A Client-Aware Dispatching Algorithm for Web Clusters Providing Multiple Services

Emiliano Casalicchio
Dept. of Computer Engineering
University of Roma Tor Vergata
Roma, Italy, 00133
ecasalicchio@ing.uniroma2.it

Michele Colajanni
Dept. of Information Engineering
University of Modena
Modena, Italy, 41100
colajanni@unimo.it

ABSTRACT

The typical Web cluster architecture consists of replicated back-end and Web servers, and a network Web switch that routes client requests among the nodes. In this paper, we propose a new scheduling policy, namely *client-aware policy* (CAP), for Web switches operating at layer-7 of the OSI protocol stack. Its goal is to improve load sharing in Web clusters that provide multiple services such as static, dynamic and secure information. CAP classifies the client requests on the basis of their expected impact on main server resources, that is, network interface, CPU, disk. At run-time, CAP schedules client requests reaching the Web cluster with the goal of sharing all classes of services among the server nodes. We demonstrate through a large set of simulations and some prototype experiments that dispatching policies aiming to improve locality in server caches give best results for Web publishing sites providing static information and some simple database searches. When we consider Web sites providing also dynamic and secure services, CAP is more effective than state-of-the-art layer-7 Web switch policies. The proposed client-aware algorithm is also more robust than server-aware policies whose performance depends on optimal tuning of system parameters, very hard to achieve in a highly dynamic system such as a Web site.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems; C.4 [**Performance of Systems**]: Design studies; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms

Algorithms, Design, Performance

Keywords

Load balancing, Dispatching algorithms, Clusters

1. INTRODUCTION

The need to optimize the performance of popular Web sites is producing a variety of novel architectures. Geographically distributed Web servers [21, 10] and proxy server systems aim to decrease user latency time through network access redistribution and reduction of amount of data transferred, respectively. In this paper we consider different Web systems, namely *Web clusters*, that use a tightly coupled distributed architecture. ¿From the user's point of view, any HTTP request to a Web cluster is presented to a logical (front-end) server that acts as a representative for the Web site. This component called *Web switch* retains transparency of the parallel architecture for the user, guarantees backward compatibility with Internet protocols and standards, and distributes all client requests to the Web and back-end servers. Cluster architectures with Web switch dispatcher(s) have been adopted with different solutions in various academic and commercial Web clusters, e.g. [2, 7, 12, 17, 20]. Valuable recent surveys are in [25, 23].

One of the main operational aspects of any distributed system is the availability of a mechanism that shares the load over the server nodes. Numerous *global scheduling*[1] algorithms were proposed for multi-node architectures executing parallel or distributed applications.

Unlike geographically distributed Web sites, where the dispatching role is taken by system components (e.g., DNS) that have only a limited control on client requests [14, 17], the Web cluster with a single Web switch controlling all workload is a very robust architecture to front Web arrivals that tends to occur in waves with intervals of heavy peaks. The motivations for a new dispatching policy come from two main considerations on Web service distributions.

The service time of HTTP requests may have very large or infinite variance even for traditional Web publishing sites. Moreover, heterogeneity and complexity of services and applications provided by Web sites is continuously increasing. Traditional sites with most static contents have being integrated with recent Web commerce and transactional sites combining dynamic and secure services.

Web switch policies that want to dispatch the requests in a highly heterogeneous and dynamic system by taking into account server states require expensive and hard to tuning

[1] In this paper we use the definition of *global scheduling* given in [11], and *dispatching* as synonymous.

WWW10, May 1-5, 2001, Hong Kong.

mechanisms for monitoring and evaluating the load on each server, gathering the results, combining them, and taking real-time decisions.

For these reasons, we propose a *client-aware policy* (CAP) policy that, in its pure version, takes dynamic decisions by looking only at client requests instead of server states. CAP partitions the Web cluster services into broad classes on the basis of the expected impact that each request may have on main server components, that is, network interface, CPU, disk. Then, it schedules client load by taking into account the service class each request belongs to. Under workload characteristics that resemble those experienced by real Web sites, we demonstrate through simulation and prototype experiments that this simple client-aware policy is much more effective than state-of-the-art dynamic algorithms when applied to Web clusters providing heterogeneous services such as static, dynamic and secure information. By using CAP, the 90-percentile of page latency time can be half of that of commonly used dynamic Web switch policies, such as Weighted-Round Robin [20] and LARD [24]. Moreover, CAP guarantees stable results for a wide range of Web sites because it does not need a hard tuning of parameters for each type of Web site as most server-aware policies require.

The remainder of this paper is organized as follows. In Section 2, we outline the typical architecture of a Web cluster with a focus on Web switches. In Section 3, we discuss some related work on global scheduling algorithms for the Web switch and propose the CAP policy. In Section 4, we present a detailed simulation model for the Web cluster, and we discuss the results for various classes of Web sites. In Section 5, we describe a prototype of Web switch operating at layer-7 that implements CAP and other policies, and analyze experimental results in a controlled environment. In Section 6, we give our final remarks.

2. WEB CLUSTERS

2.1 Architecture

A Web cluster refers to a Web site that uses two or more server machines housed together in a single location to handle user requests. Although a large cluster may consist of dozens of Web servers and back-end servers, it uses one hostname to provide a single interface for users. To have a mechanism that controls the totality of the requests reaching the site and to mask the service distribution among multiple servers, Web clusters provide a single virtual IP address that corresponds to the address of the front-end server(s). Independently of the mechanism that existing Web clusters use to routing the load, we refer to this entity as the *Web switch*. The Domain Name Server(s) for the Web site translates the site address (e.g., www.site.edu) into the IP address of the Web switch. In such a way, the Web switch acts as a centralized global scheduler that receives the totality of the requests and routes them among the servers of the cluster (see Figure 1).

We consider a Web cluster consisting of homogeneous distributed servers that provide the same set of documents and services. The details about the operations of the Web cluster are described in Section 4.1. Various academic and commercial products confirm the increasing interest in these distributed Web architectures. In the *IBM TCP router* [17], all HTTP requests reach the Web switch that distributes them by modifying the destination IP address of each incoming packet: the Web switch replaces its IP address with the private address of the selected Web server.

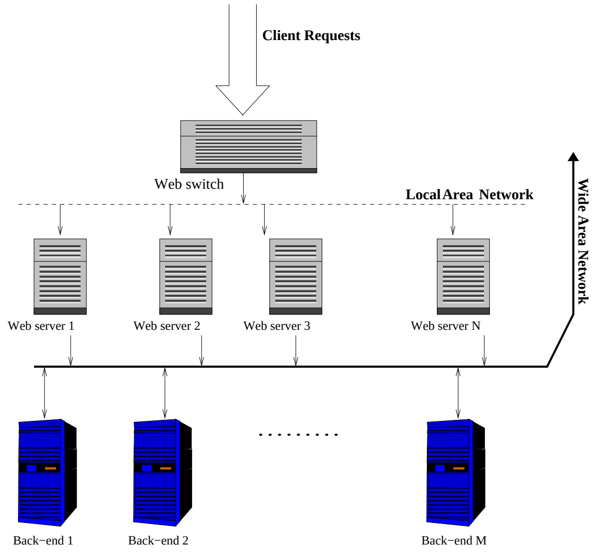


Figure 1: Web cluster architecture.

