RC22209 (W0110-048) October 16, 2001
Computer Science

IBM Research Report

The State of the Art in Locally Distributed Web-server Systems

Valeria Cardellini, Emiliano Casalicchio
Dept. of Computer Engineering
University of Roma Tor Vergata
Roma, Italy 00133

Michele Colajanni
Dept. of Information Engineering
University of Modena
Modena, Italy 41100

Philip S. Yu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

The State of the Art in Locally Distributed Web-server Systems

Valeria Cardellini, Emiliano Casalicchio
Dept. of Computer Engineering
University of Roma Tor Vergata
Roma, Italy 00133
{*cardellini, ecasalicchio*}*@ing.uniroma2.it*

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

Philip S. Yu
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
psyu@us.ibm.com

Abstract

The overall increase in traffic on the World Wide Web causes a lengthening of client requests to popular Web sites, especially in conjunction with special events. Single node platforms that do not replicate information content cannot provide the needed scalability to handle large traffic volumes and to match rapid and dramatic changes in the number of clients. The need to improve the performance of Web-based services has produced a variety of novel content delivery architectures. This paper will focus on locally distributed Web systems, where the server nodes reside at a single location. After years of proposals of new routing mechanisms, policies and system solutions (the first dated back to 1994 when the NCSA Web site had to face the first million of requests per day) many problems concerning multiple server architectures for Web sites have been solved. Other issues remain to be addressed especially at the network application level, but the principle techniques and methodologies for building scalable Web content delivery architectures placed in a single location are settled now. This paper classifies and describes main mechanisms to split the traffic load among the server nodes, discussing both the alternative architectures and the load sharing policies. To this purpose, it focuses on architectures, internal routing mechanisms, and dispatching request algorithms for designing and developing scalable Web-server systems and identifies some of the open research issues associated with the use of distributed systems for highly accessed Web sites.

Corresponding author:
Philip S. Yu
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
Phone: (914) 784 7141
Fax: (914) 784 7455
E-mail: psyu@us.ibm.com

Contents

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

Additional Key Words and Phrases: World Wide Web, Client/server, Distributed systems, Load balancing, Dispatching algorithms, Routing mechanisms, Cluster-based architectures

1 Introduction

Demands placed on Web services continue to grow and Web server systems are becoming more stressed than ever. Because of the complexity of the Web infrastructure, performance problems may arise in many points during a Web transaction. For instance, they may occur at network level because of congested Internet routers, as well as at server level either because of under-provisioned capacity or unexpected surge of requests. Although both network and server capacity have improved in recent years, and new architectural solutions have been deployed, response time continues to challenge Web system related research. In this paper, we focus on architecture solutions that aim to reduce the delays due to the Web server site assuming that network issues are examined elsewhere. Increasing demand and immediate applicability of the considered methodologies are the main motivation and guideline of this survey, respectively.

In a completely distributed control system such as Internet, the Web site is the only component that can be under the direct control of the content provider administration. Web clients, Internet backbones and routers, DNS system, and proxy servers are not controllable by a single organization, and any intervention on these components is hard to be applicable because it requires an agreement among multiple organizations.

Another motivation for focusing on Web system architecture is due to the growing complexity of Web applications and services. Web performance perceived by end users is increasingly dominated by server delays, especially when contacting busy servers [15]. Recent measures suggest that the Web servers contribute for about 40% of the delay in a Web transaction [58] and it is likely that this percentage will increase in the near future. As a consequence of the changes transforming the Web from a simple communication and browsing infrastructure to a complex medium for conducting personal businesses and e-commerce, the computational load placed by network applications on Web servers will continue to grow. A prediction made in 1995 regarding network bandwidth estimated that it would triple every year for the next 25 years. So far, this prediction seems to be approximately correct [55], while the Moore law estimates "just" a doubling of system capacity every 18 months. Other network improvements, such as private peering agreements between backbone providers, the deployment of Gigabit wide-area networks, the rapid adoption of ISDN networks, xDSL lines, and cable modems contribute to reduce network latency. With the network bandwidth increasing about twice faster than the server capacity, and increased complexity of Web-based applications, the future bottleneck is likely to be more on the server side.

1.1 Scalable Web-server systems

Web site administrators constantly face the need to increase server capacity. In this paper, Web system scalability is defined as the ability to support large numbers of accesses and resources while still providing adequate performance.

The first option used to scale Web services is to upgrade the Web server to a larger, faster machine. This strategy, referred to as *hardware scale-up* [43], simply consists in expanding a system

by incrementally adding more resources (e.g., CPUs, disks, network interfaces) to an existing node. While hardware scale-up relieves short-term pressure, it is neither a cost-effective nor a long-term solution, considering the steep growth in the client demand curve which characterizes the Web (the number of online users is growing at about 90% per annum).

Many efforts have also been directed at improving the performance of a Web server node at the software level, namely *software scale-up*. This includes the operating system and also the Web server application [14, 12, 57, 75, 80]. The Flash Web server ensures that its threads and processes are never blocked by using an asymmetric multiprocess event-driven architecture [80]. Nahum et al. [75] have analyzed how a general-purpose operating system and the network protocol stack can be improved to provide support for high-performing Web servers. Hu et al. [57] have proposed some techniques to improve the performance of the Apache Web server.

However, the approach of improving the power of a single server will not solve the Web scalability problem in a foreseeable future. A more appealing solution to keep up with ever increasing request load and provide scalable Web services is to deploy a distributed Web system architecture composed by multiple server nodes where some system component under the control of the content provider can route incoming requests among different servers. Load sharing is instrumental in obtaining high performance server systems. Hence, the load reaching the Web site must be evenly distributed among the server nodes belonging to the system, so as to reduce user-perceived latency time and to achieve the highest performance.

The approach in which the system capabilities are expanded by adding more nodes, complete with processors, storage, and bandwidths, is typically referred to as *scale-out* [43]. We further distinguish between *local scale-out* when the set of server nodes resides at a single network location, and *global scale-out* when the nodes are located at different geographical locations. Figure 1 summarizes the different approaches to achieve system scalability.

This survey presents and discusses the various approaches for managing *locally distributed Web systems* (the focused topics are written in bold in Figure 1). We describe a series of architectures, routing mechanisms, and dispatching algorithms to design local Web-server systems and identify some of the issues associated with setting up and managing such systems for highly accessed Web sites. We examine how locally distributed architectures and related management algorithms satisfy the scalability and performance requirements of Web services. We also analyze the efficiency and the limitations of the different solutions and the tradeoff among the alternatives with the aim of identifying the characteristics of each approach and their impact on performance.

1.2 Basic architecture and operations

A scalable Web-server system needs to appear as a single host to the outside world, so that users need not be concerned about the names or locations of the replicated servers and they can interact with the Web-server system as if it were a single high performance server. Hence, the basic model must adhere to the architecture transparency requirement by providing a single virtual interface to the outside world at least at the site name level. (Throughout this survey, we will use `www.site.org`

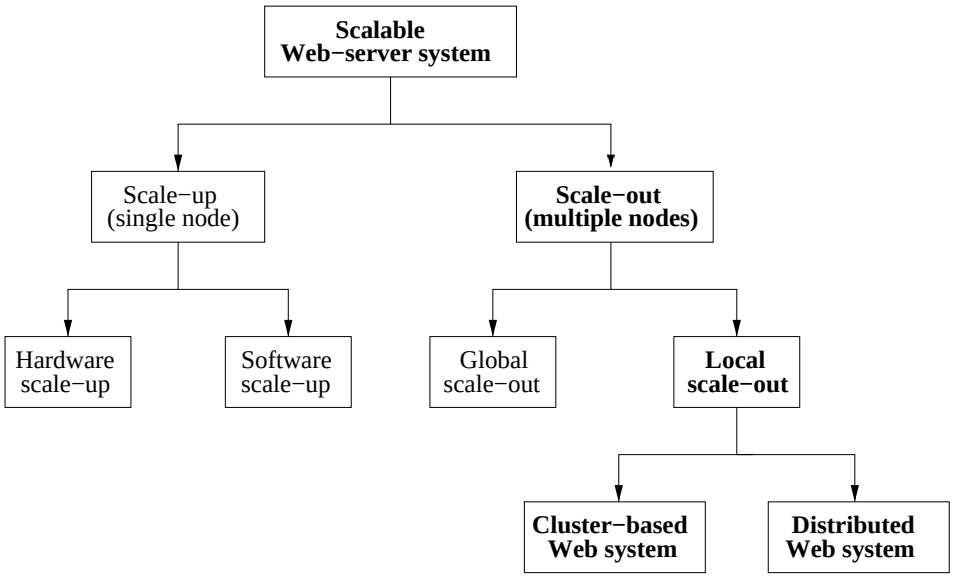

Figure 1: Architecture solutions for scalable Web-server systems.

as example name for the Web site.) This choice excludes from our analysis some architectures such as the *mirrored-server* system, a simple solution that lets users manually select alternative names for a Web site, thereby violating the transparency requirement. Unlike the users, the client applications might be aware of the effects of some dispatching mechanisms. However, in our survey the clients do not need any modification to interact with the scalable Web system. Hereafter, we refer either to *client* (or *Web browser*) as to the software entity that acts as a user agent and is responsible for implementing all the interactions with the Web server, including generating the requests, transmitting them to the server, receiving the results from the server and presenting them to the user.

From these premises, the basic Web system architecture consists of multiple server nodes, distributed on a local area with one or more mechanisms to spread client requests among the nodes. Each Web server can access all site information, independently of the degree of content replication. The Web system includes also one authoritative Domain Name System (DNS) server for translating the Web site name into one or more IP address(es) and, if necessary, one or more internal routing devices. A high-level view of the basic architecture is shown in Figure 2. A router and other network components belonging to the Web system could exist in the way between the system and the Internet. Moreover, it has to be noted that an architecture for modern Web sites consists also of back-end nodes that typically act as data servers for dynamically generated information. The main focus of this survey is on the Web server layer while the techniques concerning content distribution at the back-end layer are outlined in Section 8.

We analyze now the main phases to serve a user request to a Web site. Although a user issues one request at a time for a *Web page*, he causes multiple client-server interactions because typically

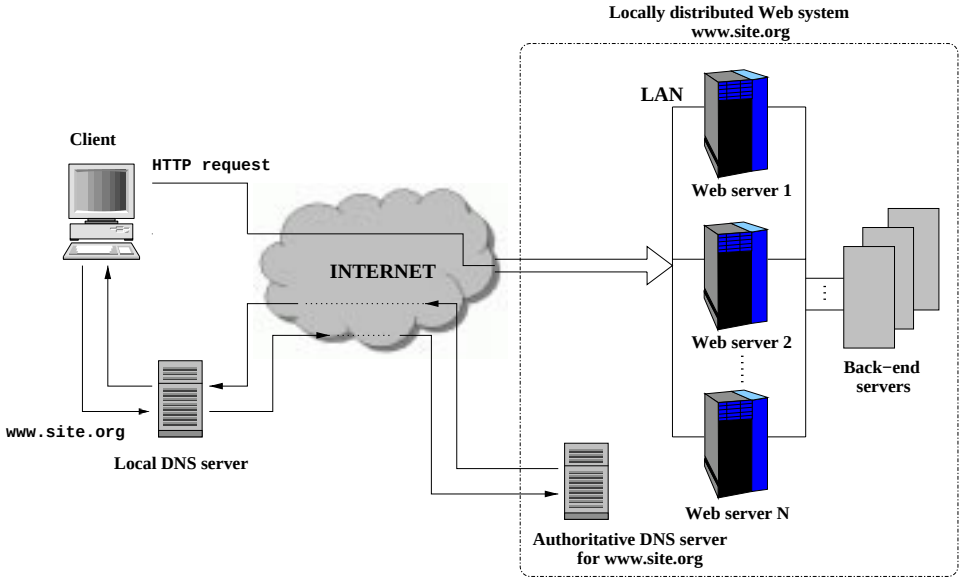

Figure 2: Model architecture for a locally distributed Web system.

a *Web page* is a multi-part document consisting of a collection of objects. A static *Web object* is a file in a specific format (e.g., an HTML file, a JPEG image, a Java applet, an audio clip), which is addressable by a single URL (e.g., `http://www.site.org/pub/index.html`). A URL has two main components: the symbolic name of the server that houses the object (e.g., `www.site.org`) and the object's path name (e.g., `/pub/index.html`). To retrieve all the Web objects composing a Web page, a browser issues multiple requests to the Web server. We refer to a *Web transaction* as the complete interaction that starts when the user sends a request to a Web site and ends when his client receives the last object related to the requested URL. A *session* is a sequence of Web transactions issued by the same user during an entire visit to a Web site.

Summing up, a Web transaction starts when the user makes a request to a Web server, by either typing an URL or clicking on a link, for example `http://www.site.org/pub/index.html`. First, the client extracts the symbolic site name (`www.site.org`) from the requested URL and asks its local domain name server to find out the IP address corresponding to that name. The local name server obtains the IP address by eventually contacting a well-known root of the domain name server hierarchy and ultimately querying the `site`'s authoritative name server. The local name server returns the obtained IP address to the client that establishes a TCP connection with the server or device corresponding to that IP address. After that, the client can send the HTTP request for `/pub/index.html` to the Web server that sends the requested object back. Most Web pages consist of a base HTML file describing the page layout and a number of objects referenced by the HTML file; the page is intended to be rendered to the user as a single unit. Once obtained the base HTML page from the Web server, the client parses it. If the client finds that embedded objects are related to the base page, it sends a separate request for each object. Depending on the

HTTP protocol used for the client/server interactions, the subsequent requests can use the same TCP connection (protocol HTTP/1.1) or different TCP connections (protocol HTTP/1.0).

1.3 Request routing mechanisms and scope of this paper

We have seen in the previous subsection that each client request for a Web page involves several operations even when the Web site is hosted on a single server. Moreover, the basic sequence for a Web transaction seen in the previous subsection can be altered by several factors, such as caching of the IP address corresponding to the site name at the client browser or at some intermediate name servers, the version of HTTP protocol being used in the client/server interaction (i.e., support or not for persistent TCP connections), the presence of the requested Web object either in the client local cache or in intermediate proxy servers located on the path between the client and the Web server.

When the Web site is hosted on multiple servers, another alternative is to decide which server of the distributed Web system has to serve a client request. This decision might occur in several places along the request path from the client to the Web site. We identify four possible levels for deciding how to route a client request to one server of the locally distributed Web-server system. All the cited components are shown in Figure 2.

- At the *Web client* level, the entity responsible for the request assignment can be any client that originates a request.
- At the *DNS* level, during the address resolution phase, the entity in charge of the request routing is primarily the authoritative DNS server for the Web site.
- At the *network* level, the client request can be directed by router devices and through multicast/anycast protocols.
- At the *Web system* level, the entity in charge for the request assignment can be any Web server or other dispatching device(s) typically placed in front of the Web site architecture.

Only a subset of the presented alternatives are considered in this paper. Indeed, the basic premise of this survey is the compatibility of all proposed solutions with existing Web standards and protocols so that any considered architecture, algorithm and mechanism could be immediately adopted without any limiting assumption and without requiring modifications to existing Internet protocols, Web standards, and client code. Therefore, we will focus on dispatching solutions that occur at system components that are under the direct control of the content provider management, i.e., the authoritative DNS, the Web servers, and some internal devices in the Web system. On the other hand, we do not consider client-based routing mechanisms [11, 74, 94, 99] because they may require some modifications of the client software [24]. Also, we do not investigate dispatching at the network level that is meaningful when the multiple nodes of the Web site architecture are distributed over a geographical area.

It is worth observing that the design and implementation of a Web site architecture consisting of multiple servers is not the only way for improving response time as perceived by the user. Basically, there are two important categories for which we provide just some references for additional reading. They are *external caching* and *outsourcing* solutions.

Probably, the most popular approach for reducing latency time and Web traffic is based on caching of the Web objects that might occur at different levels. An accurate examination of caching solutions would require a dedicated survey because they are the first techniques proposed in literature and can be deployed at different scales: from server disk caching to proxy caching to browser caching with all possible combinations [97]. In this paper, we consider some *internal caching* techniques occurring inside the Web-server system, such as Web server accelerators [29, 91] and other techniques for improving cache hit rate [8, 79], while we exclude external caching solutions, such as proxy servers and cooperative proxies [16, 95], virtual servers (or reverse proxies) [71], Web proxy accelerators [87].

In this survey, we also exclude solutions where the content provider delegates scalability for its Web services to other organizations. For example, many Web sites contract with third-party Web hosting and co-location providers. Interesting research and implementation issues come from Web-server systems that store and provide access to multiple Web sites [3, 7, 32, 70]. More recently, *Content Delivery Network* (CDN) organizations undertake to serve request traffic for Web sites from caching sites at various Internet borders [1, 2, 53].

