28, 58, 14, 59, 23, 57]. In this approach, a cache obtains a lease from a server in order to cache an object. The lease duration is the length of time for which the server must inform the cache of updates to the object. After the lease expires, the server no longer needs to inform the cache of updates to the object. If the cache wants to continue to receive update messages, it must renew its lease on the object. Lease durations can be adjusted to balance server and network overheads. Shorter leases require less storage at the server but larger numbers of lease renewal messages. In the asymptotic cases, a lease length of zero degenerates into polling, while an infinite lease length degenerates into the conventional server-driven consistency approach. In the worst case when a server is unable to communicate with a cache, the lease length bounds the maximum amount by which a cached object may be obsolete. If the server and cache are always in communication, the cached object will never be obsolete (modulo communication delays).

A variation on just using leases is to also use volume leases [58]. Volume leases are granted to a collection of objects known as a volume. In order to store an object, a cache must obtain both an object and volume lease for the object. Object leases are relatively long. Volume leases are relatively short. In the worst case when a server is unable to communicate with a cache, short volume leases bound the maximum amount by which any cached object in the volume is obsolete. The cost of maintaining the leases is low because volume leases amortize the cost of lease renewal over a large number of objects. Techniques for efficiently implementing volume leases for caching dynamic objects are presented in [57].

To summarize, this section has addressed the problem of reducing requests to Web sites by caching data within the network. We focused on CDN services, which have several advantages over traditional Web proxy caches, and techniques for maintaining cache consistency.

6 Conclusion

There are a number of open research problems concerning the improvement of Web site performance. Additional research is needed in the area of caching dynamic Web data at remote points in the network. While several methods have been developed for maintaining consistency, more work needs to be done in demonstrating the feasibility of these methods for large numbers of Web sites and remote caches. One problem that hinders work in this area is the absence of a standard protocol agreed upon by multiple vendors for maintaining cache consistency. Another problem is the difficulty researchers have in obtaining realistic data from Web sites generating significant amounts of dynamic data.

Another important area of research is related to the techniques that Web content designers can use to facilitate performance. For example, Section 3 discussed how Web pages can be constructed from fragments to improve performance and to allow portions of personalized Web pages to be cached by localizing personalized information to specific fragments. More research is needed in order to determine optimal strategies for breaking up Web pages into fragments, both for improving performance and for making it easier to design Web content.

Several aspects of the service-level control problem need to be further explored. For instance, one research topic is related to the service quality model. Current solutions take conservative approaches in ensuring that SLAs are guaranteed in all circumstances. An important open question is whether combining minimal performance level guarantees with best effort reallocation of unused resources across services leads to better performance under highly bursty request arrivals.

Another topic of importance is the implication of cooperative CDNs on Web site performance. A Web site interacts with a CDN other than the one it has explicitly selected as if the CDN were a regular Web proxy cache. Therefore, the load observed by the server is larger when its CDN directs its requests to other CDNs. More elaborate methods for CDN cooperation are required in order to guarantee that Web sites observe the offloading negotiated with their selected CDN.

References

[1] T. Abdelzaher, K.Shin, N. Bhatti, “Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach”, *Accepted to IEEE Transactions on Parallel and Distributed Systems , 2001*

[2] J. Almeida, M. Dadu, A. Manikutty, P. Cao, “Providing Differentiated Levels of Service in Web Content Hosting”, *Workshop on Internet Server Performance, 1998*

[3] Apache Software Foundation, “Apache http server project”, *http://www.apache.org/*

[4] G. Apostolopoulos, V. Peris, D. Saha, “Transport Layer Security: How much does it really cost?”, *Proceedings of IEEE INFOCOM, 1999*

[5] K. Appleby, S. Fakhouri, L.Fong, G.Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, B. Rochwerger, “Oceano - SLA Based Management of a Computing Utility”, *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, 2001*

[6] M. Aron, P. Druschel, W. Zwaenepoel, “Cluster Reserves: A Mechanism for Resource management in Cluster-based Newtork Servers”, *Proceedings of ACM Sigmetrics, 2000*

[7] M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, “Scalable Content-aware Request Distribution in Cluster-based Network Servers”, *Proceedings of Annual Usenix Technical Conference, 2000*

[8] L. Aversa, A. Bestavros, “Load Balancing a Cluster of web Servers Using Distributed Packet Rewriting”, *Proceedings of International Performance, Computing, Communications Conference, 2000*

[9] G. Banga, P. Druschel, “Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems”, *Proceedings of USENIX Symposium on Operating System Design and Implementation, 1996*