Magicrouter [2], *Distributed Packet Rewriting* [7] and Cisco *LocalDirector* [12] are other Web cluster architectures relying on a Web switch that receives the totality of client requests. In particular, Magicrouter is a mechanism of fast packet interposing where a user level process acting as a switchboard intercepts network packets and modifies them by changing addresses and checksum fields. Cisco LocalDirector rewrites the IP header information of each incoming packet according with a dynamic table of mapping between each session and the server to which it has been redirected. Unlike the TCP router that modifies only the client-to-server packets and lets the servers modify outgoing IP packets, Magicrouter and LocalDirector Web switches can be defined as *gateways* because they intervene even on server-to-client packets.

An evolution of the TCP router architecture is represented by the IBM *Network Dispatcher* that does not require a modification of the packet addresses because packet forwarding to cluster nodes is done at the MAC address level [20]. A different forwarding approach to configure a Web system with multiple servers uses the if-config-alias option, which is available in most UNIX platforms [16]. This architecture publicizes the same secondary IP address of all Web servers as the IP single virtual address, namely *ONE-IP*, of the Web cluster. This is achieved by letting the servers of the cluster share the same IP address as their secondary address, which is used only for the request distribution service.

2.2 Web switches

A key component of any Web cluster is the Web switch that dispatches client requests among the servers. They can be broadly classified according to the OSI protocol stack layer at which they operate, so we have *layer-4* and *layer-7* Web switches [25].

Layer-4 Web switches work at TCP/IP level. Since packets pertaining to the same TCP connection must be assigned to the same server node, the Web switch has to maintain a binding table to associate each client TCP session with the target server. The switch examines the header of each inbound packet and on the basis of the flag field determines whether the packet pertains to a new or an existing connection. Layer-4 Web switch algorithms are *content information blind*, because they choose the target server when the client establishes the TCP/IP connection, before sending out the HTTP request. Global scheduling algorithms executable at the layer-4 Web switch range from static algorithms (say, random, round-robin) to dynamic algorithms that take into account either network client information, (say, client IP address, TCP port), or server state information (say, number of active connections, least loaded server) or even a combination of both information.

Layer-7 Web switches can establish a complete TCP connection with the client and inspect the HTTP request content prior to decide about dispatching. In such a way, they can deploy *content information aware* distribution, by letting the Web switch examine the HTTP request and then route it to the target server. The selection mechanism (usually referred to as *delayed binding*) can be based on the Web service/content requested, as URL content, SSL identifiers, and cookies. In [5] there are many techniques to realize the dispatching granularity at the session level or at the single Web object request level. Scheduling algorithms deployed at layer-7 may use either client information (as session identifiers, file type, file size) or a combination of client and server state information. The potential advantages of layer-7 Web switches include the possibility to use specialized Web server nodes and partition the Web content among clusters of heterogeneous servers [28], and to achieve higher cache hit rates, for example, through affinity-based scheduling algorithms such as the LARD policy [24]. On the other hand, layer-7 routing introduces additional processing overhead at the Web switch and may cause this entity to become the system bottleneck. To overcome this drawback, design alternatives for scalable Web server systems that combine content blind and content aware request distribution have been proposed in [6, 26]. These architecture solutions are out of the scope of this paper which is more focused on the dispatching algorithms for Web switches.

3. WEB SWITCH ALGORITHMS

The Web switch may use various global scheduling policies to assign the load to the nodes of a Web cluster. Global scheduling methods were classified in several ways, depending on different criteria. The main alternatives are between load balancing *vs.* load sharing problems, centralized *vs.* distributed algorithms, static *vs.* dynamic policies. The Web cluster architecture with a single Web switch motivates the choice for *centralized* scheduling policies. If we consider that load balancing strives to equalize the server workload, while *load sharing* attempts to smooth out transient peak overload periods on some nodes, a Web switch should aim to share more than to balance cluster workload. Hence, the real alternative for layer-4 and layer-7 Web switches is the kind of system information they use to take assignment decisions. The main classes of policies are static and dynamic, these latter with several subclasses.

3.1 Static and dynamic global scheduling

Static policies do not consider any system state information. Typical examples are *Random* (RAN) and *Round-Robin* (RR) algorithms. RAN distributes the arrivals uniformly through the nodes. RR uses a circular list and a pointer to the last selected server to take dispatching decisions. *Dynamic policies* use some system state information while taking scheduling decisions.

We consider the three classes of dynamic algorithms. *Server-aware algorithms* route requests on the basis of some server state information, such as load condition, latency time, availability or network utilization. *Client-aware algorithms* route requests on the basis of some client information. Layer-4 Web switches can use only some basic client network information, such as IP address and TCP port. Layer-7 Web switches can examine the entire HTTP request and take decisions on the basis of detailed information about the content of the client request. *Client- and server-aware algorithms* route requests on the basis of client and server state information. Actually, most of the existing client-aware algorithms belong to this class. Indeed, although the most important information is the client request, these policies combine it with some information about the server loads. The main goal is to avoid assignments to overloaded servers.

The Web switch cannot use highly sophisticated algorithms because it has to take fast decision for dozens or hundreds of requests per second. To prevent the Web switch becoming the primary bottleneck of the Web cluster, static algorithms are the fastest solution because they do not rely on the current state of the system at the time of decision making. For this reason, these algorithms can potentially make poor assignment decisions. Dynamic algorithms have the potential to outperform static algorithms by using some state information to help dispatching decisions. On the other hand, dynamic algorithms require mechanisms that collect and analyze state information, thereby incurring potentially expensive overheads.

In this paper, we consider three widely used dispatching policies that are based on client and/or server information: *Weighted Round Robin* (WRR), *Locality Aware Request Distribution* (LARD) and *StaticPartitioning.* WRR has resulted the layer-4 policy that guarantees best load sharing in most simulations and experiments from several research groups. On the other hand, we do not expect LARD to work well in a site providing heterogeneous services, but we have chosen it because we are not aware of other layer-7 dispatching algorithms proposed by the research community. *StaticPartitioning* uses dedicated servers for specific services or multiple Web sites (co-location). This is the most representative example of a client-aware algorithm working at layer-7 in commercial Web switches [1, 19].

