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 is augmenting user-perceived response times from popular Web sites, especially in conjunction with special events. System 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 Web system architectures that consist of multiple server nodes distributed on a local area, with one or more mechanisms to spread client requests among the nodes. After years of continual proposals of new system solutions, routing mechanisms, and policies (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 layer, but the main 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 implementing scalable Web-server systems under the control of one content provider. It identifies also some of the open research issues associated with the use of distributed systems for highly accessed Web sites.

*©2002 ACM. Published in *ACM Computing Surveys* Vol. 32, No. 2, June 2002, pp. 263-311.

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

The Web is becoming the standard interface for accessing remote services of information systems, hosting data centers and application service providers. Demands placed on Web-based services continue to grow and Web-server systems are becoming more stressed than ever. The performance problems of Web-based architecture will even worsen 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. Because of the complexity of the Web infrastructure, performance problems may arise in many points during an interaction with Web-based systems. For instance, they may occur in the network because of congested Internet routers, as well as at the Web and database server, either because of under-provisioned capacity or unexpected surge of requests. Although in recent years both network and server capacity have improved, and new architectural solutions have been deployed, response time continues to challenge Web system related research. In this paper, we focus on architectural solutions that aim to reduce the delays due to the Web server site assuming that network issues are examined elsewhere. Immediate applicability of the considered methodologies and increasing demand for more complex services are the main motivations and guidelines for this survey focusing on the server side. Let us briefly outline these two points.

In an Internet-based world with no centralized administration, the Web site is the only component that can be under the direct control of the content provider. Any other component, such as Internet backbones, Web clients, routers and peering points, DNS system, and proxy servers are beyond the control of any Web operator. Hence, we prefer not to consider proposals requiring some intervention on these components that are really hard to be applicable in the short-medium term, because an agreement among multiple organizations is needed.

The other main 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 already increasingly dominated by server delays, especially when contacting busy servers [19]. Recent measures suggest that the Web servers contribute for about 40% of the delay in a Web transaction [71] and it is likely that this percentage will increase in the near future because of a double effect.

- A prediction made in 1995 regarding the network bandwidth estimated that it would triple every year for the next 25 years [65]. This prediction seems approximately correct to some authors [67], while others are more prudent [43]. Independently of this debate, the network capacity is improving faster than the server capacity, that the Moore law estimates to double every 18 months. Other improvements on the network infrastructure, 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.

- The relevance of dynamic and encrypted content is increasing. Indeed, the Web is changing from a simple communication and browsing infrastructure for getting static information to a

complex medium for conducting personal and commercial transactions that require dynamic computation and secure communications with multiple servers through middleware and application software. A Web server that provides dynamic or secure content may incur a significant performance penalty. Indeed, the generation of dynamic content can consume significant CPU cycles with respect to the service of static content (e.g., [35]), while the management of data encryption which characterizes e-commerce applications can be orders of magnitude more expensive than the provisioning of insecure content [8]. The proliferation of heterogeneous client devices, the need of data personalization, client authentication, and system security of corporate data centers and e-commerce sites place additional computational load on Web servers. Indeed, it is often necessary to establish a direct communication between clients and content providers that caching infrastructures and content delivery networks cannot easily bypass. Caching is a very effective solution to reduce the burden on Web sites providing mainly static content, such as text, graphic, and video files, while it is less effective for applications that generate dynamic and personalized information, although there is some attempt to address this issue [2, 36, 94, 101, 130].

With the network bandwidth increasing about twice faster than the server capacity, the increased percentage of dynamic content of Web-based systems, the need of a direct communication channel between clients and content providers, the server side is likely to be the main future bottleneck.