1.4 Organization of this paper

The rest of this paper is organized as following.

- Section 2 discusses and classifies locally distributed architectures for Web sites, by distinguishing *cluster-based Web systems* or simply *Web clusters*, and *distributed Web systems*.
- Section 3 and Section 4 cover the mechanisms that can be used to route requests in a Web cluster and in a distributed Web system, respectively.
- Section 5 presents the policies for load sharing and dispatching requests in the class of Web cluster systems. We present a taxonomy of the policies that have been developed in recent years by focusing on the issues that each policy addresses.
- Section 6 classifies various academic prototypes and commercial products according to the routing mechanism that is used to distribute the client requests among the servers.
- Section 7 presents some extensions of the basic system architecture to improve scalability.
- Section 8 deals with the problem of Web content placement among multiple front-end and back-end servers.
- Section 9 concludes the paper and presents some open issues for future research.

2 A taxonomy of locally distributed architectures

The basic premise of the proposed taxonomy is the compatibility of all analyzed solutions with existing Web standards and protocols so that any considered architecture, algorithm and mechanism could be immediately adopted without any limiting assumption.

Distributed architectures can be differentiated depending on the name virtualization being extended at IP level or not. Given a set of server nodes that host a Web site at a single location, we can identify two main classes of architectures:

Cluster-based Web System (or **Web clusters**) where the server nodes mask their IP addresses to clients. The only client-visible address is a *Virtual IP* (VIP) address corresponding to one device which is located in front of the set of servers.

Distributed Web System where the IP addresses of the Web server nodes are visible to client applications.

Making visible or masking server IP addresses is a key feature, because each solution implies quite different mechanisms and algorithms for distributing the client requests among the server nodes. In particular, the distributed Web system architecture is the oldest solution, where the request routing is decided by the DNS system with the possible integration of some other naming entity. The cluster-based Web system architecture is a more recent solution where request routing is entirely carried out by the internal components of the Web cluster. We can anticipate that the cluster systems are preferable to a locally distributed Web site because they can provide fine-grained control on request assignment and better availability and security. In this survey, we give more attention to cluster-based Web systems than on distributed Web systems because we consider that a "visible" architecture is more suitable to geographically dispersed servers.

The main components of a typical multi-node Web system include a *request routing mechanism* to direct the client request to a target server node, a *dispatching algorithm* to select the Web server node that is considered best suited to respond, and an *executor* to carry out the dispatching algorithm and support the routing mechanism. In this survey we will distinguish routing mechanisms from dispatching policies for the two main classes of locally distributed Web system architectures.

2.1 Cluster-based Web systems

A cluster-based Web system (briefly, *Web cluster*) refers to a collection of server machines that are housed together in a single location, are interconnected through a high-speed network, and present a single system image to the outside. Each server node of the cluster usually contains its own disk and a complete operating system. Cluster nodes work collectively as a single computing resource. Massive parallel processing systems (e.g., SP-2) where each node satisfies all previous characteristics can be assimilated to a cluster-based Web system. In literature some alternative terminology is used to refer to a Web cluster architecture. One common term is also *Web farm*, meaning the collection of all the servers, applications, and data at a particular site [43]. We prefer

the term Web cluster, as Web farm is typically used to denote an architecture for Web site hosting or co-location.

Although a Web cluster may consist of tens of nodes, it is publicized with one site name (e.g., `www.site.org`) and one virtual IP (VIP) address (e.g., 144.55.62.18). Thus, the authoritative DNS server for the Web site always performs a one-to-one mapping by translating the site name into the VIP address, which corresponds to the IP address of a dedicated front-end node(s). It interfaces the rest of the Web cluster nodes with the Internet, thus making the distributed nature of the site architecture completely transparent to both the user and the client application. The front-end node, hereafter called *Web switch*, receives all inbound packets that clients send to the VIP address, and routes them to some Web server node. In such a way, it acts as the centralized dispatcher of a distributed system with fine-grained control on client requests assignments.

A high-level view of a basic Web cluster comprising the Web switch and N servers is shown in Figure 3. It is to be noted that the response line does not appear here because the two main alternatives will be described in Section 3.

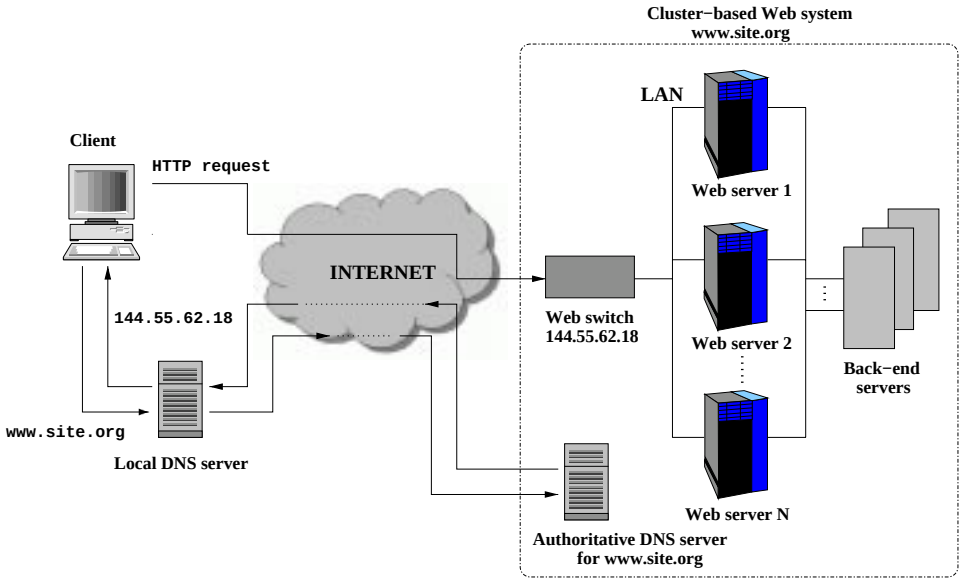

Figure 3: Architecture of a *cluster-based Web system.*

The Web switch can be implemented on either special-purpose hardware devices plugged into the network or software modules running on a special-purpose or general-purpose operating system. In this paper, we use the term Web switch to refer to the dispatching entity in general. This definition does not imply that the Web switch is a hardware device that forwards frames based on link layer addresses, or packets based on layer-3 and layer-4 information. Moreover, we prefer not to call the Web switch through the functionality it implements, for example *network/server load balancer*, as it can be found in some literature.

2.2 Distributed Web systems

A distributed Web system consists of locally distributed server nodes, whose multiple IP addresses may be visible to client applications. A high-level view of a locally distributed architecture is shown in Figure 4. Unlike the cluster-based Web system, this architecture does not have a front-end Web switch, so the client request assignment to a target Web server is typically carried out during the address resolution of the Web site name (*lookup phase*) by the DNS mechanism. In some systems, there is also a *second-level routing* which is typically carried out through some re-routing mechanism activated by a Web server that cannot fulfill a received request. The dispatching mechanisms in a distributed Web system will be examined in Section 4.

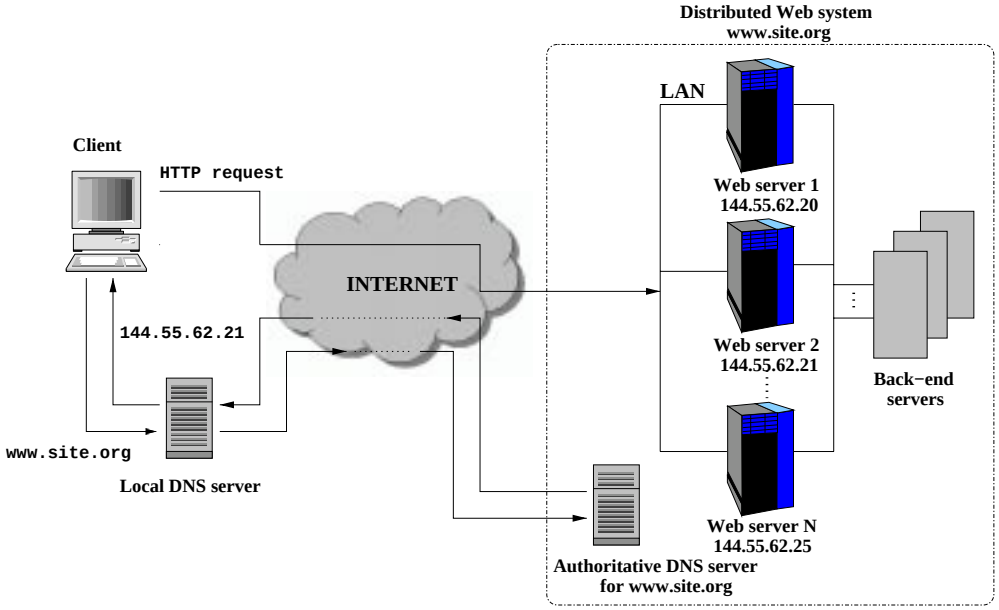

Figure 4: Architecture of a *distributed Web system*.

3 Request routing mechanisms for cluster-based Web systems

The Web switch is able to identify uniquely each node in the system through a private address that can be at different protocol levels, depending on the architecture. More specifically, the server private address may correspond to either an IP address or a lower-layer (MAC) address. There are various techniques to deploy Web clusters, however the key role is always played by the Web switch. For that reason, we first classify the Web cluster architecture alternatives according to the OSI protocol stack layer at which the Web switch routes inbound packets to the target server, that is *layer-4* or *layer-7* Web switches. The choice of the routing mechanism has also a big impact on dispatching policies because the kind of information available at the Web switch is quite different.

- Layer-4 Web switches perform *content-blind routing* (also referred to as *immediate binding*), because they determine the target server when the client asks for establishing a TCP/IP connection, upon the arrival of the first TCP SYN packet at the Web switch. As the client packets do not reach the application level, the routing mechanism is efficient but the dispatching policies are unaware of the content of the client request.

- Layer-7 Web switches can execute *content-aware routing* (also referred to as *delayed binding*). The switch first establishes a complete TCP connection with the client, examines the HTTP request at application level and then relays it to the target server. This routing mechanism is much less efficient, but it can support more sophisticated dispatching policies. (We refer to layer-7 Web switches according to the ISO/OSI protocol layers, where the application layer is the seventh. Other authors refer to switches that perform content-aware routing as *layer-5* or *application-layer* switches.)

Web cluster architectures based on layer-4 and layer-7 Web switches can be further classified on the basis of the data flow between the client and the target server, the main difference being in the return way of server-to-client. Indeed, all client requests necessarily have to flow through the Web switch. On the other hand, the target server can either respond directly to the client (namely, *one-way* architectures) or return its response to the Web switch, that in its turn sends the response back to the client (referred to as *two-way* architectures). Figure 5 summarizes the taxonomy for Web clusters that we have examined so far. Typically, *one-way* architectures are more complex and more efficient because the Web switch processes only inbound packets, while the opposite is true for two-way architectures because the Web switch has to process both inbound and outbound packets.

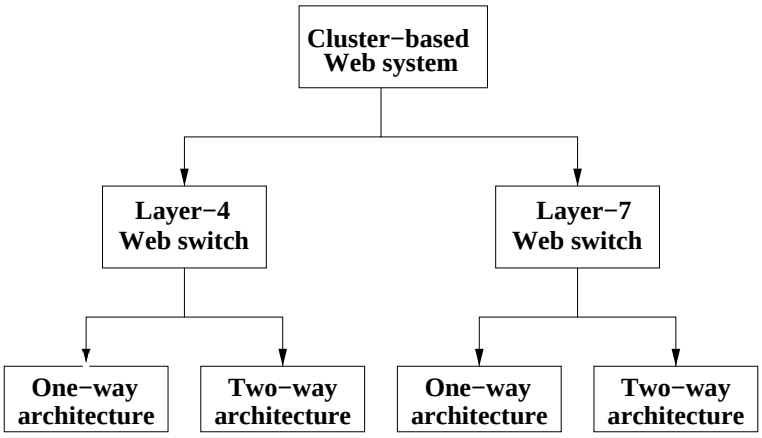

Figure 5: Taxonomy of cluster-based architectures.

3.1 Solutions based on layer-4 switches

Layer-4 Web switches work at TCP/IP level. Since packets pertaining to the same TCP connection must be assigned to the same Web server node, the client assignment is managed at TCP session level. The Web switch maintains a *binding table* to associate each client TCP session with the target server.

Upon receiving an inbound packet, the Web switch examines its header and determines, on the basis of the bits in the flag field, whether the packet pertains to a new connection, a currently established one, or none of them. If the inbound packet is for a new connection (i.e., the SYN flag bit is set), the Web switch selects a target server through the dispatching policy, records the connection-to-server mapping in an entry of the binding table, and routes the packet to the selected server. If the inbound packet is not for a new connection, the Web switch looks up the binding table to verify whether the packet belongs or not to an existing connection. If it does, the Web switch routes the packet to the server that is in charge for that connection. Otherwise, the Web switch drops the packet.

To improve Web switch performance, the binding table is typically kept in memory and accessed through a hash function. Each entry contains the tuple `<IP source address, source port, IP destination address, destination port>`, and other information (e.g., beginning time) that may be relevant for some dispatching algorithm.

Layer-4 Web clusters can be classified on the basis of the mechanism used to route inbound packets to the target server and outbound packets to the client. The main difference is in the return way, i.e., server-to-client. In *two-way* architectures both inbound and outbound packets pass through the Web switch, while in *one-way* architectures only inbound packets flow through the Web switch.

3.1.1 Two-way architectures

In *two-way* architectures, each server in the cluster is configured with a unique IP address, i.e., the private address is at IP level. Both inbound and outbound packets are rewritten at TCP/IP level by the Web switch, as shown in Figure 6.

Packet rewriting is based on the IP Network Address Translation approach [46]. The Web switch rewrites inbound packets by changing the VIP address to the IP address of the target server in the destination address field of the packet header. Outbound packets from the servers to clients must also pass back through the switch. As the source address in the outbound packets is the address of the server that has served the request, the Web switch needs to rewrite the server IP address with the VIP address, so as not to confuse the client. Furthermore, the Web switch has to recalculate the IP and TCP header checksums for both packet flows.

3.1.2 One-way architectures

In *one-way* architectures inbound packets pass through the Web switch, while outbound packets flow directly from the servers. This requires a separate high-bandwidth network connection for

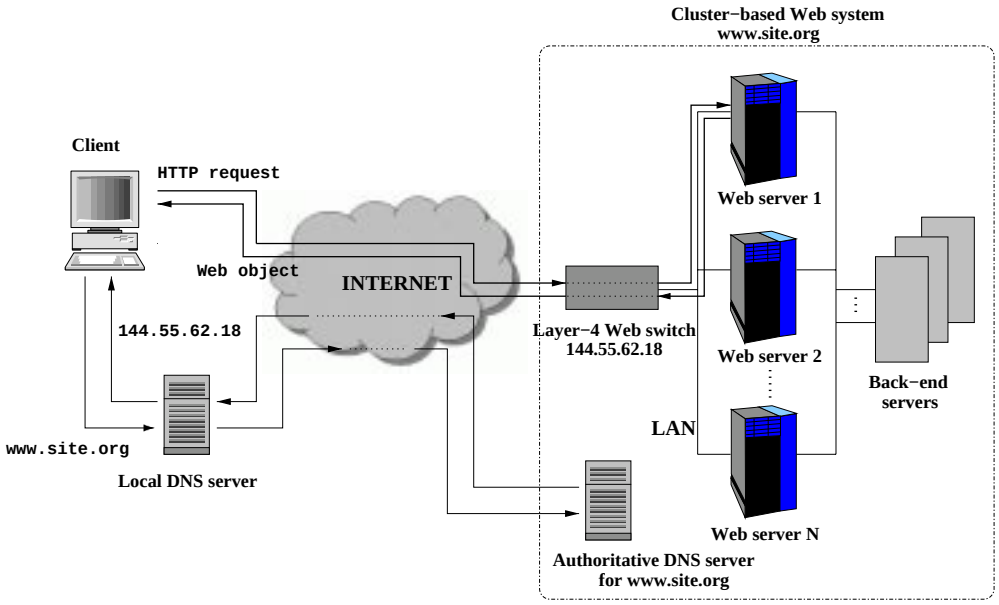

Figure 6: Layer-4 two-way architecture.

outbound packets. Figure 7 shows an example where the outbound packets flow back to the client from the Web server 1. In this figure, the level of the server private address is intentionally not specified as it can be either at IP level (layer-3) or MAC level (layer-2).

Routing to the target server can be done by means of several mechanisms, such as rewriting the IP destination address and recalculating the TCP/IP checksum of the inbound packet (*packet rewriting*), encapsulating each packet within another packet (*packet tunneling*), forwarding the packet at MAC level (*packet forwarding*). Let us describe how each mechanism works.