WRR comes as a variation of the round robin policy. WRR associates to each server a dynamically evaluated weight that is proportional to the server load state [20]. Periodically (every T_{gat} seconds), the Web switch gathers this information from servers and computes the weights. WRR is actually a class of dynamic policies that uses some information about the system state. The first issue that needs to be addressed when we consider a server state aware policy is how to compute the load state information because it is not immediately available at the Web switch. The three main factors that affect the latency time are loads on CPU, disk and network resources. Typical load measures are the number of active processes on server, mean disk response time, and hit latency time, that is, the mean time spent by each request at the server. In particular, the load indexes we consider are the number of active processes at each server

(*WRR_num* policy), and the mean service time for the requests (*WRR_time* policy). Additional information on WRR can be found in [20].

If we consider Web clusters of homogeneous servers, the main goal of the proposed policies is to augment disk cache hit rates, for example through the LARD policy [24] or other affinity-based scheduling algorithms [26, 29]. The LARD policy [24] is a content based request distribution that aims to improve the cache hit rate in Web cluster nodes. The principle of LARD is to direct all requests for a Web object to the same server node. This increases the likelihood to find the requested object into the disk cache of the server node. We use the LARD version proposed in [24] with the multiple hand-off mechanism defined in [5] that works for the HTTP/1.1 protocol. LARD assigns all requests for a target file to the same node until it reaches a certain utilization threshold. At this point, the request is assigned to a lowly loaded node, if it exists, or to the least loaded node. To this purpose, LARD defines two threshold parameters: T_{low} denoting the upper bound of a lowly loaded condition, and T_{high} denoting the lower bound of a highly loaded condition.

3.2 Client-aware policy

All previously proposed scheduling policies take static decisions independently of any state information (e.g., RAN and RR) or they take dynamic decisions on the basis of the state of the server nodes (e.g., WRR) that can be combined with client request information (e.g., LARD). We propose a *client-aware policy* (CAP) that takes into account some information associated to client requests as it can be gotten by a layer-7 Web switch. CAP, in its basic form, is a pure client-aware policy, however, it can be easily combined with some server state information. In this paper, we consider the pure CAP that does not gather any load information from servers. Pure client-aware policies have a possible great advantage over server-aware policies because server-aware algorithms often require expensive and hard to tuning mechanisms for monitoring and evaluating the load on each server, gathering the results, and combining them to take scheduling decisions. In a highly dynamic system such as a Web cluster this state information becomes obsolete quickly.

The key idea for CAP comes from the observation that dynamic policies such as WRR and LARD work fine in Web clusters that host traditional Web publishing services. In fact, most load balancing problems occur when the Web site hosts heterogeneous services that make an intensive use of different Web server's components. Moreover, almost all commercial layer-7 Web switches use client information for a static partitioning of the Web services among specialized servers [1, 19]. The simulation experiments will confirm the intuition that a *StaticPartitioning* policy, although useful from the system management point of view, achieves poor server utilization because resources that are not utilized cannot be shared among all clients. To motivate the CAP policy, let us classify Web services into four main categories.

Web publishing sites providing static information (e.g., HTML pages with some embedded objects) and dynamic services that do not intensively use server resources (e.g., result or product display requests). The content of dynamic requests is not known at the instant of a request, however, it is generated from database queries whose arguments are known before hand.

Web transaction sites providing dynamic content generated from (possibly complex) database queries built from user data provided by an HTML form. This is a disk bound service because it makes intensive use of disk resources.

Web commerce sites providing static, dynamic and secure information. For security reasons, some dynamically generated content may need a secure channel that in most cases is provided by the SSL protocol. Cryptography makes intensive use of CPU resources. Hence, Web commerce services are disk and/or CPU bound.

Web multimedia sites providing streaming audio and video services. In this paper, we do not consider this type of application that often is implemented through specialized servers and network connections.

Although realistic, this classification is done for the purposes of our paper only and does not want to be a precise taxonomy for all Web services. The idea behind the CAP policy is that, although the Web switch can not estimate precisely the service time of a client request, from the URL it can distinguish the class of the request and its impact on main Web server resources. Any Web content provider can easily tune the CAP policy at its best. Starting from the above classification, we distinguish the Web requests into four classes: *static* and *lightly dynamic* Web publishing services (N); *disk bound* services (DB), for example, in Web transaction and Web commerce sites; *CPU bound* (CB) and *disk and CPU bound* (DCB) services, for example, in Web commerce sites. In the basic version of CAP, the Web switch manages a circular list of assignments for each class of Web services. The goal is to share multiple load classes among all servers so that no single component of a server is overloaded. When a request arrives, the Web switch parses the URL and selects the appropriate server. We describe the CAP behavior through the following example.

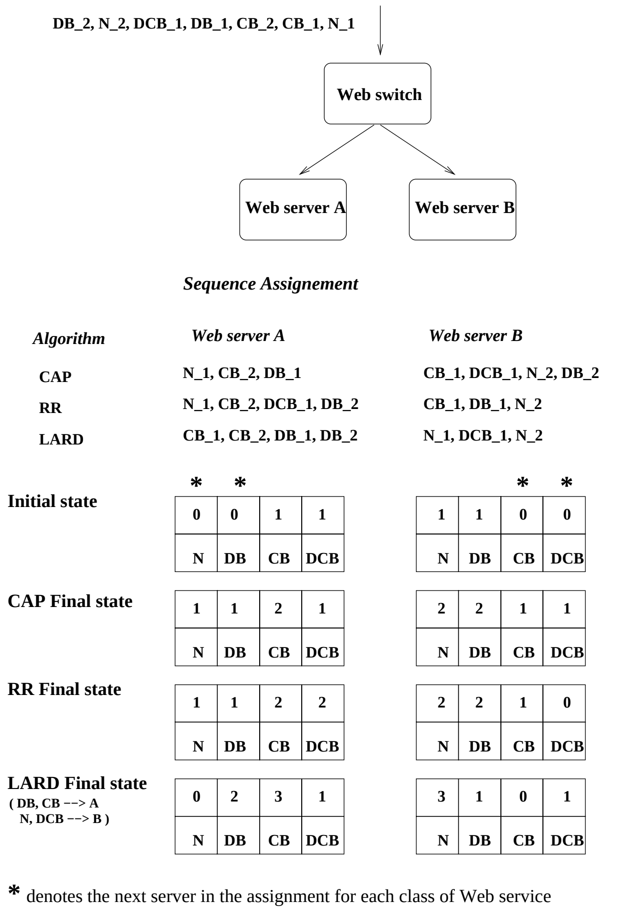
We suppose that the server A has already received one request of type CB and one of type DCB; the server B has received one request of type N, and one of type DB. The sequence of successive requests to the Web cluster is shown in Figure 2. By using the CAP assignment, server A and B have a similar number of requests for each class of service, while this does not happen when using RR or LARD. For example, in the case of RR the server A receives four intensive requests that stress the CPU and/or disk, while server B receives only one CPU bound request. In the case of LARD, we suppose that the requests of type DB and CB are assigned to the server A and those of other types to the server B. This dispatching results that the server A receives two CPU bound and two disk bound requests, while the server B receives only one request of type DCB.