[10] G. Banga, P. Druschel, J. Mogul, “Resource containers: A new facility for resource management in server systems”, *Proceedings of USENIX Symposium on Operating Systems Design and Implementation, 1999*

[11] G. Banga, J. Mogul, “Scalable kernel performance for Internet servers under realistic loads”, *USENIX Symposium on Operating Systems Design and Implementation, 1998*

[12] A. Bestavros, M. Crovella, Mark, J. Liu, Jun, D. Martin, “Distributed Packet Rewriting and its Application to Scalable Server Architectures”, *Proceedings of the International Conference on Network Protocols, 1998*

[13] A. Biliris , C. Cranor, F. Douglis, M. Rabinovich, S. Sibal, O. Spatscheck, W. Sturm, “CDN Brokering”, *Proceedings of the International Workshop on Web Caching and Content Distribution, 2001*

[14] P. Cao, C. Liu, “Maintaining Strong Cache Consistency in the World Wide Web”, *Proceedings of the 17th International Conference on Distributed Computing Systems, 1997*

[15] V. Cardellini, M. Colajanni, P. Yu, “DNS dispatching algorithms with state estimators for scalable Web-server clusters”, *World Wide Web Journal, 2(3) 1999*

[16] J. Challenger, P. Dantzig, A. Iyengar, “A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites”, *Proceedings of SC98, 1998*

[17] J. Challenger, A. Iyengar, P. Dantzig, “A Scalable System for Consistently Caching Dynamic Web Data”, *Proceedings of IEEE INFOCOM ’99*

[18] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, P. Reed, “A Publishing System for Efficiently Creating Dynamic web Content”, *Proceedings of IEEE INFOCOM 2000*

[19] M. Colajanni, P. Yu, D. Dias, “Scheduling Algorithms for Distributed Web Servers”, *Proceedings of International Conference on Distributed Computing Systems, 1997*

[20] L. Degenaro, A. Iyengar, I. Lipkind, I. Rouvellou, “A Middleware System Which Intelligently Caches Query Results”, *Proceedings of Middleware 2000*

[21] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, P. Shenoy, “Adaptive Push-Pull of Dynamic Web Data: Better Resiliency, Scalability and Coherency”, *Proceedings of International World Wide Web Conference, 2001*

[22] D. Dias, W. Kish, R. Mukherjee, R. Tewari, “A Scalable and Highly Available Web Server”, *Proceedings of IEEE Computer Conference, 1996*

[23] V. Duvvuri, P. Shenoy, R. Tewari, “Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web”, *Proceedings of INFOCOM 2000*

[24] A. Feldmann, R. Caceres, F. Douglis, G. Glass, M. Rabinovich, “Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments”, *Proceedings of IEEE INFOCOM, 1999*

[25] D. Florescu, A. Levy, D. Suciu, K. Yagoud, “Optimization of Run-time Management of Data Intensive Web Sites”, *Proceedings of VLDB Conference, 1999*

[26] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, P. Gauthier, “Cluster-Based Scalable Network Services”, *Proceedings of ACM Symposium on Operating Systems Principles, 1997*

[27] A.Freier, P. Karlton, P. Kocher, “The SSL Protocol”, *http://home.netscape.com/ eng/ssl3/ draft302.txt*

[28] C. Gray, D. Cheriton, “Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency”, *Proceedings of the 12th ACM Symposium on Operating Systems Principles, 1989*

[29] M. Harchol-Balter, M. Crovella, C. Murta, “On Choosing a Task Assignment Policy for a Distributed Server System”, *Proceedings of Performance Tools, 1998*

[30] V. Holmedahl, B. Smith, T. Yang, “Cooperative Caching of Dynamic Content on a Distributed Web Server”, *IEEE International Symposium on High-Performance Distributed Computing, 1998*

[31] G. Hunt, G. Goldszmidt, R. King, R. Mukherjee, “Network Dispatcher: A Connection Router for Scalable Internet Services”, *Proceedings of the 7th International World Wide Web Conference, 1998*

[32] “IBM Netfinity Web Server Accelerator V2.0”, IBM Corporation, *http://www.pc.ibm.com/ us/solutions/ netfinity/ server_accelerator.html*

[33] IBM Corporation, “WebSphere Application Server”, *http://www-4.ibm.com/ software/webservers/ appserv/ pr_version4.html*

[34] A. Iyengar, J. Challenger, D. Dias, P. Dantzig, “High-Performance Web Site Design Techniques”, *IEEE Internet Computing, 4(2) 2000*

[35] A. Iyengar, J. Challenger, “Improving Web Server Performance by Caching Dynamic Data”, *Proceedings of USENIX Symposium on Internet Technologies and Systems, 1997*