Packet single-rewriting. The routing to the target server is achieved by rewriting the destination IP address of each inbound packet: the Web switch replaces its VIP address with the IP address of the selected server and recalculates the IP and TCP header checksum. Thus, the private addresses of the server nodes are at IP level.

The difference from two-way architectures is in the modification of the source address of outbound packets. The Web server, before sending the response packets to the client, replaces its IP address with the VIP address and recalculates the IP and TCP header checksum [44]. To differentiate packet rewriting operations, we call *double-rewriting* those performed by two-way architectures and *single-rewriting* those carried out by one-way architectures.

Packet tunneling. IP tunneling (or IP encapsulation) is a technique to encapsulate IP datagrams within IP datagrams, thus allowing datagrams destined to one IP address to be wrapped and redirected to another IP address [82]. The effect of IP tunneling is to transform the old headers and data into the payload of the new packet. The Web switch tunnels the inbound packet to the target server by encapsulating it within an IP datagram. The header of this

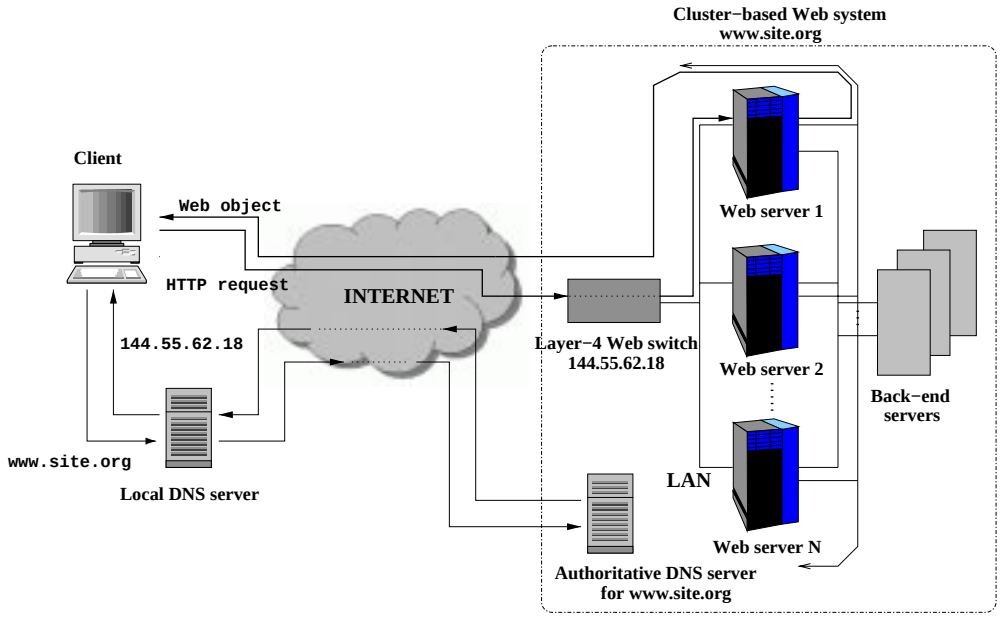

Figure 7: Layer-4 one-way architecture.

datagram contains the VIP address and the server IP address as source and destination address, respectively. The server private addresses are at IP level as in packet rewriting.
This mechanism requires that all servers support IP tunneling and have one of their tunnel devices configured with the VIP address. When the target server receives the encapsulated packet, it strips the IP header off and finds that the inside packet is destined to the VIP address configured on its tunnel device. Then, the server processes the request and returns the response directly to the client.

Packet forwarding. This approach assumes that the Web switch and the server nodes are on the same local network. More specifically, the switch and the servers must have one of their network interfaces physically linked by an uninterrupted LAN segment.
The virtual IP address is shared by the Web switch and all of the servers in the cluster through the use of primary and secondary IP addresses. That is, each server is configured with the VIP address as secondary address. This may be done through the use of loopback interface aliasing, for example by using the `ifconfig` Unix command.
Even if all nodes share the VIP address, the inbound packets reach the Web switch because the server nodes have disabled the Address Resolution Protocol (ARP) mechanism (otherwise, a collision would occur). Hence, the Web switch can forward the inbound packet to the target server by using its physical address on the LAN (i.e., the MAC address) without modifying the TCP/IP header. The packet forwarding is achieved by letting the Web switch rewrite the layer-2 destination address to the MAC address of the server and retransmitting the frame on the network. For this reason, packet forwarding is also referred to as *MAC address translation*.

Unlike packet single-rewriting and packet tunneling mechanisms, the server private addresses are now at MAC level.

When the server receives the forwarded packet, it processes it as a packet destined for itself, since it shares the VIP address. Then, it returns the response directly to the client.

3.2 Solutions based on layer-7 switches

Layer-7 Web switches work at application level, thus allowing content-aware request distribution. The mechanisms for layer-7 routing are more complex than those for content-blind routing, because the HTTP request is inspected before any dispatching decision. To this purpose, the Web switch must first establish a TCP connection with the client (i.e., the three-way handshake for the TCP connection setup phase must be completed between the client and the Web switch) and then receive the HTTP request at the application level. On the other hand, a layer-4 Web switch determines the target server as soon as it receives the initial TCP SYN packet, before the client sends out the HTTP request. Figure 8 shows the different instants in which the request decision is made by a Web switch that routes new connections at layer-4 or layer-7.

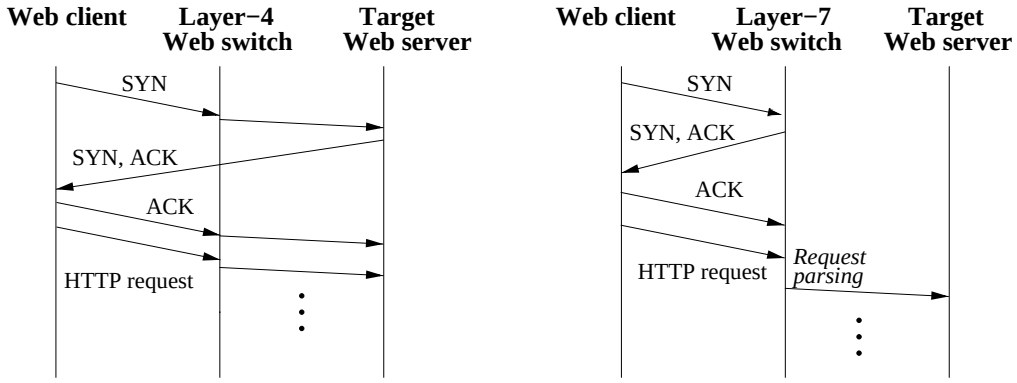

Figure 8: Operations of layer-4 routing (left) and layer-7 routing (right).

Similarly to layer-4 solutions, Web cluster architectures based on a layer-7 Web switch can be further classified on the basis of the mechanism used to send outbound packets from server to client. If we consider the data flow through the Web switch, we can distinguish among *one-way* architectures and *two-way* architectures.

3.2.1 Two-way architectures

In *two-way* architectures outbound traffic must pass back through the Web switch, as shown in Figure 9.

The proposed approaches basically differ over the mechanism the Web switch uses to route requests to the target server.

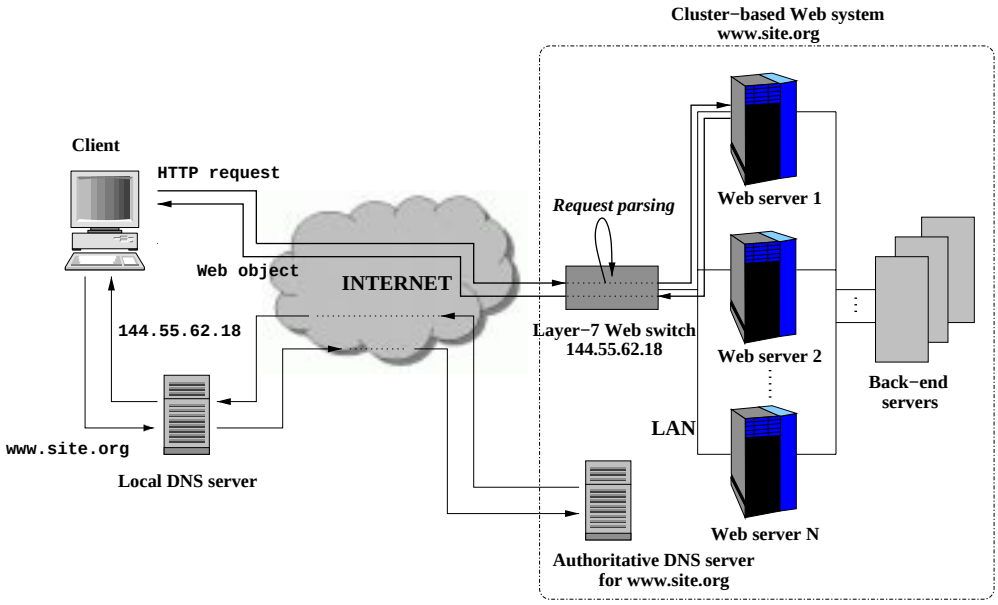

Figure 9: Layer-7 two-way architecture.

TCP gateway. In this architecture, an application level proxy running on the Web switch mediates the communication between the client and the server. This proxy accepts client connections and maintains TCP persistent connections with all the server nodes. When a request arrives on a client connection, the proxy forwards the client request to the target server through the corresponding TCP persistent connection. When the response arrives on the persistent connection back from the server, the Web switch forwards it to the client through the other connection.

TCP splicing. This mechanism aims to improve the TCP gateway approach that is computationally expensive. Now, packet forwarding occurs at network level between the network interface driver and the TCP/IP stack and is carried out directly by the operating system [34].

Once the TCP connection between the client and the Web switch has been established and the persistent TCP connection between the switch and the target server has been chosen, the two connections are spliced together. In such a way, IP packets are forwarded from one endpoint to the other without having to cross the TCP layer up to the application layer on the Web switch. Once the client-to-server binding has been determined, the Web switch handles the subsequent packets by changing the IP and TCP packet headers (IP addresses and checksum recalculations), so that both the client and the target server can recognize these packets as destined to them.

The TCP splicing mechanism can be also implemented by hardware-based switches (e.g., [5]).

3.2.2 One-way architectures

In *one-way* architectures the server nodes return outbound packets directly to clients, without passing through the Web switch, as illustrated in Figure 10.

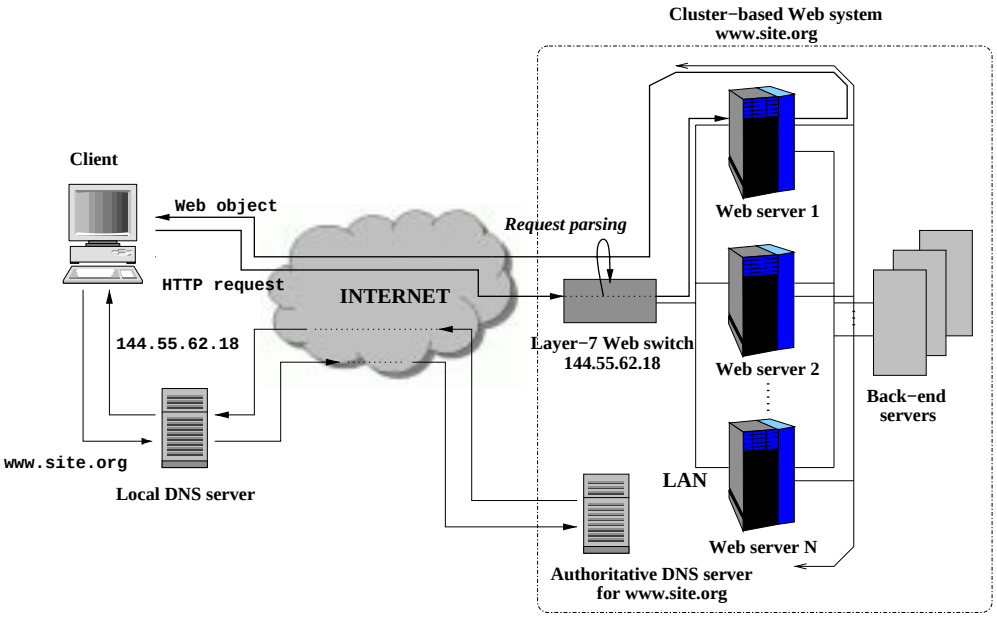

Figure 10: Layer-7 one-way architecture.

TCP handoff. Once the Web switch has established the TCP connection with the client and selected the target server, it hands off its endpoint of the TCP connection to the server [79]. The handoff protocol is layered on top of TCP and runs on the Web switch and the servers, thus requiring changes in the operating systems of both components. The TCP handoff mechanism remains transparent to the client, as data sent by the servers appear to be coming from the Web switch. Any acknowledgment packets sent by the client to the switch are forwarded to the target server by a module running at the bottom of the switch protocol stack.

The handoff mechanism allows also to handle HTTP persistent connections by letting the Web switch assign HTTP requests in the same connection to different target servers [8]. There are two mechanisms to support persistent connections: the handoff protocol can be extended by allowing the Web switch to migrate a connection between servers (*multiple handoff*), the first target server forwards the request it cannot serve to a second server that sends the response back to the client (*back-end forwarding*).

TCP connection hop. This a software-based proprietary solution implemented by Resonate [86]. Once the Web switch has established the TCP connection with the client and selected the target server, it hops the TCP connection to the server. This is achieved by encapsulating

the IP packet in an RPX packet and sending it to the server [86]. Since the server shares the same VIP address, it can reply directly to the client. Acknowledgment packets and persistent session information from clients are managed by the Web switch.

3.3 Comparison of routing mechanisms for Web clusters

Figure 11 summarizes the taxonomy for Web cluster by further detailing the taxonomy previously shown in Figure 5. We first discuss the different solutions for layer-4 and layer-7 routing, and then we compare the two classes of routing mechanisms.

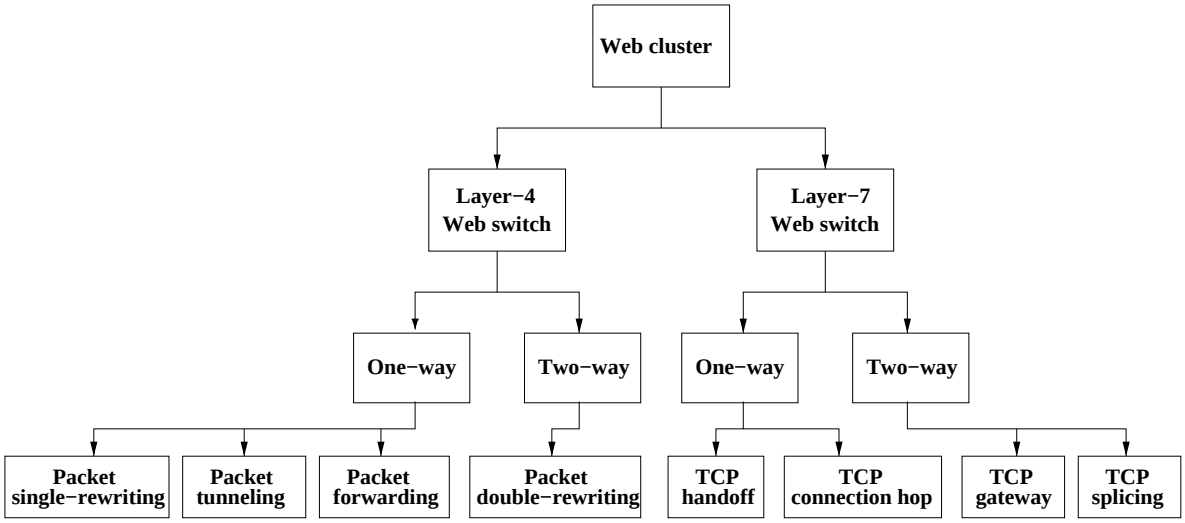

Figure 11: Detailed taxonomy of Web cluster architectures.

3.3.1 Layer-4 routing mechanisms

In two-way solutions, the server nodes may be in different LANs, with the only constraint that both inbound and outbound traffic flow through the Web switch. The main problem of these mechanisms is that the Web switch must rewrite inbound as well as outbound packets, and outbound packets typically outnumber inbound packets. Thus, the scalability (in terms of throughput) of Web clusters that use a two-way architecture is limited by the Web switch ability to rewrite packets and recalculate their checksums, even if a dedicated hardware support can be provided for the checksum operations. On the other hand, a layer-4 Web switch that uses a one-way solution can sustain a larger throughput before becoming the system bottleneck. Thus, the system performance is only constrained by the ability of the switch to set up, look up, and delete entries in the binding table.