CAP does not require a hard tuning of parameters which is typical of most dynamic policies because the service classes are decided in advance, and the scheduling choice is determined statically once the URL has been classified.

4. SIMULATION EXPERIMENTS

4.1 System model

The Web cluster consists of multiple Web servers and back-end servers, and a dedicated machine that can act as


DB_2, N_2, DCB_1, DB_1, CB_2, CB_1, N_1
Web switch
Web server A
Web server B
Sequence Assignement
* denotes the next server in the assignment for each class of Web service

Algorithm	Web server A	Web server B
CAP	N_1, CB_2, DB_1	CB_1, DCB_1, N_2, DB_2
RR	N_1, CB_2, DCB_1, DB_2	CB_1, DB_1, N_2
LARD	CB_1, CB_2, DB_1, DB_2	N_1, DCB_1, N_2

	Web server A N	Web server A DB	Web server A CB	Web server A DCB	Web server B N	Web server B DB	Web server B CB	Web server B DCB
Initial state	0 *	0 *	1	1	1	1	0 *	0 *
CAP Final state	1	1	2	1	2	2	1	1
RR Final state	1	1	2	2	2	2	1	0
LARD Final state (DB, CB --> A N, DCB --> B)	0	2	3	1	3	1	0	1

Figure 2: Example of behavior of CAP, RR and LARD dispatching policies.

a layer-4 or layer-7 Web switch. The primary DNS translates the hostname of this site into the IP address of the Web switch. The addresses of Web and back-end servers are private and invisible to the extern. Web servers, back-end servers and Web switch are connected through a local fast Ethernet with 100 Mbps bandwidth, as in Figure 1. As the focus is on Web cluster performance we did not model the details of the external network. To prevent the bridge(s) to the external network becoming a potential bottleneck for the Web cluster throughput, we assume that the system is connected to the Internet through one or more large bandwidth links that do not use the same Web switch connection to Internet [20].

Each server in the cluster is modeled as a separate component with its CPU, central memory, hard disk and network interface. All above components are CSIM processes having their own queuing systems that allow for requests to wait if the server, disk or network are busy. We use real parameters to set up the system. For example, the disk is parameterized with the values of a real fast disk (IBM Deskstar34GXP) having transfer rate equal to 20 MBps, controller delay to 0.05 msec., seek time to 9 msec., and RPM to 7200. The main memory transfer rate is set to 100MBps. The network interface is a 100Mbps Ethernet card. The back-end servers are replicated database servers that reply to dynamic requests. The Web server software is modeled as an Apache-like server, where an HTTP daemon waits for requests of client connections. As required by the HTTP/1.1 protocol, each HTTP process serves all files specified in a Web request.

The client-server interactions are modeled at the details of TCP connections including packets and ACK signals. Each client is a CSIM process that, after activation, enters the system and generates the first request of connection to the Web switch of the Web cluster. The entire period of connection to the site, namely, *Web session*, consists of one or more page requests to the site. At each request, the Web switch applies some routing algorithm and assigns each connection to a server. The server dedicates a new HTTP process for that connection. Each client request is for a single HTML page that typically contains a number of embedded objects. A request may include some computation or database search, and a secure connection. The client will submit a new request only after it has received the complete answer, that is, the HTML page and all (possible) embedded objects. A user think time between two page requests models the time required to analyze the requested object and decide about a new request.

It is worth observing that for a fair comparison of the Web switch algorithms our models consider the overhead difference in dispatching requests at layer-4 and layer-7 Web switches. For example, the delays introduced by layer-7 switching are modeled with the values given in [5].

4.2 Workload model

Special attention has been devoted to the workload model that incorporates all most recent results on the characteristics of real Web load. The high variability and self-similar nature of Web access load is modeled through heavy tail distributions such as Pareto, lognormal and Weibull distributions [4, 8, 9, 15]. Random variables generated by these distributions can assume extremely large values with non-negligible probability.

The number of *page requests* per client session, that is, the number of consecutive Web requests a user will submit to the Web site, is modeled according to the inverse Gaussian distribution. The time between the retrieval of two successive Web pages from the same client, namely the *user think time*, is modeled through a Pareto distribution [9]. The number of *embedded objects* per page request including the base HTML page is also obtained from a Pareto distribution [9, 22]. The *inter-arrival time of hit requests*, that is, the time between retrieval of two successive hit requests from the servers, is modeled by a heavy-tailed function distributed as a Weibull. The distribution of the file sizes requested to a Web server is a hybrid function, where the body is modeled according to a lognormal distribution, and the tail according to a heavy-tailed Pareto distribution [8]. A summary of the distributions and parameters used in our experiments is in Table 1.

To characterize the different Web services classified as in Section 3.2 we have modeled also the impact of a secure channel on server performance (that is, the presence of CPU bound requests) and the impact of intensive database queries (that is, disk bound requests). Our model includes all main CPU and transmission overheads due to SSL protocol interactions, such as key material negotiation, server authentication, and encryption and decryption of key mate-

Web cluster	
Number of servers	2-32
Disk transfer rate	20 MBps
Memory transfer rate	100 MBps
HTTP protocol	1.1
Intra-servers bandwidth	100 Mbps
Clients	
Arrival rate	100-300 clients per second
Requests per session	Inverse Gaussian ($\mu = 3.86$, $\lambda = 9.46$)
User think time	Pareto ($\alpha = 1.4$, $k = 2$)
Objects per page	Pareto ($\alpha = 1.1 - 1.5(1.33)$, $k = 1$)
Hit size request (*body*) (*tail*)	Lognormal ($\mu = 7.640$, $\sigma = 1.705$) Pareto ($\alpha = 1.383$, $k = 2924$)

Table 1: Parameters of the system and workload model.

rial and Web information. The CPU service time consists of encryption of server secret key with a public key encryption algorithm such as RSA, computation of Message Authentication Code through a hash function such as MD5 or SHA, and data encryption through a symmetric key algorithm, such as DES or Triple-DES. Most CPU overhead is caused by data encryption (for large size files), and public key encryption algorithm (RSA algorithm), that is required at least once for each client session, when the client has to authenticate the server. The transmission overhead is due to the server certificate (2048 bytes) sent by the server to the client the server hello and close message (73 bytes), and the SSL record header (about 29 bytes per record). Table 2 summarizes the throughput of the encryption algorithm used in the *secure* workload model.

Category	*Throughput (Kbps)*
RSA(256 bit)	38.5
Triple DES	46886
MD5	331034

Table 2: Secure workload model.

We compare the performance of different scheduling policies for Web clusters under three main classes of workload.