[36] A. Iyengar, E. MacNair, T. Nguyen, “An Analysis of Web Server Performance”, *Proceedings of GLOBECOM, 1997*

[37] A. Iyengar, M. Squillante, L. Zhang, “Analysis and characterization of large-scale Web server access patterns and performance”, *World Wide Web, 2(1, 2), 1999*

[38] K. Johnson, J. Carr, M. Day, F. Kaashoek, “The Measured Performance of Content Distribution Networks”, *Proceedings of the International Web Caching and Content Delivery Workshop, 2000*

[39] P. Joubert, R. King, R. Neves, M. Russinovich, J. Tracey, “High-Performance Memory-Based Web Servers: Kernel and User-Space Performance”, *Proceedings of the USENIX Annual Technical Conference, 2001*

[40] B. Krishnamurthy, J. Mogul, D. Kristol, “Key differences between HTTP/1.0 and HTTP/1.1”, *Proceedings of the International World Wide Web Conference, 1999*

[41] E. Levy-Abegnoli, A. Iyengar, J. Song, D. Dias, “Design and Performance of a Web Server Accelerator”, *Proceedings of IEEE INFOCOM, 1999*

[42] Q. Luo, J. Naughton, R. Krishnamurthy, P. Cao, Y. Li, “Active Query Caching for Database Web Servers”, *WebDB (Informal Proceedings) 2000*

[43] J. Mogul, "The case for persistent-connection HTTP", *Proceedings of SIGCOMM, 1995*

[44] I. Molnar, "Answers from planet TUX: Ingo Molnar responds", *Slashdot (http://slashdot.org/ articles/ 00/07/20/ 1440204.shtml)*

[45] Open Market, "FastCGI", *http://www.fastcgi.com/*

[46] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, "Locality-Aware Request Distribution in Cluster-based Network Servers", *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 1998*

[47] V. Pai, P. Druschel, W. Zwaenepoel, "IO-Lite: A Unified I/O Buffering and Caching System", *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation 1999*

[48] K. Rajamani, A. Cox, "A Simple and Effective Caching Scheme for Dynamic Content", *Rice University CS Technical Report TR 00-371, 2000*

[49] P. Rodriguez, C. Spanner, E. Biersack, "Web Caching Architectures: Hierarchical and Distributed Caching", *Proceedings of the International Web Caching Workshop, 1999*

[50] T. Schroeder, S. Goddard, B. Ramamurthy, "Scalable Web Server Clustering Technologies", *IEEE Network, May/June 2000*

[51] A. Shaikh, R. Tewari, M. Agrawal, "On the Effectiveness of DNS-based Server Selection", *Proceedings of the IEEE INFOCOM, 2001*

[52] J. Song, E. Levy, A. Iyengar, D. Dias, "Design Alternatives for Scalable Web Server Accelerators", *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2000*

[53] J. Song, A. Iyengar, E. Levy, D. Dias, "Architecture of a Web Server Accelerator", *To appear in Computer Networks*

[54] B. Urgaonkar, P. Shenoy, "Sharc: Managing Resources in Shared Cluters", *Technical Report TR01-08, Department of Computer Science, University of Massachusetts, 2001.*

[55] A. van de Ven, "kHTTPd Linux http accelerator", *http://www.fenrus.demon.nl*

[56] Wang, J., "A Survey of Web Caching Schemes for the Internet", *ACM Computer Communication Review, 29(5), 1999*

[57] J. Yin, L. Alvisi, M. Dahlin, A. Iyengar, "Engineering server-driven consistency for large scale dynamic web services", *Proceedings of the 10th International World Wide Web Conference, 2001*

[58] J. Yin, L. Alvisi, M. Dahlin, C. Lin, "Volume Leases for Consistency in Large-Scale Systems", *IEEE Transactions on Knowledge and Data Engineering, Vol. 11 No. 4, 1999*

[59] H. Yu, L. Breslau, S. Shenker, "A Scalable Web Cache Consistency Architecture", *Proceedings of ACM SIGCOMM'99*