If we consider the different mechanisms for one-way architectures, we see that packet single-rewriting causes the same overhead as packet double-rewriting, but it reduces switch operations because the more numerous outbound packets are rewritten by the Web servers and not by the Web switch. However, this requires modifications in the kernel of the server operating system.

Packet forwarding mechanisms aim to limit the overhead of packet rewriting. As the Web switch processes only inbound packets and, beside that, at MAC level that avoids expensive checksum recalculations, the cluster scalability is primarily limited by the capacity of the cluster network link to Internet. The drawback of packet forwarding is that it requires the same network segment to connect the Web switch and all the Web server nodes. However, this restriction has a very limited practical impact since the Web switch and the servers are likely to be connected through the same LAN.

Solutions that employ packet tunneling have good scalability, although lower than packet forwarding. Moreover, they require the servers to support IP tunneling which is not yet a standard for current operating systems.

3.3.2 Layer-7 routing mechanisms

An advantage of two-way architectures combined with a layer-7 Web switch is that caching can be implemented on the Web switch. This allows such device to reply directly to a request if it can be satisfied from the cache with a consequent load decrease on server nodes.

A simple implementation is the main advantage of the TCP-gateway approach that can be implemented on any operating system. The main drawback is the cost of this solution as both request and response data flows through the Web switch up to the application level. TCP splicing reduces TCP gateway overhead, because it eliminates the expensive copying and context switching operations that result from the use of an application-level proxy. However, even in this instance, the Web switch can easily become the bottleneck of the cluster as it needs to modify the TCP/IP headers.

Layer-7 Web switches that use one-way solutions enable the server nodes to respond directly to the clients, thus offering higher scalability than two-way solutions. In particular, the TCP handoff approach scales better than TCP splicing as shown in [9]. The main drawback of one-way solutions at layer-7 lies in that they require modifications in the operating system of both the Web switch and the servers.

3.3.3 Layer-4 vs. layer-7 routing

The main advantage of layer-7 routing mechanisms over layer-4 solutions is the possibility of using content-aware dispatching algorithms at the Web switch. We will see in Section 5.3 that through these policies it is possible to achieve high disk cache hit rates, partition the Web content among the servers, employ specialized server nodes, assign subsequent SSL sessions to the same server, and achieve a fine grain even with HTTP/1.1 persistent connections.

On the other hand, layer-7 routing mechanisms introduce severe processing overhead at the Web switch to the extent that may cause the dispatcher to severely limit the Web cluster scalability [9, 91]. As an example, Aron et al. show in [9] that the peak throughput achieved by a layer-7 switch that employs TCP handoff is limited to 3500 conn/sec, while a software based layer-4 switch implemented on the same hardware is able to sustain a throughput up to 20000 conn/sec.

To overcome this drawback, alternative solutions for scalable Web server systems, which combine content-blind and content-aware request distribution, have been proposed. They are described in Section 7.

Table 1 outlines the main features and tradeoffs of the various mechanisms we have discussed for Web cluster architectures.

Approach	*Routing*	*Data flow*	*Pros*	*Cons*
Layer-4 (two-way)	Content-blind	Inbound/outbound	Flexible, portable	Switch throughput, TCP connection grain control
Layer-4 (one-way)	Content-blind	Inbound	Simple, fast routing	Network topology, TCP connection grain control
Layer-7 (two-way)	Content-aware	Inbound/outbound	Simple, caching, server specialization, HTTP grain control, content partition, SSL session reuse	Switch bottleneck, slowest routing
Layer-7 (one-way)	Content-aware	Inbound	Server specialization, HTTP grain control, content partition, SSL session reuse	Switch bottleneck, complex routing

Table 1: A summary of local routing mechanisms for Web clusters.

4 Request routing mechanisms for distributed Web systems

In this section, we analyze the routing mechanisms that apply to distributed Web systems. We distinguish the mechanisms according to the level at which the decision on client request routing occurs along the path from the client to the Web site. Therefore, we analyze in Section 4.1 DNS-based mechanisms where the routing takes place during the address resolution phase, and in Section 4.2 (re-)routing mechanisms that are carried out by the Web servers. DNS-based level mechanism is used only for the first-level routing and is typically centralized at a single entity, while mechanisms deployed at the Web system are typically distributed and used for the second level of routing or re-routing.

4.1 DNS routing mechanisms

DNS-based routing is the first solution that has been proposed in 1994 to handle multiple Web servers hosting a Web site. It was originally conceived for locally distributed Web systems even if now it is commonly used in geographically distributed Web systems [64]. DNS-based routing intervenes during the address lookup phase at the beginning of the Web transaction when the name of the Web site must be mapped to one IP address of a component of the Web system. Through this simple mechanism, the *authoritative DNS server* (A-DNS) for the Web site can select a different server for every address resolution request reaching it [19]. The A-DNS replies to address requests with a tuple `<IP address, TTL>`, where the first entry is the IP address of one of the nodes in the distributed Web-server system, and the second entry is the Time To Live (TTL) denoting

the period of validity of the name-address mapping that can be cached in the name servers along the path from the A-DNS to the local DNS of the client. In fact, address caching limits the A-DNS control on dispatching of the client requests as it reduces to a small percentage the address requests that actually need the A-DNS server to handle address resolution [36, 44]. (Measures on real traces indicate that the requests under direct control of the A-DNS responsible for a highly accessed Web site are less than 5% of the total requests reaching the system.) Indeed, along the resolution chain between the client and the A-DNS there are several name servers which can hold a valid mapping for the site name. When this mapping is found in one of the name servers on this path and the TTL is not expired, the address request is resolved, thus bypassing the name resolution decision provided by the A-DNS. Only address mapping requests made after the expiration of the TTL in all the name server caches on the path reach the A-DNS. The number of address lookups resolved by the A-DNS is further reduced because of caching at the browser level. As a consequence of both network- and client-level address caching, the DNS-based approach permits a very coarse grain request dispatching.

To limit the effects of address caching at network level and allow for a finer grain request distribution, the A-DNS can specify a very low value for the TTL. However, this approach has its own drawbacks and limits. First of all, it increases the Internet traffic for address resolutions that might overwhelm the A-DNS to the extent that, if not replicated among multiple name servers, the A-DNS becomes the system bottleneck. Moreover, if any user request needs an address resolution, the response time perceived by users is likely to increase [88]. Finally, the TTL period chosen by the A-DNS does not work on browser-level caching and, beside that, low TTL values might be overridden by non-cooperative intermediate name servers that impose their minimum TTL when the suggested value is considered too low.

4.2 Web server routing mechanisms

Some routing mechanisms can be implemented also by the Web servers that can (re)direct a client request to another node. Specifically, we consider the *triangulation* mechanism implemented at TCP/IP level, and *HTTP redirection* and *URL rewriting* mechanisms that work at application level.

4.2.1 Triangulation

When triangulation routing is used, the client continues to send packets to the first contacted server even if the request is actually serviced by a different node. The first node routes client packets to the second server at the TCP/IP level. The routing mechanism is based on packet tunneling [10], which has been described in Section 3.1. Upon the arrival of a new TCP connection request from the Web switch, the Web server decides to serve it locally or to redirect it. In the latter instance, the server encapsulates the original datagram from the client into another datagram. The target node recognizes that the datagram has been re-routed and responds directly to the client. Subsequent packets from the client pertaining to the same TCP connection continue to reach the first contacted

node, which re-routes them to the target server until the connection is closed.

4.2.2 HTTP redirection

The HTTP protocol standard, starting from version 1.0, allows a Web server to respond to a client request with a 301 or 302 status code in the response header that instructs the client to resubmit its request to another node [17, 50]. The built-in HTTP redirection mechanism supports only per URI-based redirection. The status code 301 ("Moved Permanently") specifies that the requested resource has been assigned a new permanent URI and any future reference to this resource will use the returned URI. The status code 302 (corresponding to "Moved Temporarily" in HTTP/1.0 and to "Found" in HTTP/1.1 protocol specifications) notifies the client that the requested resource resides temporarily under a different URI. Figure 12 shows the flow of requests and response when HTTP redirection is activated.

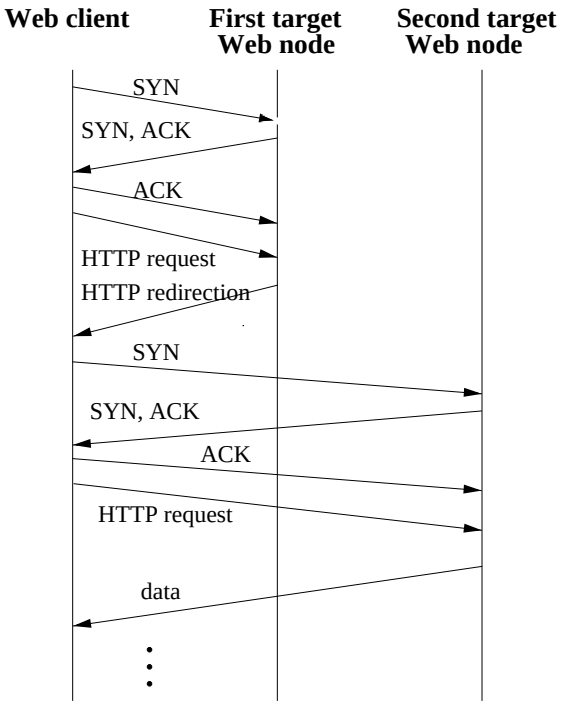

Figure 12: HTTP redirection.

An advantage of HTTP redirection is that replication can be managed at a medium granularity level, down to individual Web pages. Furthermore, HTTP redirection allows content-aware routing, as the first server receiving the HTTP request can take into account the content of the request in selecting another appropriate node.

The main drawback is that this mechanism adds an extra round-trip time to the request processing, as every HTTP redirection requires the client to initiate a new TCP connection with the

destination node. This extra round-trip time increases the network component of the response time; however, it is not automatic that the end user experiences a slower response time for the requested page. Indeed, the increased network time could be compensated by a sensible reduction in the server response time, especially when the first contacted Web node is highly loaded.

A minor drawback of HTTP redirection is due to the fact that most Web browsers do not handle redirection as specified by HTTP standard that requires the browser to display the originally requested URL instead of the redirected URL. Instead, most browsers display and bookmark the name of the new server to which the client has been redirected, thus defeating the routing mechanism when HTTP redirection is implemented by system entities different from Web servers.

4.2.3 URL rewriting

URL rewriting is a quite different mechanism to implement Web server re-routing. Now, the first contacted node changes dynamically the links for the embedded objects within the Web page that it serves to point to another node [67]. Such a mechanism integrated with a multiple-level DNS routing technique is also used by Content Delivery Networks, such as Akamai [2]. The drawback of URL rewriting is that it introduces additional load on the redirecting Web node because each Web page has to be dynamically generated in order to contain the modified object references. Furthermore, it may cause a considerable DNS overhead, as an additional address resolution is necessary to map the new URL into the corresponding IP address. It has been demonstrated that address lookup might take substantially longer than network round-trip time [35].

4.3 Comparison of routing mechanisms for distributed Web systems

DNS-based routing determines the server destination of client requests during the address resolution phase that is typically activated at most once for each Web session. The request distribution among the Web nodes is very coarse because all client requests in a session will reach the same server. Moreover, address caching at intermediate name servers and client browsers further limits the necessity of contacting the A-DNS. Setting TTL to low values allows for a finer grain request distribution, but it might limit general applicability because of the presence of non-cooperative name servers, make the A-DNS a potential bottleneck, and increase the latency time perceived by users.

Although initially conceived for locally distributed architectures (NCSA's Web site), DNS-based routing can scale well geographically. The popularity of this approach for wide-area Web systems and for Content Delivery Networks is increasing due to the seamless integration with standard DNS and the generality of the name resolution process, which works across any IP-based application.

A solution to address DNS issues is to add a second-level routing carried out by the Web servers through a Web system-based re-routing mechanism operating at TCP or HTTP level. The main disadvantage of triangulation is the overhead imposed on the first contacted server, as it must continue to forward client packets to the destination node. That is to say, the triangulation mechanism does not allow the first server to completely get rid of the redirected requests. Moreover,

as triangulation is a content-blind routing mechanism, it requires full content replication, and does not allow fine grain dispatching when the Web transaction is carried out through a HTTP/1.1 persistent connection.

Unlike triangulation-based solutions, application-level Web system mechanisms, such as HTTP redirection and URL rewriting, do not require the modification of packets reaching or leaving the Web-server system. This allows to take into account the requested content in the dispatching decision thus providing also fine grain re-routing. The HTTP redirection is fully compatible to any client software, however its use limits the Web service to HTTP requests only and may cause an increase in response time and network traffic, since a redirected Web page request requires two TCP connections prior to be serviced.

Application-level Web system mechanisms can avoid ping-pong effects that occur when an already re-routed request is further selected for reassignment. A cookie is set when the request is redirected for the first time, so prior to decide about reassignment the server inspects if a cookie is present or not. The triangulation mechanism is free from these side effects as the destination node can deduce if the request has already been re-routed by simply inspecting the source packet address.

Table 2 outlines and summarizes the features and tradeoffs of the various routing mechanisms for distributed Web systems.

Approach	*Data flow*	*Pros*	*Cons*
DNS	Direct	Simple, general applicability	Coarse-grained control, content-blind, limited scalability
Triangulation	Triangular	Simple	Redirecting node overhead, TCP-grained control, content-blind
HTTP redirection	Redirection	Simple, medium-grained control, content-aware	Transmission overhead, HTTP requests only
URL rewriting	Redirection	Fine-grained control, content-aware	Server overhead, HTTP requests only, multiple address lookups

Table 2: A summary of routing mechanisms for distributed Web systems.

5 Dispatching algorithms for cluster-based Web systems

After the analysis of the mechanisms for routing requests, in this section we describe the policies that can be used to select the target server node in a cluster-based Web system. The dispatching policy has an immediate effect on both performance experienced by the users and scalability of the system.

In a Web cluster the dispatching policy is carried out by the Web switch that acts as a global scheduler for the system[1]. Global scheduling policies have been classified in several ways, following

[1]We use the definition of *global scheduling* given in [28] and *dispatching* as synonymous.

different criteria [28, 89, 90, 96]. There are several alternatives, such as *load balancing* vs. *load sharing*, *centralized* vs. *distributed*, and *static* vs. *dynamic* algorithms, but only a subset of them can be actually applied in a Web cluster. This architecture with a single Web switch that receives all incoming requests drives the choice to centralized dispatching policies. Moreover, if we consider that the main load sharing objective is to smooth out transient peak overload periods while load balancing algorithms strive to equalize the loads on all the server nodes [63, 90], it is clear that a Web switch should aim to share rather than to balance cluster workload. Absolute stability is not always necessary and sometime impossible to achieve in a highly dynamic system such as a Web cluster hosting a popular Web site.

The real alternative is therefore among static vs. dynamic algorithms, although the Web switch cannot use highly sophisticated dispatching algorithms because it has to make fast decision for hundreds or thousands of requests per second. *Static* algorithms are the fastest solution to prevent the Web switch from becoming the primary bottleneck of the Web cluster because they do not rely on the current state of the system at the time of decision making. However, 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 in potentially expensive overheads. The requirements listed below summarize the constrains for dispatching algorithms that we will analyze in this section.

1. Low computational complexity, because dispatching decisions are required to be made in real-time.

2. Full compatibility with existing Web standards and protocols.

3. All state information needed by a dispatching policy has to be actually accessible on the Web switch. In particular, the switch and servers of the Web cluster are the only entities that can collect and exchange load information. We do not consider any state information that needs active cooperation from other components that do not belong to the content provider.

5.1 A taxonomy of dispatching algorithms

We have seen that in Web clusters the only practical choice among all global scheduling policies lies in the *static* vs. *dynamic* algorithms. A third class of load sharing policies that has been widely investigated in literature is the class of *adaptive* algorithms, where the load sharing policy as well as the policy parameters can change on the basis of system and workload conditions [90]. However, to the best of our knowledge, no existing Web switch uses adaptive algorithms.

Static dispatching algorithms do not consider any state information while making assignment decisions. Instead, dynamic algorithms can take into account a variety of system state information that depends also on the OSI protocol stack layer at which the Web switch operates. Because of the importance of this factor, we prefer to first classify the dispatching algorithms among *content-blind*

dispatching, if the Web switch works at TCP/IP level, and *content-aware dispatching*, if the switch works at application level.

We then use the literature classification by distinguishing static and dynamic algorithms. It is to be noted that we assume that static algorithms are deployed only by Web switches that operate at TCP/IP level because the use of a sophisticated architecture such as a layer-7 switch is motivated only if its benefits are fully exploited by the dispatching algorithm. Dynamic algorithms can be further classified according to the level of system state information being used by the Web switch. We consider the following three classes.