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-server systems is to upgrade the Web server to a larger, faster machine. This strategy, referred to as *hardware scale-up* [53], 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 long-term nor a cost-effective solution, considering the step 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 improving the server's operating system (e.g., [16, 70, 91, 98]), building more efficient Web servers (e.g., [97, 127]), and implementing different scheduling policies of requests (e.g., [18, 49]). [98] have proposed a unified I/O buffering and caching system for general operating systems that supports zero-copy I/O. To improve the performance of the Apache Web server, [70] have proposed some techniques that reduce the number of system calls in the typical I/O path. [91] have analyzed how a general-purpose operating system and the network protocol stack can be improved to provide support for high-performing Web servers. As far as concerns proposals for more efficient Web server software, the Flash Web server [97] ensures that its threads and processes are never blocked by using an asymmetric multiprocess event-driven

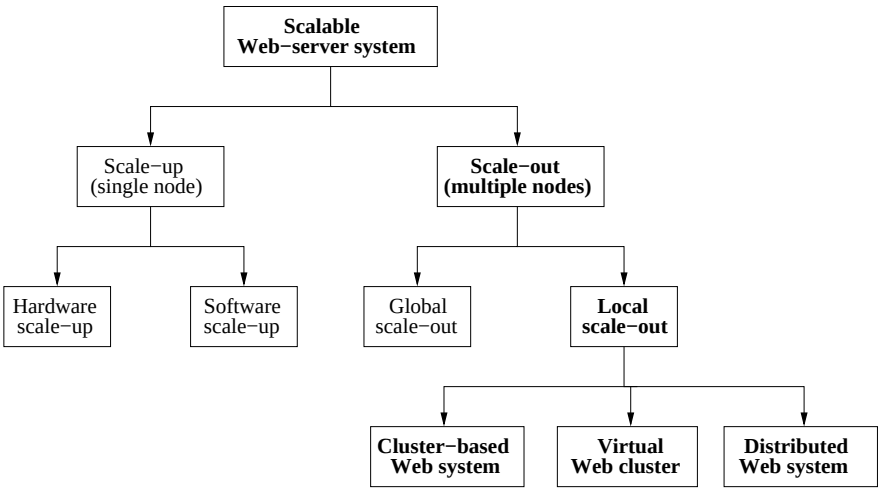

Figure 1: Architecture solutions for scalable Web-server systems.

architecture, while the Zeus Web server [127] uses a small number of single-threaded I/O processes, where each one is capable of handling thousands of simultaneous connections. Some literature has focused on new scheduling policies for client requests; for example, Bansal and Harchol-Balter [18] have proposed the use of the Shortest Remaining Processing Time first policy to reduce the queueing time at the server.

Improving the power of a single server will not solve the Web scalability problem in a foreseeable future. Another solution to keep up with ever increasing request load and provide scalable Web-based services is to deploy a distributed Web system composed by multiple server nodes that can also exploit scale-up advancements. The load reaching this Web site must be evenly distributed among the server nodes, so as to improve system performance. Therefore, any distributed Web system must include some component (under the control of the content provider) that routes client requests among the servers with the goal of load sharing maximization. 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* [53]. 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 Web-server 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-based 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

In this survey, 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* behind the client application.

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 system 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` as a name for the Web site.) This choice excludes from our analysis widely adopted solutions, such as the *mirrored-server* system that lets users manually select alternative names for a Web site, thereby violating the transparency requirement.

Unlike the users, it is admissible that the client application is aware of the effects of some mechanism used to dispatch requests among the multiple servers. However, in our survey we assume that the clients do not need any modification to interact with the scalable Web system.

From these premises, the basic Web system architecture considered in this survey consists of multiple server nodes, grouped in a local area with one or more mechanisms to spread client requests among the nodes and, if necessary, one or more internal routing devices. Each Web server can access all site information, independently of the degree of content replication. The Web system requires also one authoritative Domain Name System (DNS) server for translating the Web site name into one or more IP address(es). This name server typically belongs to a network different from the LAN segment used for the Web servers. 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 the system architecture for a modern Web site 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 in the back-end layer are outlined in Section 9.