Web publishing site containing *static* and *lightly dynamic* documents. A static document resides on the disk of the Web server; it is not modified in a relatively long time interval and is always cacheable. The cache of each node is set to 15% of the total size of the Web site document tree. A lightly dynamic document is cacheable with 0.3 probability.

Web transaction sites contain 60% of static documents and 40% of dynamically created documents. Database queries to back-end servers require intensive disk use and their results are not cacheable.

Web commerce sites have 30% of static requests, 30% of lightly dynamic requests and various combinations for the remaining 40% of requests.

4.3 Simulation results

As the main measure for analyzing the performance of the Web cluster we use the cumulative frequency of the *page latency time*. The goal of this paper on Web cluster performance allows us not to include all delays related to Internet in page latency time.

4.3.1 Optimal tuning of server-aware policies

Most dynamic policies depend on system state information that is not immediately available at the Web switch. It is often expensive to gather server state information, and it is difficult to choose the best parameters of dynamic policies in highly variable and heterogeneous Web systems. The CAP policy, in its basic form, does not require setting any parameter other than the choice of classes of services that can be identified from the URL. The LARD strategy requires setting the parameters T_{low} and T_{high} for server utilization as described in Section 3. In our experiments we set T_{low}=0.3 and T_{high}=0.7, and we did not observed considerable changes for lightly different values.

The most complex policy to tune is the WRR that is sensible to the adopted load metric and to the selected T_{gat} value for gathering server state information. To show the difficulty of optimal tuning parameters of some dynamic policies in Figure 3 we show the sensitivity of WRR_num and WRR_time with respect to the T_{gat} period for an almost static (Web publishing) and a highly variable workload (Web commerce). As a performance metrics, we use the *90-percentile* of page latency time, that is, the page latency time limit that the Web site guarantees with 0.9 probability. The T_{gat} value has a big influence on performance especially for sites with heterogeneous services. If not well tuned, a dynamic policy such as WRR can behave worse than static policies such as RAN and RR. In both instances, the number of active connections (WRR_num) seems to be the best load metric and low T_{gat} values seem to be preferable to higher values. In the remaining part of the paper, we use the number of active processes as a load metric and assume that we are always able to choose the best parameters for the WRR policy. We will refer to it simply as WRR.

In the simulations, we consider the ideal StaticPartitioning algorithm that for any workload scenario is able to partition the dedicated servers proportionally to the percentage arrival of requests for each class of services.

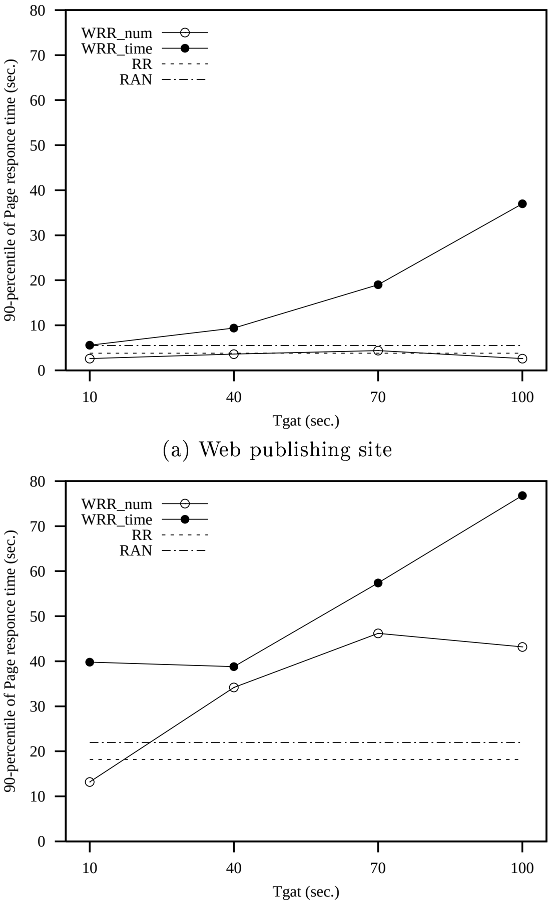
4.3.2 Web publishing

We assume a first scenario where all requests are for static documents and a second scenario where requests are for lightly dynamic documents. Figure 4(a) shows that the LARD strategy that exploits the reference locality of Web requests performs better than CAP and WRR. LARD guarantees with very high probability that the page latency time is less than 1 second. The analogous probability is about 0.9 for CAP and 0.8 for WRR.

In the next experiments, we consider only requests for lightly dynamic documents that have a lower probability of being found in the disk cache. Figure 4(b) shows that CAP and LARD have a similar behavior even if LARD still performs slightly better. However, if we compare this figure with Figure 4(a) we can observe that, while CAP and WRR maintain similar results, actually LARD is truly penalized by a lower cache hit rate.

4.3.3 Web transaction

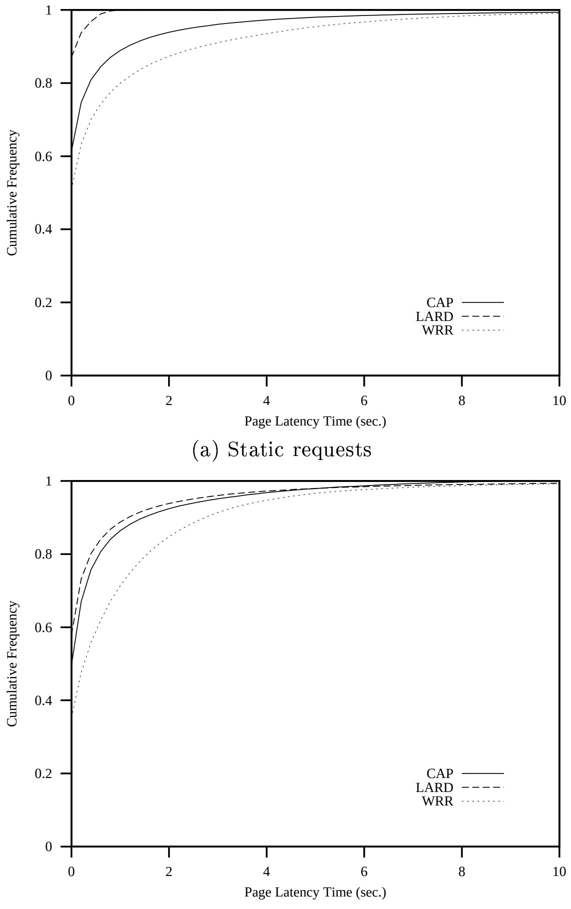
When the document hit rate is low and a relatively high percentage of requests (40%) is disk bound, the CAP policy starts to exploit its benefits for multiple service characteristics. Figure 5 shows that the page latency time of CAP is much better than that achieved by WRR, LARD and Static-Partitioning algorithms. CAP policy improves Web cluster



(a) Web publishing site

(b) Web commerce site

Figure 3: Sensitivity of WRR policies to the T_{gat} parameter and load metric.