Client state aware policies. The Web switch routes requests on the basis of some client information. Layer-4 Web switches can use only *network-level* client information such as client IP address and TCP port. On the other hand, layer-7 Web switches can examine the entire HTTP request and make decisions on the basis of more detailed information about the client.

Server state aware policies. The Web switch assigns requests on the basis of some server state information, such as current and past load condition, latency time, and availability. Furthermore, in content-aware dispatching, the switch can also take into account information on the content of the server disk caches.

Client and server state aware policies. The Web switch routes requests by combining client and server state information. Actually, most of the existing client state aware policies belong to this class, because they always use some more or less precise information about the server loads (at least server availability).

Figure 13 summarizes the taxonomy for dispatching algorithms that we have examined so far. We recall that static algorithms as well as server state aware policies are meaningful only for content-blind Web switches operating at TCP/IP level.

5.2 Content-blind dispatching policies

In this section, we describe the main content-blind dispatching policies according to the taxonomy shown in Figure 13, and detailed in Figure 14 with some representative algorithms for each category at the bottom level.

5.2.1 Static algorithms

Static policies do not consider any system state information. Typical examples are **Random** and **Round-Robin** (RR) algorithms. Random distributes the incoming requests uniformly through the server nodes with equal probability of reaching any server. RR uses a circular list and a pointer to the last selected server to make dispatching decisions, that is, if S_i was the last chosen node, the new request is assigned to S_{i+1}, where $i + 1 = (i + 1) \, mod \, N$ and N is the number of server nodes. Therefore, RR utilizes only information on past assignment decision.

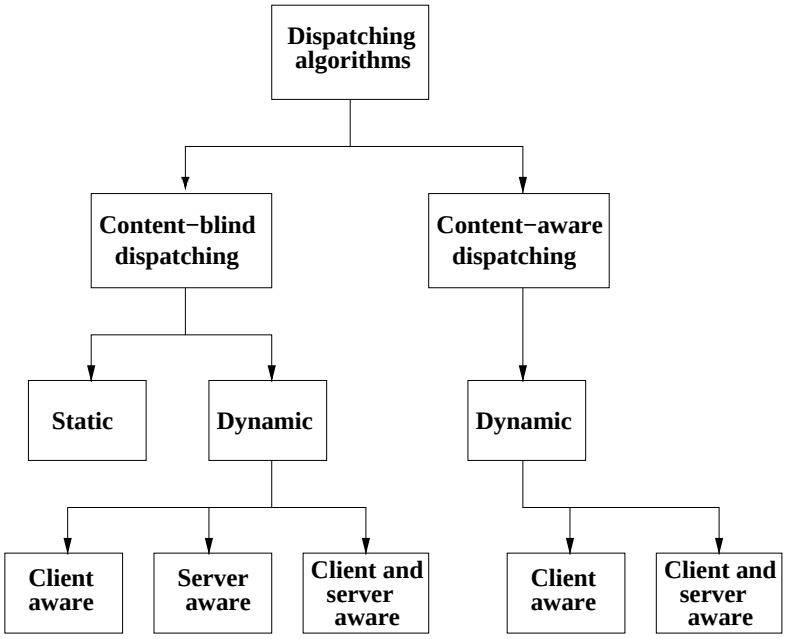

Figure 13: Taxonomy of dispatching policies in Web clusters.

Both Random and RR policies can be easily extended to treat servers with different processing capacities by making the assignment probabilistic on the basis of the server capacity [36]. To this purpose, if C_i indicates the server capacity, the relative server capacity ξ_i ($0 \leq \xi_i \leq 1$) is defined as $\xi_i = C_i/max(C)$, where $max(C)$ is the maximum server capacity among all the server nodes. It is to be noted that the server capacity is a configuration parameter, thus a static information. For Random policy, heterogeneous capacities can be taken into account by assigning different probabilities to the servers according to their capacity. The RR policy can treat heterogeneous server nodes in the following way. A random number ϱ ($0 \leq \varrho \leq 1$) is generated, and, assuming S_i was the last chosen node, the request is assigned to S_{i+1} only if $\varrho \leq \xi_i$. Otherwise, S_{i+2} becomes the next candidate and the process recurs, that is another random number is generated and compared with the relative capacity of S_{i+2}.

Different processing capacities can be also treated by using the so-called **static Weighted RR** (for short, static WRR), which comes as a variation of the Round-Robin policy. Each server is assigned an integer weight w_i that indicates its capacity. Specifically, $w_i = C_i/min(C)$, where $min(C)$ is the minimum server capacity among all the server nodes. The dispatching sequence will be generated according to the server weights [68]. As an example, let us assume that S_1, S_2, and S_3 have the weights 3, 2, and 1, respectively. Then, a dispatching sequence can be $S_1\, S_1\, S_2\, S_1\, S_2\, S_3$.

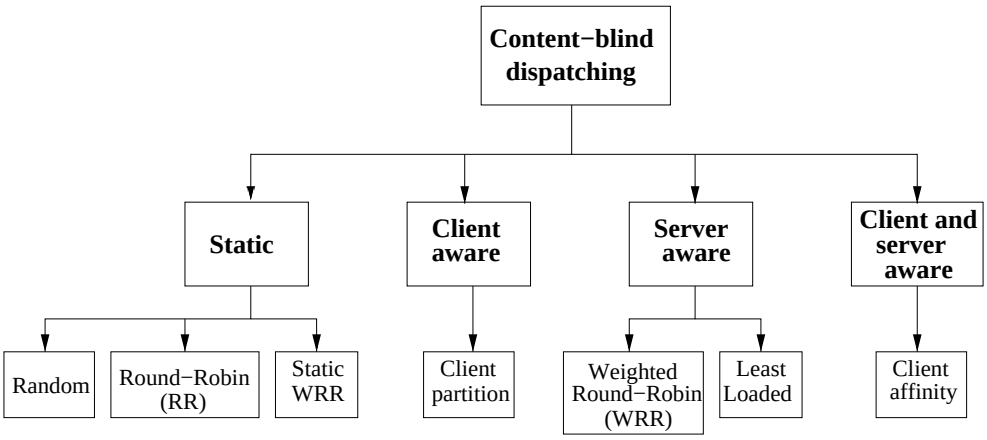

Figure 14: Content-blind dispatching algorithms.

5.2.2 Client state aware algorithms

As layer-4 Web switches are content information blind, the type of information regarding the client is limited to that contained in TCP/IP packets, i.e., IP source address and TCP port numbers. These coarse client information can be used to provide a simple method to statically partition the server nodes and to assign groups of clients identified through their IP address to different servers.

5.2.3 Server state aware algorithms

Client information is immediately available at the Web switch because it receives all requests for connection. On the other hand, when we consider dispatching algorithms that use some server state information we have to address several issues: which *server load index*? How and when to compute it? How and when to transmit it to the Web switch? These are well known problems in networked system [40, 49]. The note in Section 5.4.2 is devoted to the analysis of some alternatives in Web clusters.

Once a server load index is selected, the Web switch can apply different dispatching algorithms. A common scheme is to have the new connection assigned to the server with the lowest load index (the so-called **Least Loaded** policy). For example, in the **Least Connections** policy, which is usually adopted in commercial products (e.g., Cisco's LocalDirector [33], F5's Networks' BIG/ip [48]) the Web switch assigns the new request to the server with the fewest active connections. A simple extension, which assigns static weights to the servers according to their capacity, allows to take into account heterogeneity in server capacity [68]. The underlying idea is that servers with greater capacity should support a larger number of active connections: this can be simply achieved by dividing the number of active connections by the server weight. As a further example, in the **Fastest Response** policy, the Web switch assigns the new connection to the server which is responding faster, i.e., showing the smallest latency time in the last observation interval.

The **Weighted Round-Robin** (WRR) algorithm is a variation of the static WRR. WRR as-

sociates each server with a dynamically evaluated weight that is proportional to the server load state [59]. Periodically, the Web switch gathers load index information from the servers and computes the weights, that are dynamically incremented for each new connection assignment.

5.2.4 Client and server state aware algorithms

Client information available at a layer-4 Web switch (e.g., the source IP address and the service port number within the TCP header) is usually combined with some server state information to provide the so called *client affinity* [59, 68]. Instead of assigning each new connection to a server only on the basis of the server state regardless of any past assignment, consecutive connections from the same client can be assigned to the same server for either performance or functional reasons. As an example of performance reasons, consecutive SSL connections from the same client are assigned to the same server during the life span of the SSL key, so as to avoid time- and resource-consuming operations for SSL key negotiation and generation. An example of functional requirement for client affinity is FTP, as this protocol uses two connections for the same client-server interaction. In policies based on client affinity, the client and server information have a different weight: the client information, when available, usually overrides server information for assignment decisions.

5.2.5 Considerations on content-blind dispatching

For a layer-4 Web switch, static algorithms are the fastest dispatching solution because they do not rely on any system state information in making the decision. Furthermore, they are very easy to implement. However, these stateless algorithms might make poor assignment decisions due to highly variable service times and resource consumption that characterize Web workload.

Dynamic algorithms have the potential to outperform static algorithms by using some state information in the process of dispatching decision. However, they require mechanisms that collect and analyze state information, thereby incurring in potentially expensive overheads (see Section 5.4.2). Furthermore, setting the right parameters of dynamic policies can be a difficult task in highly variable systems such as Web clusters.

Server state aware algorithms seem to be the best choice, even if not all policies work fine. For example, the least loaded approach tends to drive servers to saturation as all requests are sent to the same server until new information is propagated. This "herd effect" is well known in distributed systems [40, 73], yet the least loaded approach is commonly used in commercial products. On the other hand, many experiments and simulation results have demonstrated that the WRR policy compromises simplicity with efficacy at best [27, 59].

5.3 Content-aware dispatching policies

The complexity of layer-7 Web switches that can examine the HTTP request motivates the use of more sophisticated content-aware distribution policies. We detail the taxonomy for content-aware dispatching shown in Figure 13 with an additional level that considers the main goal of the dispatching policies. Figure 15 summarizes the taxonomy of the content-aware dispatching policies

and shows at the bottom level the proposed algorithms that use information about the requested URL for different purposes, such as

- to improve reference locality in the server caches so to reduce disk accesses (*cache affinity*);
- to use specialized server nodes to provide different Web services (*specialized servers*), such as streaming content, dynamic content, and to partition the Web content among the servers, for increasing secondary storage scalability;
- to increase load sharing among the server nodes (*load sharing*).

Furthermore, additional information regarding the HTTP request, such as cookies and SSL identifiers, can be also used to exploit *client affinity* algorithms. Indeed, the SSL protocol involves a computationally expensive handshake procedure (certificates exchange, encryption and compression negotiation, session ID setup), while subsequent SSL sessions can skip the handshake (by using again the same session ID) for a limited period of time. However, it is to be noted that support for stateful services can be also provided by Web switches that operate at TCP/IP level (although with a lower degree accuracy as explained in Section 5.3). Indeed, stateful services can be identified through the service port (e.g., 443 for SSL) and a connection reuse timeout can be set [59].

Finally, when HTTP/1.1 persistent connections are used, a layer-7 Web switch can assign requests traveling on the same TCP connection to different servers, thus achieving a granularity control down to individual HTTP requests. On the other hand, a layer-4 switch must assign the entire TCP connection to the same server. It implies that multiple HTTP requests on the single persistent connection reach the same server, that is the control granularity on which the assignment is activated is at the level of the entire TCP connection. With HTTP/1.0, there is no difference for the granularity control between layer-4 and layer-7 routing because a one-to-one correspondence exists between an HTTP request and a TCP connection.

5.3.1 Client state aware algorithms

Let us first consider the left part of the taxonomy in Figure 15. In cache affinity policies, the file space is typically partitioned among the server nodes. A hash function can be used to perform a static partitioning of the files. The dispatching policy running on the Web switch (namely, **Hash** algorithm) uses the same function. This scheme exploits at maximum the locality of references in the server nodes and achieves the best cache hit rate. However, it can be applied to Web sites providing static content only. Moreover, it ignores load sharing completely, as it is difficult to partition the file space in such a way that the requests are balanced out. Indeed, if a small set of files accounts for a large fraction of requests (a well-know characteristic of Web workload, e.g., [6, 38]), the server nodes serving those critical files will be more loaded than others.

For Web sites providing heterogeneous Web services, the requested URL can be used to statically partition the servers according to the service type they handle. The goal is to employ specialized servers for certain type of requests, such as dynamic content, multimedia files, streaming video [98].

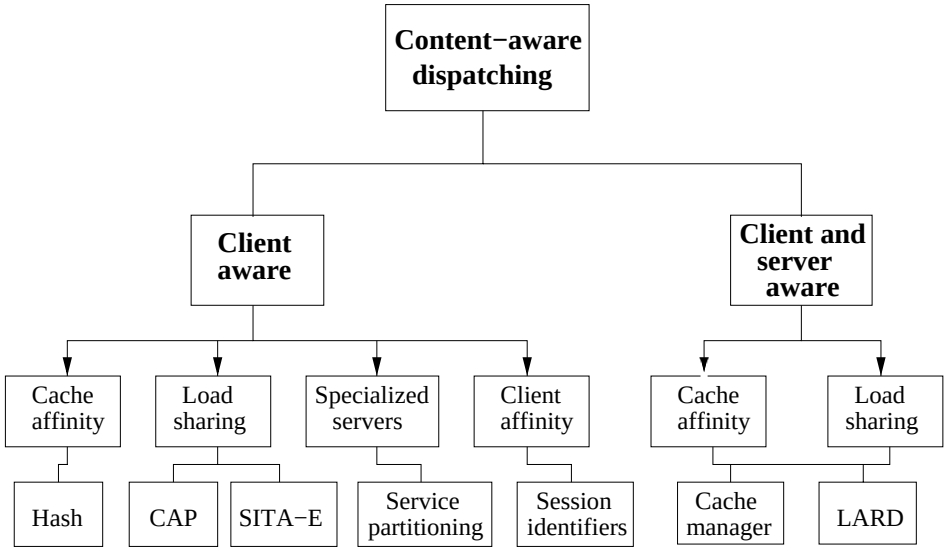

Figure 15: Content-aware dispatching algorithms.

We refer to this policy as to **Service Partitioning**. Most commercial content-aware switches deploy this type of approach (e.g., F5's BIG-IP [48], Resonate's Central Dispatch [86]).

The third main goal of the content-aware dispatching algorithms is to improve load sharing among the server nodes. These strategies do not require static partitioning of the file space and the Web services. Two policies belong to this class: **Size Interval Task Assignment with Equal load** (SITA-E) [56] and **Client-Aware Policy** (CAP) [27]. The former is more oriented to Web sites providing static information, the latter to sites providing Web services with different computational impact on system resources.

The **SITA-E** policy partitions dynamically Web content among the servers according to the file size distribution. It defines the size range associated with each server in such a way that the total load directed to each server is the same [56]. The Web switch determines the size of the requested file and selects the target server on the basis of this information. The goal is to assign light tasks to the nodes that only handle light tasks thus separating light from heavy tasks. The SITA-E policy founds on the basic assumption that the service time of a request is proportional to its size; this assumption is, however, valid for static content only.

Most load sharing problems occur when the Web site provides heterogeneous services that make an intensive use of different Web server resources. Previously described policies do not consider this issue as they either focus on the provision of static content or address multiple services through a static server partitioning. To improve load sharing in Web clusters that provide multiple services, the **CAP** policy takes into consideration the requested service [27]. Web requests are classified on the basis of their impact on main Web server resources, such as static, lightly dynamic, intensive dynamic (or disk bound services, CPU bound services, disk and CPU bound services). Although

the Web switch cannot estimate the service time of a client request accurately, it can distinguish the class of the request from the URL and estimate its impact on main Web server resources. The Web switch manages a circular list of server assignments for each class of Web services. 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 dispatching choice is determined statically once the requested URL has been classified.

The last type of client information that the Web switch can use to select a server is the so-called **session identifier**, like cookies and SSL identifiers. Based on this information, the Web switch can assign all Web transactions from the same client to the same server. Session identifiers provide a powerful means to maintain client affinity at the individual client granularity. In particular, they avoid the limitations of the client IP address identification at a layer-4 Web switch, when Web proxies in the client-server path can squeeze a large number of users into a small number of different IP addresses.

5.3.2 Client and server state aware algorithms

Dispatching algorithms implemented at application level can also use a combination of client and server state information. In this section, we describe two policies that have been specifically designed to consider both client and server information. Other client aware policies (e.g., CAP) can be easily integrated with some server state information. In the taxonomy in Figure 15, the proposed policies belong to two classes because they use client information for cache affinity purposes and server information for load sharing goals.