We analyze now the main phases to serve a user request to a Web site. A user issues one request at a time for a *Web page* which is intended to be rendered to the user as a single unit. Nevertheless each user request (click) causes multiple client-server interactions. There are many types of Web pages, but typically a *Web page* is a multi-part document consisting of a collection of objects that may be a static or dynamically generated file. An object could be provided by the first contacted Web site or even by another site (e.g., banners). As in this brief description we cannot cover all possible instances, we focus on the most common example of Web page. It consists of a base HTML file describing the page layout and a number of objects referenced by the base HTML file hosted on the same Web site. A *static Web object* is a file in a specific format (e.g., an

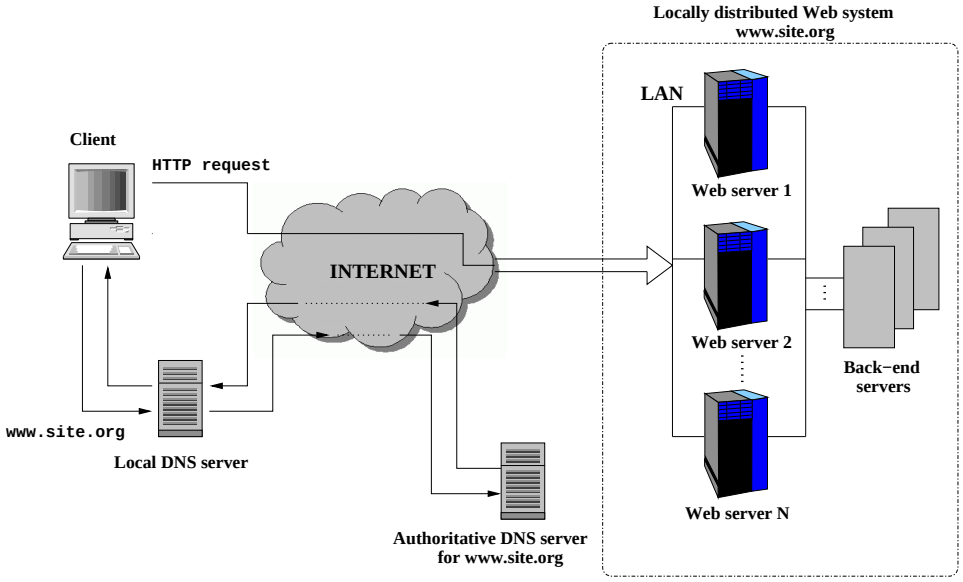

Figure 2: Model architecture for a locally distributed Web system.

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 *dynamic Web object* requires some computation on the server side and, possibly, some additional information from the user.

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 a URL or clicking on a link, for example `http://www.site.org/pub/index.html`. First, the client (browser) extracts the symbolic site name (`www.site.org`) from the requested URL and asks, through the resolver, 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 some intermediate name servers or a well-known root of the domain name server hierarchy, and ultimately querying the `site`'s authoritative name server. The resolver returns the obtained IP address to the client that can establish a TCP connection with the server or device corresponding to that IP address. After the completion of the TCP connection, the client sends the HTTP request (using the GET method) for `/pub/index.html` to the Web server that sends the requested object back. 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 (HTTP/1.1) or different TCP connections (HTTP/1.0) to the same server.

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 (that is, support or not for persistent TCP connections), the presence of the requested Web object either in the client local cache or in intermediate cache 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, that is, the authoritative DNS, the Web servers, and some internal devices of the Web system. On the other hand, we do not consider client-based routing mechanisms [15, 90, 119, 126] because they may require some modifications to the client software [30]. In addition, we do not investigate dispatching solutions at the network level that are 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 to improve 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 [84, 122]. In this paper, we consider some *internal caching* techniques occurring inside the Web-server system, such as Web server accelerators [36, 111, 112] and other techniques for improving cache hit rate [11, 96], while we exclude external caching solutions, such as proxy servers and cooperative proxies [20, 120, 122], virtual servers (or reverse proxies) [84], Web proxy accelerators [106].

In this survey, we also exclude solutions where the content provider delegates scalability for its Web-based 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 [4, 10, 40, 83, 123]. More recently, *Content Delivery Network* (CDN) organizations undertake to serve request traffic for Web sites from caching sites at various Internet borders [2, 55, 88, 63].