(a) Static requests

(b) Dynamic requests

Figure 4: Web publishing: cumulative frequency of page latency time.

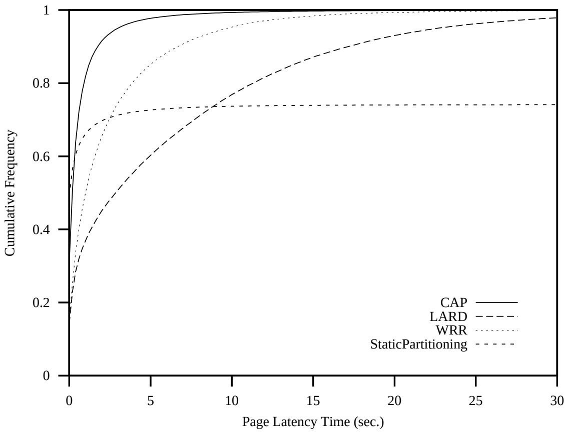


Figure 5: Web transaction: cumulative frequency of page latency time

performance of 35% over WRR and 55% over LARD. Indeed, the 90-percentile of page latency time using CAP is less than 2 seconds, while it is about 7 seconds for WRR and about 20 seconds for LARD. LARD show bad results because incoming requests are assigned to a Web server first on the basis of document locality and then on the basis of server load state. Our results confirm the intuition that routing requests to the same server until it reaches a highly loaded state does not work in Web transaction sites. Static-Partitioning performs even worse than LARD. On the other hand, the layer-4 WRR policy is quite robust, because its performance does not degrade too much for sites with heterogeneous services.

4.3.4 Web commerce

This is the most heterogeneous workload. We consider three scenarios where lightly dynamic, CPU bound and disk bound requests are mixed in different ways.

In the first set of experiments we assume that 40% of total requests need secure connections and cryptography. This is the most critical scenario because CPU bound requests affect performance of the other 60% of requests that have to use the CPU, even if for less intensive operations such as parsing a request and building a HTTP response. Figure 6(a) shows how this very critical (and, we can say, unrealistic) scenario deteriorates performance of all dispatching strategies. However, multiple service scheduling allows CAP to achieve still best results.

In the second scenario, we introduce both CPU and disk bound requests, but CPU bound requests (20%) differ from disk bound requests (20%). In Figure 6(b) we see that the CAP policy provides really good page latency times, while WRR is at its limit and LARD does not guarantee a scalable Web cluster. The improvement of the CAP strategy is considerable, especially if we consider that the WRR curve refers to the WRR policy with best parameters. For example, CAP achieves a page latency time of about 2.5 seconds with 0.9 probability. For WRR and StaticPartitioning, the analogous latency time is achieved with a probability of about 0.68, and for LARD with a probability of less than 0.4.

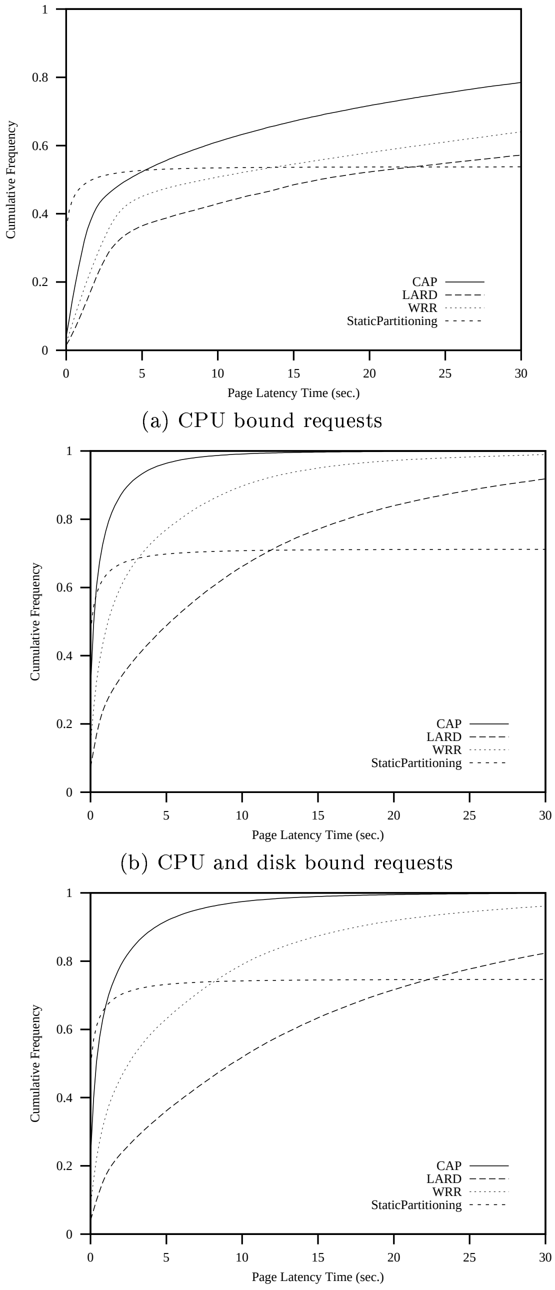
In the last set of experiments, we consider a realistic Web commerce site where the requests can be for secure (20%), disk bound (10%) or both (10%) services. Even if the workload becomes more onerous, Figure 6(c) shows that the CAP policy guarantees a scalable Web cluster. Indeed, it achieves a 90-percentile for the page latency time of less than 5 seconds. The analogous percentile is equal to 20 seconds for WRR and it is higher than 35 seconds for LARD. For example, WRR, LARD and StaticPartitioning have a page latency time of 5 seconds with a probability of 0.66, 0.72 and 0.37, respectively.

5. PROTOTYPE EXPERIMENTS

The simulation results conviced us about the opportunity of building a prototype cluster with a layer-7 Web switch equipped with CAP. In this section we outline the architecture and give some performance results that confirm main conclusions obtained by the simulation model.

5.1 Prototype architecture

The Web cluster consists of a Web switch node, connected to the back-end nodes and the Web servers through a high speed LAN. The distributed architecture of the cluster is hidden to the HTTP client through a unique Virtual IP (VIP) address. Different mechanisms were proposed to implement a layer-7 Web switch at various operating system levels. The most efficient solutions are the TCP hand-off [24] and the TCP splicing [13] that are implemented at the kernel level. The application layer solutions are undoubtely less efficient than kernel level mechanisms, but their implementation is cheaper and sufficient for the purposes of this paper that is focused more on dispatching algorithms than on Web switch products. Among the application level solutions, we selected the reverse proxy approach proposed in [18] that is based on the Apache Web server software. This mechanism allows us to implement and test any layer-7 dispatching algorithm without modifications at the kernel level. The drawback of this approach is a higher overhead on response time perceived by the clients. Aron et al. show that TCP hand-off outperforms TCP splicing techniques [6]. Moreover, the overheads of Web switches operating at layer-7 pose serious scalability problems that we, as other authors [26], noticed in the experiments. There is no doubt that a real system should work at the kernel level but addressing the switch performance issue is out of the scope of this paper.