The **Locality-Aware Request Distribution** (LARD) policy is a content-aware request distribution that considers both locality and load balancing [8, 79]. The basic principle of LARD is to direct all requests for the same Web object to the same server node as long as its utilization is below a given threshold. By so doing, the requested object is more likely to be found into the disk cache of the server node. Some check on the server utilization is useful to avoid overloading servers and, indirectly, to improve load sharing. When a server utilization reaches a given watermark, the dispatcher assigns the request to a lowly loaded node, if it exists, or to the least loaded server. A scheme similar to the LARD policy has been implemented also in the HACC cluster [101].

While in LARD policy the Web switch maintains the mapping from a given file to a set of nodes that serve that file, the **Cache manager** dispatching policy relies on a cache manager that is aware of cache content of all Web servers [20]. Each server provides periodically this information to the cache manager. If the requested object is not cached in any server, the Web switch selects the least loaded server. Otherwise, it selects the lightest loaded server having the object cached, provided that its load is within a threshold over the least loaded server [20].

5.3.3 Considerations on content-aware dispatching

Content-aware dispatching policies based only on client information provide better performance when they take into account the impact of the service being requested, as in the CAP policy [27].

On the other hand, static partitioning algorithms can saturate the capacity of some servers while others are underloaded.

Pure client state aware policies have a great advantage over policies that use also server information, as they do not require expensive and hard to tune mechanisms for monitoring and evaluating the load on each server, gathering the results, and combining them to make dispatching decisions (see Section 5.4.2). However, even client state aware policies should take into account at least a binary server information in order to avoid routing the requests to temporarily unavailable or overloaded servers.

Dispatching policies that aim to improve cache hit rates, such as LARD, give best results for Web sites providing static information and some simple database information. On the other hand, when we consider Web clusters that provide highly heterogeneous content, only policies, such as CAP, that aim to share the load among all (or most) of server components, can provide satisfactory performance [26].

5.4 Analysis of dispatching algorithms

In this section, we first compare content-blind and content-aware dispatching. Then, we give some considerations about pros and cons of using server state information in Web clusters.

5.4.1 Content-blind vs. content-aware dispatching

Content-aware dispatching policies can potentially outperform the content-blind algorithms as they rely on more detailed client information in making the assignment decision. For example, the LARD algorithm shows substantial performance advantages over the WRR strategy when considering static content [8, 27].

On the other hand, operations at layer-7 are expensive, hence client state aware policies must limit the parsing of client information not to cause excessive overhead on the Web switch. For example, a cookie might be 4096 characters long and this information would be carried on many TCP segments. If the Web switch has to inspect every cookie before assigning the corresponding client request, the latency time increases and the Web switch can easily become the system bottleneck.

It is important that new content-aware dispatching algorithm consider also the heterogeneity of Web services and do not focus only on improving cache hit rate of static content. The motivation is that the complexity of services and applications provided by Web sites is ever increasing as demonstrated by the integration of traditional Web publishing sites with e-commerce and Web-based information systems requiring dynamic and secure services. .

5.4.2 A note on server state information

Various issues need to be addressed when we consider dispatching policies based on some server state information: first of all, the choice for one or more *server load index(es)*; then the way to compute the load state information and the frequency of the samples; finally, due to the cluster architecture, some indexes may not be immediately available at the Web switch, so we have to

decide how to transmit them and how frequently. Any of these choices can have a strong impact on final performance of the dispatching algorithms, to the extent that the same policy can behave much better or much worse of other algorithms depending on the quality of the chosen load indexes.

The three main factors that affect the latency time of a Web request are loads on CPU, disk, and network resources of the Web server nodes. Typical server state information includes the CPU utilization evaluated over a short interval, the instantaneous CPU queue length periodically observed, the amount of available memory, the disk or I/O storage utilization, the instantaneous number of active connections, the number of active processes, and the object latency time, i.e., the completion time of an object request in the Web cluster.

Load indexes can be further classified into three classes according to the way they are evaluated: *input indexes*, *server indexes*, and *forward indexes*. Input indexes are computed by the Web switch and do not require any server cooperation. Server indexes are evaluated by each server and transmitted to the Web switch. Forward indexes are information got directly by the Web switch that emulates HTTP requests to the Web servers.

The Web cluster architecture determines the feasibility and convenience of using some load indexes instead of others. For example, input indexes and server indexes can be used in one-way and two-way architectures while forward indexes are meaningful in one-way architectures only, because in two-way architectures the Web switch can keep track of each connection without the need of generating additional traffic in the Web cluster.

Both server and forward indexes typically take longer than input indexes to acquire. However, input index information is limited to the number of active connections that provides a rough estimate of the state of the Web servers as seen by the binding table of the Web switch.

On the other hand, server indexes can provide detailed information such as CPU and disk utilization, object latency time, number of active connections, and the number of processed packets. This last information can be the most useful load index when the number of transferred packets varies greatly from connection to connection, for example, when large files can be transmitted or when HTTP/1.1 persistent connections are used. In one-way architecture, server indexes have to be computed by a process monitor running on each server and periodically transmitted to (or get by) the Web switch. In two-way cluster architectures the same mechanism can be used, otherwise a subset of server indexes (e.g., number of active connections, transmitted packets, object latency time) can be inferred by the Web switch that has a full control on the data flow. Other server indexes, such as CPU and disk utilization, always need a process monitor on the servers and a communication mechanism from the servers to the Web switch.

It is to be noted the combination of a set of load indexes into a single index that reflects the server load is an interesting research issue that has not yet been investigated in the context of Web clusters.

In addition to the choice of the server load index, all server state aware policies face the problem of updating the load information. The intervals between updates of the load indexes need to be evaluated carefully to make sure that the system remains stable. If the interval is too long, performance may be poor because the system is responding to old information about the server

loads. On the other hand, too short intervals can result in system overreaction and instability. A simple strategy for interpreting stale load information that can apply to layer-4 Web switches has been proposed in [40].

6 Classification of products and prototypes

In the last years, the size of Web-server systems market has rapidly expanded: in 1996 it was close to $0 worth, while in 2000 it reached $580M. Therefore, there is a large number of companies that investigate solutions and propose products in this field. In this section, we classify research prototypes and commercial products that implement cluster-based Web systems according to the architecture taxonomy outlined in the Section 3 and illustrated in Figure 11. We can anticipate that commercial products typically use simple dispatching policies, whereas research prototypes have investigated more sophisticated solutions. It is also important to observe that the names of the companies and products tend to change frequently because of continuous merges and acquisitions in this field. For example, in June 2000 Cisco Systems [33] acquired ArrowPoint, one of the first companies to commercialize layer-7 Web switches, while in January 2001 NortelNetworks [76] entered content-aware dispatching market by acquiring Alteon WebSystems that was one of the market leaders.

6.1 Products based on a layer-4 Web switch

Table 3 classifies some commercial products and research prototypes that work at TCP/IP level. Some products, such as Linux Virtual Server [68], appear in the table multiple times as they can be configured to support more than one request routing mechanism.

In the remaining part of this subsection we describe some of research prototypes and commercial products based on a layer-4 Web switch. Specifically, we focus on one-way architectures which use packet forwarding (i.e., IBM Network Dispatcher [59, 61] and ONE-IP [41]) as it turns out to be a scalable solution. We also consider some two-way architectures that use packet double-rewriting. This routing mechanism is implemented either by a special-purpose hardware (i.e., Cisco's LocalDirector [33]) or by a software running on a common operating system (i.e., Magicrouter [4]).

LocalDirector. The LocalDirector product from Cisco Systems [33] is an early commercial implementation of a layer-4 Web switch based on the NAT approach. It offers various dispatching policies, among which the least connections that selects the server with the least number of active connections, and the fastest response algorithm that dispatches the request to the server that was fastest in responding to the previous connection requests.

Through the use of a sticky flag, this device can also support some stateful services, such as SSL. This is accomplished by directing multiple connections from the same client to the same server within a period of time that is set by default to 5 minutes.

Magicrouter. Magicrouter, developed at the University of California at Berkeley, implements a layer-4 Web switch with packet double-rewriting [4]. It uses a mechanism of *fast packet*

Two-way	One-way		
Packet double-rewriting	*Packet single-rewriting*	*Packet tunneling*	*Packet forwarding*
Cisco's LocalDirector [33] Magicrouter [4] Linux Virtual Server [68] LSNAT [92] F5 Networks' BIG/ip [48] Foundry Networks' ServerIron [51] Cyber IQ's HyperFlow [39] HydraWEB's Hydra2500 [60] Coyote Point's Equalizer [37]	TCP Router [44]	Linux Virtual Server [68]	IBM Network Dispatcher [59, 61] Linux Virtual Server [68] ONE-IP [41] LSMAC [54] Intel's NetStructure Traffic Director [62] Nortel Networks' Alteon 780 [76] Foundry Networks' ServerIron [51] Radware's WSD Pro [85]

Table 3: Layer-4 Web clusters.

interposing, where a user level process, acting as a switchboard, intercepts client-to-server and server-to-client packets and modifies them by changing the addresses and checksum fields. To share the load among the Web-server nodes, three dispatching algorithms are considered: round-robin, random and incremental load. The last is similar to selecting the least loaded server and is based on the current server load plus an adjustment related to the number of active connections.

Network Dispatcher. IBM Network Dispatcher [59, 61] is an extension of the basic TCP router [41] and it is now a component of IBM WebSphere Edge Server. Network Dispatcher provides a packet forwarding mechanism, where all the server nodes share the same VIP address. The switch forwards packets, destined to the Web cluster, to a selected server by using its MAC address on the LAN, hence without modifying the TCP/IP headers. The dispatching policy implemented by the Web switch can be dynamically based on server load and availability. Specifically, the Network Dispatcher uses a weighted round-robin algorithm to distribute connections among the server nodes. The Network Dispatcher is able to support stateful services through a *client affinity* mechanism that is similar to Cisco's sticky flag.

ONE-IP. ONE-IP is one of the first implementations of a layer-4 Web switch based on packet forwarding. Implemented at the Bell Labs [41], ONE-IP uses the `if config alias` option to configure the aliasing interface of all Web-server nodes with the same VIP address (here called ONE-IP). Two different dispatching algorithms are supported. Through *routing-based dispatching*, the Web switch selects the destination server based on a hash function, that maps the client IP address into the primary IP address of a server, and then reroutes the packet

to the selected server. Through *broadcast-based dispatching*, the Web switch broadcasts the packets having ONE-IP as destination address to every server in the cluster. A local filtering allows each server to evaluates whether it is the actual destination of the packets. This is done by applying a hash function to the client IP address and comparing the result with its assigned identifier.

The main advantage of the ONE-IP approach is that the Web switch does not need to keep track of any system state information, for example it does not need to maintain the active connections in a binding table. Its weak point is the use of a hash function to select the server, based on the client IP address. Although the hash function could be dynamically modified to provide fault tolerance, this approach is not able to adapt to dynamic conditions when clients unevenly load the servers. Furthermore, the proposed hash function cannot take into account heterogeneous capacity of the servers.

6.2 Products based on a layer-7 Web switch

In Table 4 we classify some commercial products and research prototypes that work at application level. Some products listed herein, such as Foundry Networks' ServerIron [51], have already been considered in Table 3 as they can be configured to support both layer-4 and layer-7 routing mechanisms.

Two-way		**One-way**	
TCP gateway	*TCP splicing*	*TCP handoff*	*TCP connection hop*
IBM Network Dispatcher CBR [61]	[34]	ScalaServer [8, 79]	Resonate's Central Dispatch [86]
CAP [27]	Nortel Networks' Web OS SLB [76]		
HACC [101]	Foundry Networks' ServerIron [51]		
	Cisco's CSS [33]		
	F5 Networks' BIG/ip [48]		
	Radware's WSD Pro+ [85]		
	HydraWEB's Hydra2500 [60]		
	Zeus's Load Balancer [100]		
	[98]		

Table 4: Layer-7 Web clusters.

We examine two research prototypes and two commercial products, as representative examples of each routing mechanism shown in 4.

HACC. The Harvard Array of Cheap Computers (HACC) architecture [101] has been developed at Harvard University with the goal of improving the locality of reference in request streams.

The Web switch (here called *Smart Router*) performs the request routing by working as an enhanced TCP gateway. The Smart Router is partitioned into two layers, the High Smart Router (HSR) and the Low Smart Router (LSR), both layered on top of the TCP/IP stack. The low-level LSR, implemented at kernel level, is in charge of the network Functionalities, i.e., it manages connection setup and termination, forwards requests to server nodes, and responds to the clients. The high-level HSR is responsible for selecting the target server node on the basis of the locality properties and capacity of the nodes. To perform this task, the HSR monitors not only the load on the servers but also the state of the files stored by means of a tree-based structure. Requests for new objects are assigned to the least loaded server and the tree-based structure is updated. Subsequent requests for the same object are assigned to the same server to improve locality of reference in disk cache.

WebOS SLB. Nortel Networks' WebOS SLB [76] performs content-aware dispatching by means of a two-way architecture based on TCP splicing. The Web switch splices the connection established with the client to a connection with a selected server. To this purpose, it modifies the TCP header of every packet that travels between the client and the server to perform TCP/IP header recalculation and sequence number adjustments. To limit the bottleneck risks at the Web switch, WebOS SLB relies on a dedicated network processor integrated with a concurrency operating environment.

The dispatching policy is basically a service partitioning algorithm that allows specialized servers to store specific object types. Hence, the client request is assigned by the Web switch to a target server that is selected according to the type of the file being requested.

ScalaServer. The ScalaServer prototype, developed at Rice University, implements a one-way architecture based on TCP handoff [8, 79]. The request routing occurs at application level and allows the server nodes to send the response packets directly to clients. The client request is accepted by the Web switch that analyzes the URL content and dispatches the request to an appropriate server, selected on the basis of the LARD policy described in Section 5.3. After the server selection, the Web switch informs the target server of the status about the network connection and the server takes over the connection, and communicates directly with the client. Incoming traffic on already established connections is forwarded to the target server through an efficient forwarding module layered at the bottom of the Web switch's protocol stack.

ScalaServer requires that the server operating system be modified in order to support the TCP handoff protocol. On the other hand, the handoff protocol is transparent to the Web server application running on the server nodes, thus allowing the use of any off-the-shelf Web server.

To support HTTP/1.1 persistent connections and assign requests in the same connection to different servers, the ScalaServer prototype has been extended with a *back-end forwarding* mechanism, that allows the original target node to forward a request to a second server selected by the Web switch [8].

Central Dispatch. To the best of our knowledge, Resonate's Central Dispatch [86] is the only commercial product that implements a one-way architecture with a layer-7 Web switch. It offers a proprietary solution called Connection Hop, which resembles the TCP handoff mechanism proposed in [8, 79]. Central Dispatch is a distributed software solution that is loaded on the Web switch and on each Web server node. The connection hop operates at the network layer between the network interface card and the TCP/IP stack, thus minimizing the latency on incoming packets. Dispatching of requests that arrive at the Web switch is based on client information and server availability and performance. Upon receipt of the HTTP request, the switch parses the URL to determine the content being requested. If more than one server is available to serve the request, the Web switch transfers the TCP connection to the least loaded server.

7 Integrated dispatching mechanisms

One common objection raised against the Web cluster architecture concerns the Web switch that represents a single point of failure and a potential system bottleneck. The former problem can be solved only by a replication of the switch device integrated with some heartbeat technique [69] or by a distributed dispatching mechanism [10]. This section focuses on the latter issues and presents some extensions to the basic cluster architecture described in Section 2, with the main goal of improving Web system scalability through a combination of request routing mechanisms. Specifically, we examine the combination of DNS and Web switch mechanisms and the integration of caching mechanisms in the layer-7 Web switch to improve the system scalability. We also consider the combination of layer-4 and layer-7 mechanisms and decoupling of the Web switch functions among distinct nodes to address the scalability issue that may occur in the Web switch.

7.1 Solutions that improve Web cluster scalability

A simple solution to avoid a system bottleneck at the Web switch is to use multiple Web clusters, each with a front-end switch and a visible IP address. During the address resolution phase, the authoritative DNS can dispatch the client requests among the Web clusters through simple static algorithms, such as round-robin. Each Web switch can use another dispatching algorithm to share the load among the Web servers of each cluster. A similar architecture was first proposed by Dias et al. where the Web switch operated at TCP level [44], and it is now adopted by several geographically distributed Web systems.

Another approach for improving system throughput is to integrate a cache into a layer-7 Web switch, to achieve the so called *Web server accelerator* [29]. The caches store frequently accessed Web objects and respond to requests for these objects, thus relieving that workload from the Web servers. The proposed system uses sophisticated mechanisms that allow caching of dynamic objects too [65].