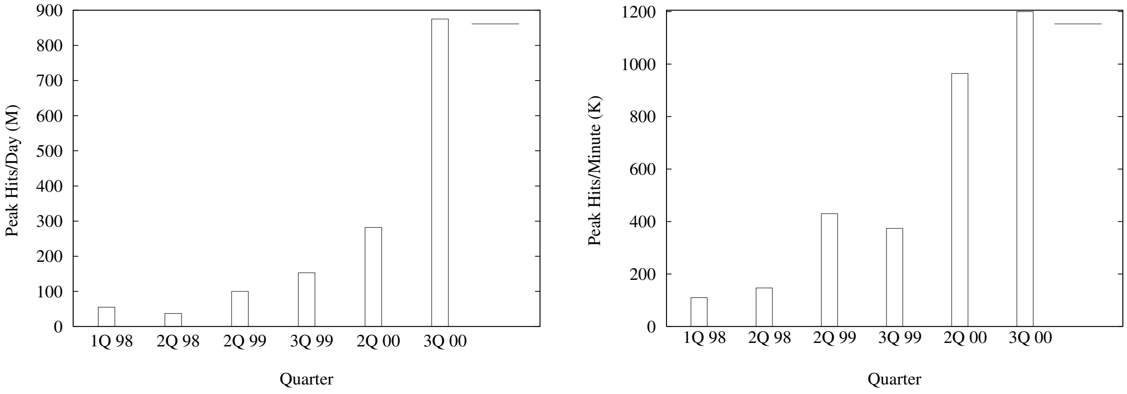
[60] Zeus Technology Limited, "Zeus Web Server", *http://www.zeus.co.uk*

[61] X. Zhang, M. Barrientos, B. Chen, M. Seltzer, "HACC: An Architecture for Cluster-Based Web Servers", *Proceedings of the USENIX Windows NT Symposium, 1999*

[62] H. Zhu, H. Tang, T. Yang, "Demand-driven Service Differentiation in Cluster-based Network Services", *Proceedings of the IEEE INFOCOM, 2001*

[63] H. Zhu, T. Yang, D. Watson, O. Ibarra, T. Smith, “Adaptive Load Sharing for Clustered Digital Library Servers”, *International Journal on Digital Libraries, 2(4) 2000*

[64] H. Zhu, T. Yang, “Class-based Cache Management for Dynamic Web Content”, *Proceedings of the IEEE INFOCOM, 2001*



(a)

(b)

Figure 1: Maximum number of hits during (a) a single day (in millions) and (b) a single minute (in thousands), at major sporting and event Web sites hosted by IBM from February 1998 through September 2000. Each bar represents a different quarter of a year.

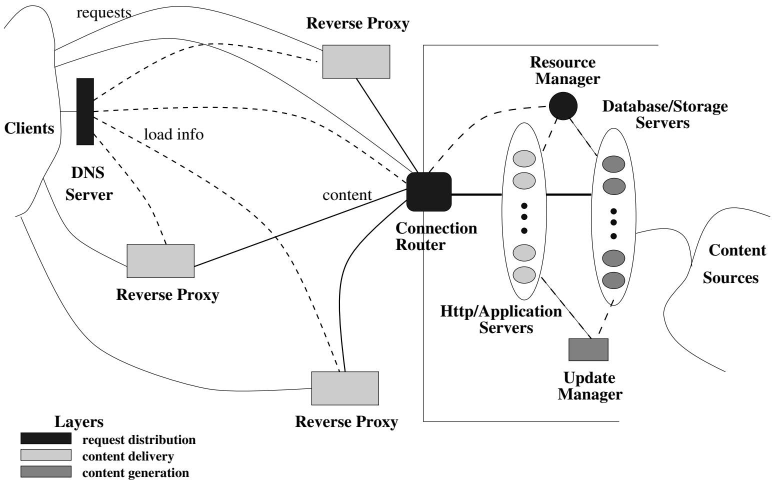


Figure 2: Components of a Web Site Architecture

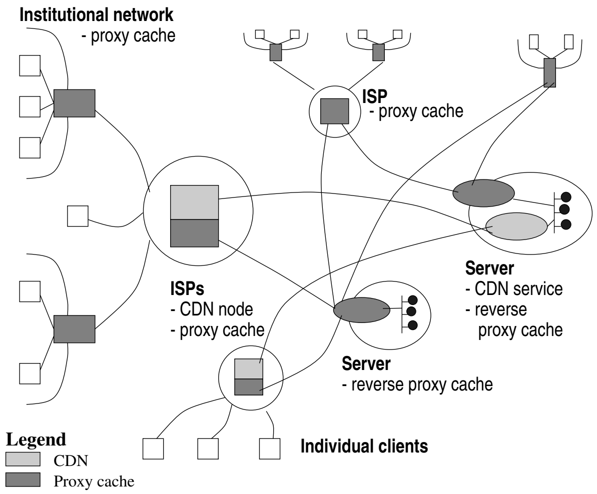


Figure 3: Internet infrastructure with proxy caches and CDNs.