(a) CPU bound requests

(b) CPU and disk bound requests

(c) CPU and/or disk bound requests

Figure 6: Web commerce scenario: cumulative frequency of page latency time.

Our system implementation is based on off-the-shelf hardware and software components. The clients and servers of the system are connected through a switched 100Mbps Ethernet that does not represent the bottleneck for our experiments. The Web switch is implemented on a PC PentiumII-450Mhz with 256MB of memory. We use four PC Pentium MMX 233Mhz with 128MB of memory as Web servers. All nodes of the cluster use a 3Com 3C905B 100bTX network interface. They are equipped with a Linux operating system (kernel release 2.2.16), and Apache 1.3.12 as the Web server software. On the Web switch node, the Apache Web server is configured as a reverse proxy through the modules *mod_proxy* and *mod_rewrite* [3]. The dispatching algorithms are implemented as C modules that are activated once at startup of the Apache servers. The dispatching module communicates with the *rewriting engine*, which is provided by the *mod_rewrite*, over its *stdin* and *stdout* file handles. For each map-function lookup, the dispatching module receives the key to lookup as a string on *stdin*. After that, it has to return the looked-up value as a string on *stdout*.

Because the LARD algorithm requires also information about the server load, we implement on each Web server a *Collector* C module that collects the number of active HTTP connections. Every 10 seconds the *Manager* C module installed on the Web switch gathers server load information through a socket communication mechanism.

5.2 Experimental results

To test the system we use a pool of client nodes that are interconnected to the Web cluster through a dedicated Fast Ethernet. The clients runs on four PentiumII PCs. As synthetic workload generator we use a modified version of the Webstone benchmark [27] (version 2.1). The main modifications concern the client behavior for which we introduce the concept of user think-time and embedded objects per Web page as described in Section 4.2. Moreover, the file size of each object is modeled through a Pareto distribution. To emulate a dynamic workload, we create three CGI services: two of them stress the CPU of the Web server nodes in a lightly or intensive way; the third service stresses both the CPU and disk. In the experiments we compare the performance of dispatching strategies under three different scenarios: static workload, light dynamic workload, and intensive dynamic workload. In the dynamic scenario, 80% of requests are for static objects and 20% are for dynamic services. The dynamic requests are classified as CPU-light (10%), CPU-intensive (6%) and CPU-Disk-intensive (4%).

The main performance metric is the Web cluster throughput measured in connections per second (conn/sec). We prefer this measure because it represents system performance better than Mbps especially for highly heterogeneous workloads.

In the first set of experiments we compare the performance of CAP and LARD under static and light dynamic workload. Figure 7 shows the system throughput in the case of static scenario. We can see that LARD and CAP perform similarly. When we pass to consider a light dynamic workload (Figure 8), the performance results of CAP and LARD change completely because dynamic requests stress the server resources in a quite different way. Figure 9 presents the system throughput for an intensive dynamic workload that confirms that CAP outperforms LARD performance. Compared to this latter algorithm, CAP throughput increases from 13% when 160 clients are in the system, to 21% when 240 clients are connected. All the above results let us think that if the complexity and variety of client requests augment, the improvement of CAP can increase as well. The main motivation for this result is the better load balancing achieved by CAP with respect to LARD. To this purpose, we show in Figure 10 the minimum and maximum server utilizations in a cluster where the Web switch uses a LARD and CAP policy, respectively.

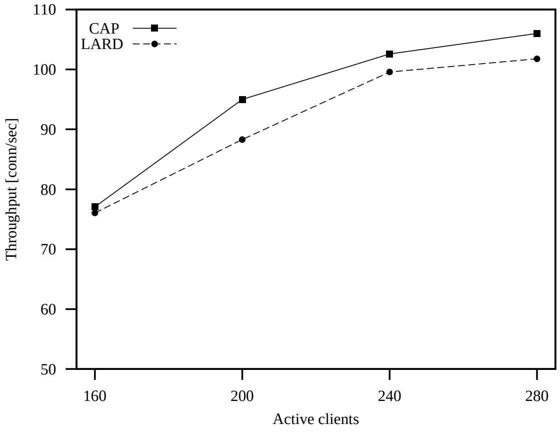


Figure 7: Static workload: system throughput.

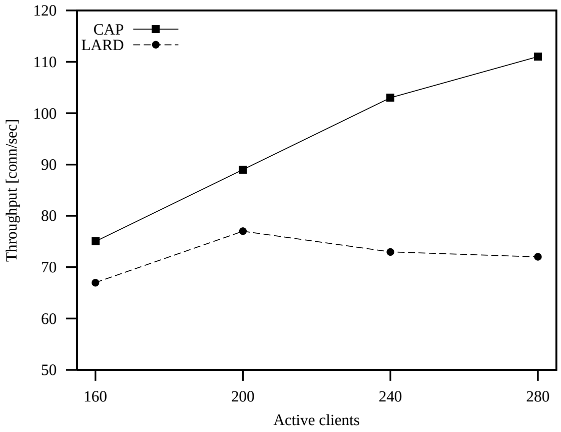


Figure 8: *Light* dynamic workload: system throughput

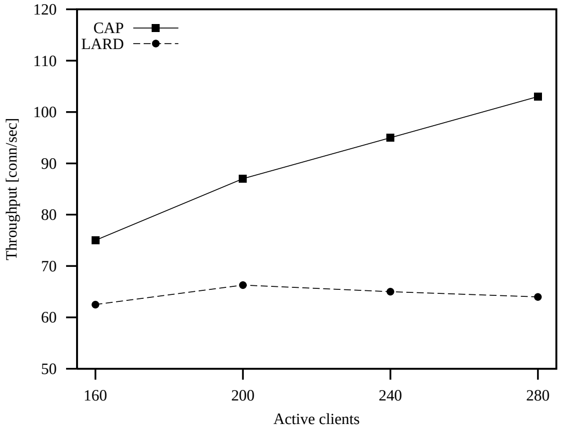


Figure 9: *Intensive* dynamic workload: system throughput

6. CONCLUSIONS

Web cluster architectures are becoming very popular for supporting Web sites with large numbers of accesses and/or heterogeneous services. In this paper, we propose a new scheduling policy, called *client-aware policy* (CAP), for Web switches operating at layer-7 of the OSI protocol stack to route requests reaching the Web cluster. CAP classifies the client requests on the basis of their expected impact on main server components. At run-time, CAP schedules client requests reaching the Web cluster with the goal of sharing