An evolution of the basic Web server accelerator architecture adds another layer of system

components between the Web switch and the Web servers [91]. The additional layer consists of a set of Web server accelerators, basically an array of caches with some dispatching functionality. The goal is to improve Web cluster performance by increasing cache hit rates. The front-end switch can be a layer-4 or a layer-7 Web switch. When a layer-4 switch receives incoming requests, it routes them to the cache nodes regardless of the content of the request. Therefore, the request may be sent to a wrong cache node that does not contain a cached copy of the object. If that happens when the requested object is small, the first node gets the requested object from a node containing the cached copy and sends the response back to the client. Instead, when the requested object is large, the first node hands off the TCP connection to a node containing a cached copy of it, and this node responds directly to the client without passing through the first node or the layer-4 switch. The use of a layer-7 Web switch as a front-end can reduce the work of the cache nodes and limit the percentage of request redirection, however it achieves a lower aggregate throughput because of the overhead of content-aware routing mechanisms.

Another proposal to improve Web cluster efficiency through a global caching mechanism implemented by the Web servers has been studied in [25]. The first dispatching level carried out by the authoritative DNS or a layer-4 Web switch selects one Web server by means of a simple algorithm. Depending on the size of the requested object, the presence of it in the local cache or in the caches of other servers, the first contacted Web server always replies to the client immediately or after having retrieved the file from another server.

7.2 Solutions that improve Web switch scalability

The real risk of a system bottleneck in the front-end exists when the cluster uses a layer-7 Web switch. Indeed, the additional overhead caused by content-aware routing can reduce the system scalability by one order of magnitude with respect to layer-4 switches.

To overcome this drawback, design alternatives for high performance Web clusters propose the combination of layer-4 and layer-7 switches. A layer-4 Web switch is the front-end node of the cluster with its visible IP address. This switch receives all requests destined to the Web cluster and distributes them between two or more layer-7 Web switches with some simple dispatching algorithm. The layer-7 Web switches distribute the client requests received by the layer-4 switch to the Web servers according to some content-aware policy.

Aron et al. [9] designed a prototype in which a front-end layer-4 switch receives client requests and distributes them among the server nodes. In their turn, these nodes may forward the incoming request to another server node based on the requested content. A *distributor* component resides on each server node, in such a way that the HTTP server and the distributor processes are co-located on the same node. Upon receiving a new TCP connection request, the layer-4 Web switch selects one distributor on the basis of the server load. The distributor accepts the connection, parses the client request, and contacts a *dispatcher* located on the internal LAN for the assignment. If the dispatcher selects a different server, the distributor hands off the connection through the TCP handoff protocol towards the server chosen by the dispatcher. Then, the server responds

directly to the client without going through the layer-4 switch. When a TCP connection handoff occurs, the distributor sends a message to the layer-4 switch by instructing it to route packets not to the original server but to the node chosen by the dispatcher. In this architecture the second-level dispatching decision is centralized and executed by a single dispatcher, while the second-level routing is distributed and carried out by each distributor component running on the server nodes. The motivation for this choice is that the processing overhead on a layer-7 switch is caused by the routing mechanism and not by the server selection task. However, even if in [9] the authors show that the dispatcher node is not the system bottleneck, a distributed dispatching algorithm carried out by the server node could avoid the communication overhead caused by a centralized dispatcher.

The Multi-Node Load Balancing (MNLB) architecture from Cisco Systems [33] proposes an alternative solution for improving Web switch scalability. To eliminate the scalability limitations that occur in two-way layer-4 architectures (e.g., Cisco's LocalDirector [33]), MNLB separates the packet-by-packet processing functions (changing addresses and port numbers, recalculating checksums) from the server selection action. The front-end switches, called *forwarding agents*, receive incoming packets from the clients. Multiple switches can be used to spread the load for reliability and scalability reasons (the switch selection can be carried out by the authoritative DNS server for the Web site or through the use of a multicast MAC address.) If the packet belongs to a new connection, the agent forwards it to the *service manager* node which makes the dispatching decision based on server information. The manager informs the agent about its decision, so subsequent packets belonging to the same connection are forwarded directly to the target server by the agent without passing through the service manager.

8 Placement of Web content and services

The scalability of a Web cluster depends also on the methods used to organize and access information within the site. Data placement is a widely investigated research topic in distributed systems and distributed databases and cannot be covered completely in one section. We outline main ideas and give references for further reading by distinguishing static information from information that is dynamically generated at the time of a client request.

8.1 Distribution of static information

When we consider locally distributed Web systems that do not use a content-aware dispatching mechanism, any server node should be able to respond to client requests for any part of the provided content tree. This means that each server owns or can access a replicated copy of the Web site content, unless internal re-routing mechanisms are employed. There are essentially two mechanisms for distributing static information among the Web servers of the cluster. One is to replicate the content tree across independent file systems running on the servers; the other is to share information by means of a distributed file system such as Andrew File System (AFS) or Network File System (NFS).

The first technique requires that each server in the cluster maintains a local copy of the Web documents on its local disk. In such a way, each server only has to access its own disk, without any extra communication with the other servers of the cluster. However, content replication has a high storage overhead and, even worse, it requires any content update to be propagated to all the nodes in short periods of time. An efficient mechanism for updating and controlling the documents should be implemented to maintain consistency among the data stored on the servers. For sites providing stable information, the data updating could be executed during the night, not to overload the local network and disks in peak hours; furthermore the majority of Web requests are for read-only access, where maintenance of consistency is not crucial. However, if data are highly volatile and frequently updated, the execution of the updating and controlling process may cause heavy network and disk overheads.

The other technique for sharing information uses a distributed file system such as AFS and NFS: each document is divided into logical blocks, which are distributed across the servers disks. The use of a distributed file system ensures the consistency of information and does not require a large amount of disk space. On the other hand, it introduces a communication overhead between the servers and may increase the response time as the server nodes have first to obtain the file information from the file server before sending it to the client. Each technique has its benefits and drawbacks. The choice for the best solution depends on the size of Web content, the frequency of documents updating, the required level of data integrity and security, and the possibility of implementing an efficient caching mechanism.

Web clusters based on layer-7 Web switches can use the same two strategies, i.e., replicating the content tree on each server node or shared it through a distributed file system. However, they can also use a third alternative by partitioning the content tree among the Web server nodes. This technique has two main advantages. It increases secondary storage scalability without the overhead due to a distributed file system. It allows the use of specialized server nodes to improve responses for different file types, such as streaming content, CPU-intensive requests, and disk-intensive requests [48, 86, 98]. On the other hand, content partitioning can lead to load imbalance produced by the uneven distribution of Web documents popularity, because the servers storing hot documents can be overwhelmed by client requests. It is also true that suitable caching mechanisms can alleviate server overload due to hot spots because frequently accessed documents are likely not to require a disk access.

Full replication or full partition of Web content are two opposite choices. If we consider that the access patterns to Web files are highly skewed, a partial replication of the most popular objects among all servers and a partition of the others could be the most cost-effective solution. By carrying this approach to the extremes, Pierre et al. propose a sophisticated mechanism that simultaneously use several strategies for replicating Web content [84]. Indeed, the traditional static placement of (static) data has potential weaknesses as the access pattern might even change quickly. Hence, it would be interesting to investigate dynamic placement approaches that keep statistics about the workload composition and automatically move and/or replicate objects at different Web servers.

8.2 Distribution of dynamic services

Although at the beginning Web is largely based on static and read-only information, more and more Web sites provide Web pages that are personalized for the client or created dynamically by the execution of some application process. While a Web server node can typically deliver several hundred static files per second, dynamic pages often require orders of magnitude higher costs. Nevertheless, dynamic pages and services are essential in modern Web sites where Web technologies have emerged as a valid alternative to traditional client-server computing. Indeed, the Web simplicity and its compatibility with existing protocol is making this technology the preferred standard interface for accessing many services exploited through computer networks. The so called *Web-based information systems* or *Web-based enterprise applications* are mainly based on complex interaction of several processes that require dynamic services and computation.

Dynamic Web services, databases and other (legacy) applications are typically hosted on a set of servers different from the Web server nodes. Hence, the Web cluster architecture for e-commerce Web sites and Web-based information systems tend to have a multi-tiered structure, where all the machines providing the same services are connected by the same LAN segment. A typical architecture is shown in Figure 16. A Web switch is located between the Internet and the first set of Web server nodes (presentation layer). A so called application server layer can be interposed between the Web servers and the back-end servers (data layer). (In less complex architectures where the application layer is thin, application processes can run on the Web server nodes).

The Web server nodes run the HTTP daemons that listen on some network port for the client requests assigned by the Web switch, prepare the content requested by the clients, send the response back to the clients or to the Web switch depending on the cluster architecture, and finally return to the listen status. The Web server nodes are capable of handling requests for static content, whereas they forward requests for dynamic content to back-end nodes. To this purpose, while a Web server is preparing the requested content, it might call an application program running on an application server. These application servers run the software that handles all operations between browser-based clients and a company's back-end databases. The application server accepts requests from Web servers, executes the business logic, and interacts with database servers or other legacy applications. The database server layer hosts and maintains databases, and provide powerful database manipulating functions to the application servers.

Strict security strategies are employed to protect the safety of these Web clusters. A boundary firewall typically connects the Web switch to Internet. Another firewall interconnects the LAN segment of the Web server layer with that of the application server layer. A third firewall is interposed between the database server layer and the application server layer. These firewalls are configured to filter all traffic among the server layers so that, for example, the database servers can only be contacted by the application servers which, in their turn, can only be reached by the Web servers.

The generation of dynamic content opens several new issues that are beyond the scope of this paper. The alternative solutions depend also on the application software, the chosen middleware

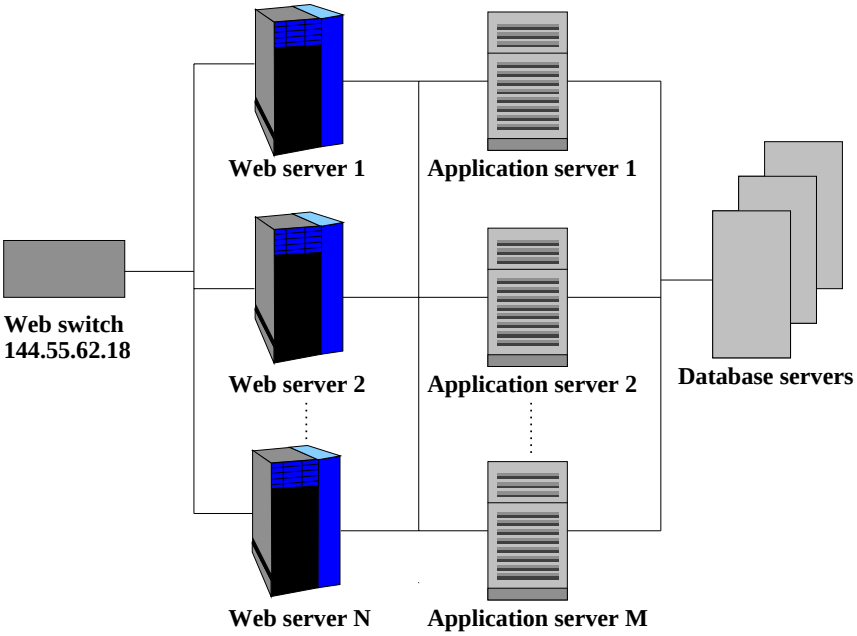

Figure 16: A multi-tier architecture for a Web cluster.

and database technology. For example, commercial Web service softwares such as BEA WebLogic and IBM WebSphere have evolved from simple Web servers into complex Web application servers that use CGI, Java Server Page, Microsoft Active Server Pages, XML, and other technologies. Among them, CGI, Server Side Includes, Server API (e.g., Netscape NSAPI, Microsoft ISAPI) and Java can be used for managing dynamic requests at the server layer. The software at the application layer is basically a gateway from Web servers to databases and legacy applications. A wide spectrum of new technologies is coming on the scene, such as Cold Fusion, Domino, WebBuilder, IBM DB2 WWW Connection, Oracle Web Application Server, Informix Universal Web Connect, and Sybase Dynamo Netscape Application Server.

As a consequence, the operations for dynamic services might be highly sophisticated and involve several processes. For example, with the Enterprise Java Beans (EJB) technology, used by Persistence PowerTier and BEA WebLogic Server, processing one dynamic request might involve the Web switch, Web server, servlet, session-bean, and entity-bean before reaching the database.

The level of multiple indirections in multi-tier Web cluster architectures where each layer consists of multiple server nodes allows request routing and dispatching to be implemented at different levels, from the Web switch to the Web server layer to the Web application server layer. For example, application servers can support application partitioning by distributing application logic among multiple servers. Similarly, the components of large-scale applications can be grouped to facilitate partitioning and management. A multi-tier cluster architecture might use a *second-level Web switch*, that is in charge of selecting an appropriate application server node for requests that

need dynamic processing. This approach to request routing can be further extended to the third level of server nodes, composed of back-end servers which respond to queries originated by the application servers.

Zhu et al. [102] have studied the problem of request dispatching in a multi-tier architecture, where the second-level switch is integrated in each Web server node, called *master node*. It selects the appropriate *slave node* that has to process the dynamic request and sends the result back to the master. The slave node selection is based on a prediction model that estimates the expected cost for processing the dynamic request on each slave node. Similar multi-tier architectures have been also analyzed in [52]. A different solution for load balancing based on CORBA middleware technology is proposed in [78] where several dispatching strategies have been also proposed and evaluated.

We can summarize that request dispatching, load sharing and load balancing at the internal layers are implemented in most commercial products, but they are not widely studied topics in the research community. Indeed, probably due to the complexity of achieving an optimal dispatching, all products prefer to use very simple algorithms, typically round-robin and least loaded. The most original proposals in this field are not oriented towards load balancing, but rather they aim to improve performance of the multi-tier systems through caching of query results and dynamic content [21, 42, 83, 77].

9 Summary and research perspectives

Much effort has been devoted in recent years to improve the scalability of systems supporting Web sites. Systems with multiple nodes are the leading architectures to build highly accessed Web sites that have to guarantee scalable services and to support ever increasing request load. In this paper, we have analyzed routing mechanisms and dispatching algorithms that are suitable for locally distributed Web systems. We have proposed an original taxonomy of the architectures, the routing mechanisms and dispatching algorithms. Based on this material, we have analyzed the efficiency and the limitations of the different techniques and evaluated the tradeoff among the considered alternatives. In this section we present some research topics that are likely to impact future Web cluster architectures.

The Web is becoming the standard interface for accessing remote services and applications, and there is no doubt that Web clusters will be the basic architecture for Web-based information systems, Web hosting centers, and Application Service Providers. Hence, there is general consensus that the research interest in this field is likely to increase. This conclusion is also motivated by the observation that the performance problems of Web-based architecture will tend to become worse because of the proliferation of heterogeneous client devices, the need of client authentication and system security, the increased complexity of middleware and application software, and the high availability requirements of corporate data centers and e-commerce Web sites.

One research path is in the direction of combining performance with fault-tolerance and security, and accessibility from different client devices, all topics that are still seen as separate issues in

Web clusters. A step further is the objective of designing Web clusters that give guaranteed performance or support quality of service (QoS). A significant amount of QoS related research has focused on the network infrastructure, however network QoS alone is not sufficient to support end-to-end QoS. To avoid that high priority traffic may be dropped at the server, the Web-server system should also have mechanisms and policies for delivering end-to-end QoS. Some proposals for providing differentiated service qualities to different classes of users have focused on single-node Web servers [13, 18, 31, 47, 66, 72, 81, 93] and this research topic has been moved towards Web cluster-based platforms only recently. A multi-node Web architecture integrated with admission control and performance isolation mechanisms has been proposed in [7]. Some recent research efforts focus on how providing QoS support through the Web switch, by taking into account request information at layer-4 switches [23] as well as at layer-7 switches [30, 103], where a detailed content-aware information allows to achieve performance isolation in Web clusters at a server-level granularity. In particular, resource utilization can be improved by dynamically adjusting server partitions based on fluctuating requests arrival rates and servers load conditions [22, 103].

While layer-4 Web cluster architectures may be considered a solved problem, the area of content-aware architectures needs further research. Dispatching algorithms that combine effectively client and server information, and adaptive policies are not yet fully explored. Some companies commercialize layer-7 Web switches with very simple dispatching mechanisms that are mainly oriented to a static partition of the Web content and services among the server nodes. Also, the scalability problem posed by layer-7 routing has not been completely solved and non-centralized dispatching algorithms can be a theme of in-depth investigation. A related issue is to avoid state information inconsistency among the multiple dispatchers.

Even more challenging is the study of optimal resource management in multi-tier architectures because the multitude of involved technologies and complexity of process interactions occurring at the middle-tier let the vast majority of products prefer quite naive dispatching algorithms and solutions. Combining load balancing and caching of dynamic content in multi-tier systems is also worth of further investigation.