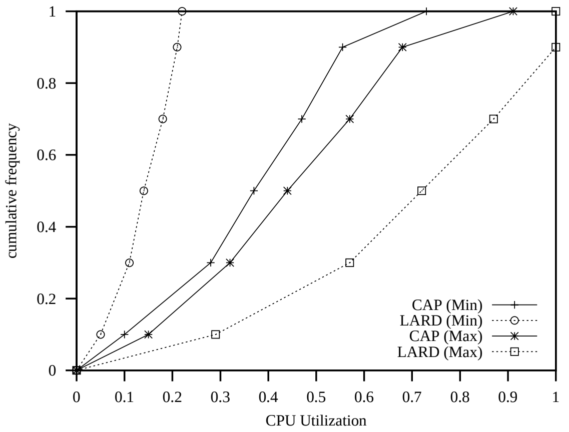


Figure 10: *Light* dynamic workload: distributions of min-max server utilizations in the Web cluster.

all classes of services among the servers, so that no system resource tends to be overloaded. We demonstrate through simulation and experiments on a real prototype that dispatching policies that improve Web caches hit rates give best results for traditional Web publishing sites providing most static information and some lightly dynamic requests. On the other hand, CAP provides scalable performance even for modern Web clusters providing static, dynamic and secure services. Moreover, the pure CAP has the additional benefit of guaranteeing robust results for very different classes of Web services because it does not require a hard tuning of system parameters as many other server-aware dispatching policies do.

Acknowledgments

The authors would like to thank Mauro Andreolini from the University of Rome for his support in prototype experiments. This work was carried out under the partial support of Banca di Roma (Res. contract BDR-2001 on "Advanced technologies").

7. REFERENCES

[1] Alteon WebSystems, Alteon 780 Series, in www.alteonwebsystems.com/products/

[2] E. Anderson, D. Patterson, E. Brewer, "The Magicrouter, an application of fast packet interposing", unpublished Tech. Rep., Computer Science Department, University of Berkeley, May 1996.

[3] Apache docs., in www.apache.org/docs/mod/

[4] M.F. Arlitt, C.L. Williamson, "Internet Web servers: Workload characterization and performance implications", *IEEE/ACM Trans. on Networking*, vol. 5, no. 5, Oct. 1997, pp. 631-645.

[5] M. Aron, P. Druschel, W. Zwaenepoel, "Efficient support for P-HTTP in cluster-based Web servers", *Proc. USENIX 1999*, Monterey, CA, June 1999.

[6] M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, "Scalable content-aware request distribution in cluster-based network servers", *Proc. USENIX 2000*, San Diego, CA, June 2000.

[7] L. Aversa, A. Bestavros, "Load balancing a cluster of Web servers using Distributed Packet Rewriting", *Proc. of IEEE IPCCC'2000*, Phoenix, AZ, February 2000.

[8] P. Barford, A. Bestavros, A. Bradley, M.E. Crovella, "Changes in Web client access patterns: Characteristics and caching implications", *World Wide Web*, Jan. 1999.

[9] P. Barford, M.E. Crovella, "A performance evaluation of Hyper Text Transfer Protocols", *Proc. of ACM Sigmetrics '99*, Atlanta, Georgia, May 1999, pp. 188-197.

[10] V. Cardellini, M. Colajanni, P.S. Yu, "Dynamic load balancing on scalable Web server systems", Proc. of *MASCOTS'2000*, San Francisco, Aug. 2000.

[11] T.L. Casavant, J.G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems", *IEEE Trans. on Software Engineering*, vol. 14, no. 2, Feb. 1988, pp. 141-154.

[12] Cisco's LocalDirector, in www.cisco.com

[13] A. Cohen, S. Rangarajan, H. Slye, "On the performance of TCP splicing for URL-aware redirection", *Proc. of 2nd USENIX Symposium on Internet Technologies and System*, Boulder, CO, Oct. 1999.

[14] M. Colajanni, P.S. Yu, D. Dias, "Redirection algorithms for load sharing in distributed Web server systems", *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, no. 6, pp. 585-600, June 1998.

[15] M.E. Crovella, A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and possible causes", *IEEE/ACM Trans. on Networking*, vol. 5, no. 6, Dec. 1997, pp. 835-846.

[16] O.P. Damani, P.E. Chung, Y. Huang, C. Kintala, Y.-M. Wang, "ONE-IP: Techniques for hosting a service on a cluster of machines", *Proc. of 6th Intl. World Wide Web Conf.*, Santa Clara, CA, Apr. 1997.

[17] D.M. Dias, W. Kish, R. Mukherjee, R. Tewari, "A scalable and highly available Web server", *Proc. of 41st IEEE Comp. Society Int. Conf.*, Feb. 1996.

[18] R.S. Engelschall, "Load balancing your Web site", *Web Techniques Magazine*, Vol. 3, May 1998.

[19] F5 Networks Inc. BigIP Version 3.0, in www.f5labs.com

[20] G.D.H. Hunt, G.S. Goldszmidt, R.P. King, R. Mukherjee, "Network Web switch: A connection router for scalable Internet services", Proc. of *7th Int. World Wide Web Conf.*, Brisbane, Australia, April 1998.

[21] A. Iyengar, J. Challenger, D. Dias, P. Dantzig, "High-Performance Web Site Design Techniques", *IEEE Internet Computing*, vol. 4 no. 2, March/April 2000.

[22] B.A. Mah, "An empirical model of HTTP network traffic", *Proc. of IEEE Int. Conf. on Computer Communication*, Kobe, Japan, April 1997.

[23] R. Mukherjee, "A scalable and highly available clustered Web server", in *High Performance Cluster Computing: Architectures and Systems, Volume 1*, Rajkumar Buyya (ed.), Prentice Hall, 1999.

[24] V.S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, "Locality-aware request distribution in cluster-based network servers", *In Proc. of 8th ACM Conf. on Arch. Support for Progr. Languages*, San Jose, CA, Oct. 1998.

[25] T. Schroeder, S. Goddard, B. Ramamurthy, "Scalable Web server clustering technologies", *IEEE Network*, May-June 2000, pp. 38-45.

[26] J. Song, E. Levy-Abegnoli, A. Iyengar, D. Dias, "Design alternatives for scalable Web server accelerators", *Proc. 2000 IEEE Int. Symp. on Perf. Analysis of Systems and Software*, Austin, TX, Apr. 2000.

[27] Webstone Mindcraft Inc., *Webstone2.01*, in www.mindcraft.com/webstone/ws201index.html

[28] C.-S. Yang, M.-Y. Luo, "A content placement and management system for distributed Web-server systems", *Proc. of IEEE 20th Int. Conf. on Distributed Computing Systems*, Taipei, Taiwan, Apr. 2000.

[29] X. Zhang, M. Barrientos, J.B. Chen, M. Seltzer, "HACC: An architecture for cluster-based Web servers", *Proc. 3rd USENIX Windows NT Symp.*, Seattle, July 1999.