The actual improvement of the response time as perceived by the user comes from a combination of technologies, where the multiplication of content provider servers is integrated with geographically dispersed cache servers supported by the content providers themselves or by third-party organizations. Techniques for solving the problems and taking advantage of the potentials originated by the cooperation of multiple servers and multiple caches (e.g., dynamic placement of content, data prefetching, consistency) are still in their infancy, as well as the analysis of the mutual effects of content delivery caching and load distribution [45]. Finally, we observe that most of the topics and algorithms analyzed in this paper change completely if we assume that the multiple servers (or clusters) of the content provider are distributed on a geographic rather than on a local area.

References

[1] Adero. `http://www.adero.com`.

[2] Akamai. `http://www.akamai.com`.

[3] V. A. F. Almeida, I. M. M. Vasconcelos, and J. N. C. Arabe. The effect of the heterogeneity on the performance of multiprogrammed parallel systems. In *Proc. of Workshop on Heterogeneous Processing*, Beverly Hills, CA, 1992.

[4] E. Anderson, D. Patterson, and E. Brewer. The Magicrouter, an application of fast packet interposing. `http://www.cs.berkeley.edu/~eanders/projects/magicrouter/`, May 1996.

[5] G. Apostolopoulos, V. Peris, P. Pradhan, and D. Saha. Securing electronic commerce: Reducing the SSL overhead. *IEEE Network*, 14(4):8–16, July/Aug. 2000.

[6] M. F. Arlitt and T. Jin. A workload characterization study of the 1998 World Cup Web site. *IEEE Network*, 14(3):30–37, May/June 2000.

[7] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proc. of ACM Sigmetrics 2000*, pages 90–101, Santa Clara, CA, June 2000.

[8] M. Aron, P. Druschel, and Z. Zwaenepoel. Efficient support for P-HTTP in cluster-based Web servers. In *Proc. of USENIX 1999 Conf.*, pages 185–198, Monterey, CA, June 1999.

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

[10] L. Aversa and A. Bestavros. Load balancing a cluster of Web servers using Distributed Packet Rewriting. In *Proc. of IEEE Int'l Performance, Computing, and Communication Conf.*, pages 24–29, Phoenix, AZ, Feb. 2000.

[11] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. Enhancing the Web infrastructure: From caching to replication. *IEEE Internet Computing*, 1(2):18–27, Mar./Apr. 1997.

[12] G. Banga and P. Druschel. Measuring the capacity of a Web server under realistic loads. *World Wide Web*, 2(1):69–83, May 1999.

[13] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. *World Wide Web*, 2(1-2), 1999.

[14] G. Banga and J. C. Mogul. Scalable kernel performance for Internet servers under realistic loads. In *Proc. of USENIX 1998 Conf.*, New Orleans, LA, June 1998.

[15] P. Barford and M. E. Crovella. Critical path analysis of TCP transactions. In *Proc. of ACM Sigcomm 2000*, pages 127–138, Stockholm, Sweden, Aug. 2000.

[16] G. Barish and K. Obraczka. World Wide Web caching: Trends and techniques. *IEEE Communications*, 38(5), May 2000.

[17] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0.* RFC 1945, May 1996.

[18] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 13(5):64–71, Sept./Oct. 1999.

[19] T. Brisco. *DNS support for load balancing.* RFC 1794, Apr. 1995.

[20] R. B. Bunt, D. L. Eager, G. M. Oster, and C. L. Williamson. Achieving load balance and effective caching in clustered Web servers. In *Proc. of 4th Int'l Web Caching Workshop*, pages 159–169, San Diego, CA, Apr. 1999.

[21] K. S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal. Enabling dynamic content caching for database-driven Web sites. In *Proc. of 2001 ACM SIGMOD Int'l Conf. on Management of Data*, 2001.

[22] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli. Web switch support for differentiated services. *ACM Performance Evaluation Review*, to appear in 2001.

[23] V. Cardellini, E. Casalicchio, M. Colajanni, and S. Tucci. Mechanisms for quality of service in Web clusters. *Computer Networks*, to appear in 2001.

[24] V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic load balancing on Web-server systems. *IEEE Internet Computing*, 3(3):28–39, May/June 1999.

[25] E. V. Carrera and R. Bianchini. Efficiency vs. portability in cluster-based network servers. In *Proc. of 8th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, June 2001.

[26] E. Casalicchio, V. Cardellini, and M. Colajanni. Content-aware dispatching algorithms for cluster-based Web servers. *Cluster Computing*, 5(2), 2002.

[27] E. Casalicchio and M. Colajanni. A client-aware dispatching algorithm for Web clusters providing multiple services. In *Proc. of 10th Int'l World Wide Web Conf.*, pages 535–544, Hong Kong, May 2001.

[28] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. on Software Engineering*, 14(2):141–154, Feb. 1988.

[29] J. Challenger, A. Iyengar, P. Dantzig, D. Dias, and N. Mills. Engineering highly accessed Web sites for performance. In Y. Deshpande and S. Murugesan, editors, *Web Engineering*, pages 247–265. Springer-Verlag, 2001.

[30] X. Chen and P. Mohapatra. Providing differentiated service from an Internet server. In *Proc. IEEE Int'l Conf. on Computer Communications and Networks*, Boston, MA, Oct. 1999.

[31] L. Cherkasova and P. Phaal. Session based admission control: a mechanism for improving performance of commercial Web sites. In *Proc. of Int'l Workshop on Quality of Service*, London, June 1999.

[32] L. Cherkasova and S. Ponnekanti. Optimizing the "content-aware" load balancing strategy for shared Web hosting service. In *Proc. of IEEE Mascots 2000 Symp.*, pages 492–499, San Francisco, CA, Aug./Sept. 2000.

[33] Cisco Systems. `http://www.cisco.com/`.

[34] A. Cohen, S. Rangarajan, and H. Slye. On the performance of TCP splicing for URL-aware redirection. In *Proc. of USENIX Symp. on Internet Technologies and Systems*, Boulder, CO, Oct. 1999.

[35] E. Cohen and H. Kaplan. Proactive caching of DNS records: Addressing a performance bottleneck. In *Proc. of Symp. on Applications and the Internet*, pages 85–94, San Diego, CA, Jan. 2001.

[36] M. Colajanni, P. S. Yu, and D. M. Dias. Analysis of task assignment policies in scalable distributed Web-server systems. *IEEE Trans. on Parallel and Distributed Systems*, 9(6):585–600, June 1998.

[37] Coyote Point. `http://www.coyotepoint.com`.

[38] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Trans. on Networking*, 5(6):835–846, Dec. 1997.

[39] Cyber IQ Systems. `http://www.holontech.com/`.

[40] M. Dahlin. Interpreting stale load information. *IEEE Trans. on Parallel and Distributed Systems*, 11(10):1033–1047, Oct. 2000.

[41] O. P. Damani, P. E. Chung, Y. Huang, C. Kintala, and Y.-M. Wang. ONE-IP: Techniques for hosting a service on a cluster of machines. *Computer Networks*, 29(8-13):1019–1027, 1997.

[42] L. Degenaro, A. Iyengar, I. Lipkind, and I. Rouvellou. A middleware system which intelligently caches query results. In *Proc. of ACM/IFIP Middleware 2000*, pages 24–44, Palisades, NY, Apr. 2000.

[43] B. Devlin, J. Gray, B. Laing, and G. Spix. Scalability terminology: Farms, clones, partitions, and pack: RACS and RAPS. Technical Report MS_TR-99-85, Microsoft Research, Dec. 1999.

[44] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available Web server. In *Proc. of 41st IEEE Computer Society Int'l Conf.*, pages 85–92, San Jose, CA, Feb. 1996.

[45] R. Doyle, J. S. Chase, S. Gadde, and A. M. Vahdat. The trickle-down effect: Web caching and server request distribution. In *Proc. of 6th Int'l Workshop on Web Caching and Content Delivery*, Cambridge, MA, June 2001.

[46] K. Egevang and P. Francis. *The IP Network Address Translator (NAT).* RFC 1631, May 1994.

[47] L. Eggert and J. Heidemann. Application-level differentiated services for Web servers. *World Wide Web*, 2(3):133–142, July 1999.

[48] F5 Networks. `http://www.f5labs.com/`.

[49] D. Ferrari and S. Zhou. An empirical investigation of load indices for load balancing applications. In *Proc. of Performance 1987*, pages 515–528, North-Holland, The Netherlands, 1987.

[50] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* RFC 2616, June 1999.

[51] Foundry Networks. `http://www.foundrynet.com/`.

[52] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Proc. of 16th ACM Symp. on Operating Systems Principles*, pages 78–91, Saint-Malo, France, Oct. 1997.

[53] S. Gadde, J. Chase, and M. Rabinovich. Web caching and content distribution: A view from the interior. *Computer Communications*, 24(1-2):222–231, Feb. 2001.

[54] X. Gan and B. Ramamurthy. LSMAC: An improved load sharing network service dispatcher. *World Wide Web*, 3(1):53–59, Jan. 2000.

[55] J. Gray and P. Shenoy. Rules of thumb in data engineering. In *Proc. of IEEE 16th Int'l Conf. on Data Engineering*, pages 3–10, San Diego, CA, Apr. 2000.

[56] M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. *J. of Parallel and Distributed Computing*, 59:204–228, Sept. 1999.

[57] Y. Hu, A. Nanda, and Q. Yang. Measurement, analysis and performance improvement of Apache Web server. In *Proc. of IEEE Int'l Performance, Computing, and Communications Conf.*, Phoenix, AZ, Feb. 1999.

[58] C. Huitema. Network vs. server issues in end-to-end performance. Keynote speech at Performance and Architecture of Web Servers 2000, Santa Clara, CA. `http://kkant.ccwebhost.com/PAWS2000/huitema_keynote.ppt`.

[59] G. S. Hunt, G. D. H. Goldszmidt, R. P. King, and R. Mukherjee. Network Dispatcher: A connection router for scalable Internet services. *Computer Networks*, 30(1-7):347–357, 1998.

[60] HydraWEB Technologies. `http://www.hydraweb.com/`.

[61] IBM. IBM Network Dispatcher. `http://www.ibm.com/software/network/dispatcher`.

[62] Intel. Intel NetStructure. `http://www.intel.com/netstructure`.

[63] O. Kremier and J. Kramer. Methodical analysis of adaptive load sharing algorithms. *IEEE Trans. on Parallel and Distributed Systems*, 3(6):747–760, Nov. 1992.

[64] T. T. Kwan, R. E. McGrath, and D. A. Reed. NCSA's World Wide Web server: Design and performance. *IEEE Computer*, 28(11):68–74, Nov. 1995.

[65] E. Levy-Abegnoli, A. Iyengar, J. Song, and D. Dias. Design and performance of a Web server accelerator. In *Proc. of IEEE Infocom 1999*, pages 135–143, New York, NY, Mar. 1999.

[66] K. Li and S. Jamin. A measurement-based admission-controlled Web server. In *Proc. of IEEE Infocom 2000*, pages 651–659, Tel Aviv, Israel, Mar. 2000.

[67] Q. Li and B. Moon. Distributed Cooperative Apache Web server. In *Proc. of 10th Int'l World Wide Web Conf.*, Hong Kong, May 2001.

[68] Linux Virtual Server project. `http://www.linuxvirtualserver.org/`.

[69] M.-Y. Luo and C.-S. Yang. Constructing zero-loss Web services. In *Proc. of IEEE Infocom 2001*, Anchorage, Alaska, Apr. 2001.

[70] M.-Y. Luo and C.-S. Yang. System support for scalable and reliable and highly manageable Web hosting service. In *Proc. of USENIX Symposium on Internet Technology and Systems*, San Francisco, CA, Mar. 2001.

[71] A. M. Luotonen. *Web proxy servers.* Prentice Hall, 1997.

[72] D. A. Menascé, J. Almeida, R. Fonseca, and M. A. Mendes. Business-oriented resource management policies for e-commerce servers. *Performance Evaluation*, 42(2-3):223–239, Sept. 2000.

[73] M. Mitzenmacher. How useful is old information. *IEEE Trans. on Parallel and Distributed Systems*, 11(1):6–20, Jan. 2000.

[74] D. Mosedale, W. Foss, and R. McCool. Lessons learned administering Netscape's Internet site. *IEEE Internet Computing*, 1(2):28–35, Mar./Apr. 1997.

[75] E. M. Nahum, T. Barzilai, and D. Kandlur. Performance issues in WWWW servers. *IEEE/ACM Trans. on Networking*, to appear in 2001.

[76] Nortel Networks. Nortel Networks Web OS. `http://www.nortelnetworks.com/`.

[77] Oracle Inc. Oracle i-cache. `http://www.oracle.com/`.

[78] O. Othman, C. O'Ryan, and D. C. Schmidt. Strategies for CORBA middleware-based load balancing. *IEEE Distributed Systems Online*, 2(3), Mar. 2001.

[79] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. M. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proc. of 8th ACM Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, San Jose, CA, Oct. 1998.

[80] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable Web server. In *Proc. of USENIX 1999 Conf.*, pages 199–212, Monterrey, CA, June 1999.

[81] R. Pandey and R. Barnes, J. F. Olsson. Supporting quality of service in HTTP servers. In *Proc. of ACM Symp. on Principles of Distributed Computing*, pages 247–256, Puerto Vallarta, Mexico, June 1998.

[82] C. Perkins. *IP encapsulation within IP.* RFC 2003, Oct. 1996.

[83] Persistence Software Inc. Dynamai. `http://www.persistence.com/dynamai`.

[84] G. Pierre, I. Kuz, van Steen M., and A. S. Tanenbaum. Differentiated strategies for replicating Web documents. *Computer Communications*, 24(2):232–240, Feb. 2001.

[85] Radware Inc. `http://www.radware.com/`.

[86] Resonate Inc. `http://www.resonate.com/`.

[87] D. Rosu, A. Iyengar, and D. Dias. Web proxy acceleration. *Cluster Computing*, to appear in 2001.

[88] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of DNS-based server selection. In *Proc. of IEEE Infocom 2001*, Anchorage, Alaska, Apr. 2001.

[89] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE Computer Society Press, 1995.

[90] N. G. Shivaratri, P. Krueger, and M. Singhal. Load distributing for locally distributed systems. *IEEE Computer*, 25(12):33–44, Dec. 1992.

[91] J. Song, E. Levy-Abegnoli, A. Iyengar, and D. Dias. Design alternatives for scalable Web server accelerators. In *Proc. of 2000 IEEE Int'l Symp. on Performance Analysis of Systems and Software*, pages 184–192, Austin, TX, Apr. 2000.

[92] P. Srisuresh and D. Gan. *Load sharing using IP Network Address Translation.* RFC 2931, 1998.

[93] N. Vasiliou and H. L. Lutfiyya. Providing a differentiated quality of service in a World Wide Web server. *ACM Performance Evaluation Review*, 28(2):22–28, Sept. 2000.

[94] R. Vingralek, M. Sayal, Y. Breitbart, and P. Scheuermann. Web++ architecture, design and performance. *World Wide Web*, 3(2):65–77, Apr. 2000.

[95] J. Wang. A survey of Web caching schemes for the Internet. *ACM Computer Communication Review*, 29(5):36–46, Oct. 1999.

[96] Y. T. Wang and R. J. T. Morris. Load sharing in distributed systems. *IEEE Trans. on Computers*, 34(3):204–217, Mar. 1985.

[97] D. Wessels. *Web caching*. O'Reilly and Associates, Cambridge, MA, 2001.

[98] C.-S. Yang and M.-Y. Luo. A content placement and management system for distributed Web-server systems. In *Proc. of 20th IEEE Int'l Conf. on Distributed Computing Systems*, pages 691–698, Taipei, Taiwan, Apr. 2000.

[99] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using Smart Clients to build scalable services. In *Proc. of USENIX 1997 Conf.*, pages 105–117, Anaheim, CA, Jan. 1997.

[100] Zeus Technologies Ltd. Zeus Web server. `http://www.zeustechnologies.com`.

[101] X. Zhang, M. Barrientos, J. B. Chen, and M. Seltzer. HACC: An architecture for cluster-based Web servers. In *Proc. of 3rd USENIX Windows NT Symp.*, pages 155–164, Seattle, WA, July 1999.

[102] H. Zhu, B. Smith, and T. Yang. Scheduling optimization for resource-intensive Web requests on server clusters. In *Proc. of 11th ACM Symp. on Parallel Algorithms and Architectures (SPAA'99)*, pages 13–22, June 1999.

[103] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In *Proc. of IEEE Infocom 2001*, Anchorage, Alaska, Apr. 2001.