1.4 Organization of this paper

The rest of this paper is organized as follows.

- Section 2 discusses and classifies locally distributed architectures for Web sites, by distinguishing *cluster-based Web systems* or simply *Web clusters*, *virtual Web clusters*, and *distributed Web systems*.
- Sections 3, 4, and 5 cover the mechanisms that can be used to route requests in a Web cluster, in a virtual Web cluster, and in a distributed Web system, respectively.
- Section 6 describes 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 7 classifies various academic prototypes and commercial products according to the routing mechanism that is used to distribute the client requests among the servers in a Web cluster.
- Section 8 presents some extensions of the basic system architecture to improve scalability.
- Section 9 outlines the problem of Web content placement among multiple front-end and back-end servers, that is orthogonal to this paper.
- Section 10 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 the IP layer or not. Given a set of server nodes that host a Web site at a single location, we can identify three main classes of architectures:

- *Cluster-based Web system* (or *Web cluster*) 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 Web server layer.
- *Virtual Web cluster* where the VIP address is the only IP address visible to the clients like in a cluster-based Web system. Unlike the previous architecture, this VIP address is not assigned to a single front-end device but shared by all the server nodes.
- *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 other naming components. The cluster-based Web system architecture is a more recent solution where request routing is entirely carried out by the internal entities of the Web cluster. We can anticipate that, for a Web site where the server nodes are in the same location, a cluster-based system is preferable to a distributed system because the former architecture can provide fine-grained control on request assignment, and better availability and security that are key requirements for present and future Web-based services [69]. On the other hand, distributed Web systems with their visible IP-architecture are suitable to implement Web sites where the servers are geographically dispersed. These observations motivate the position of the survey that focuses on cluster-based Web systems, and considers distributed Web systems mainly for historical reasons.

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 three 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

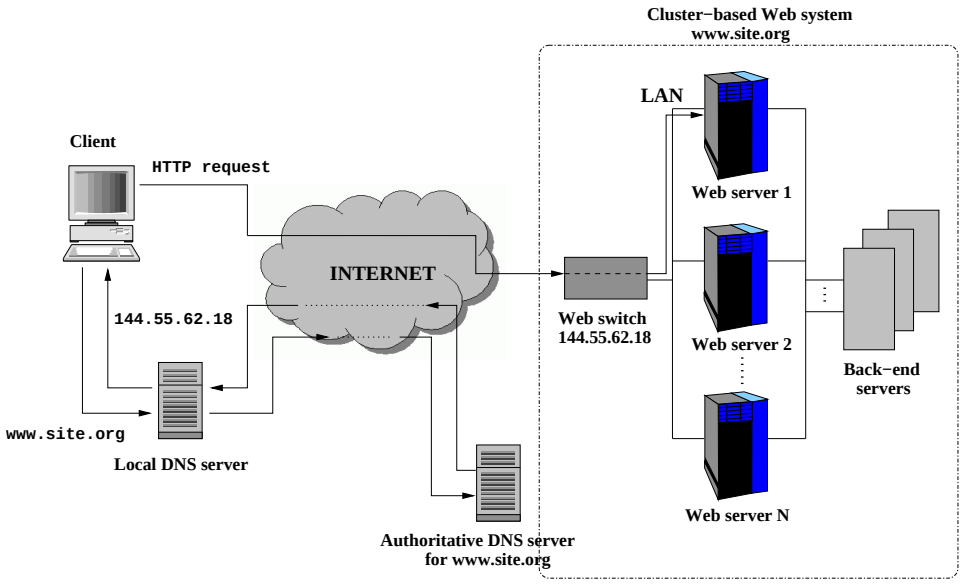

Figure 3: Architecture of a *cluster-based Web system.*

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 [53]. We prefer the term Web cluster, as Web farm is often 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. The authoritative DNS is not included in the system box because it does not have any role in client request routing.

The Web switch can be implemented on either special-purpose hardware devices plugged into the network (also called Application Specific Integrated Circuit chips) or software modules running

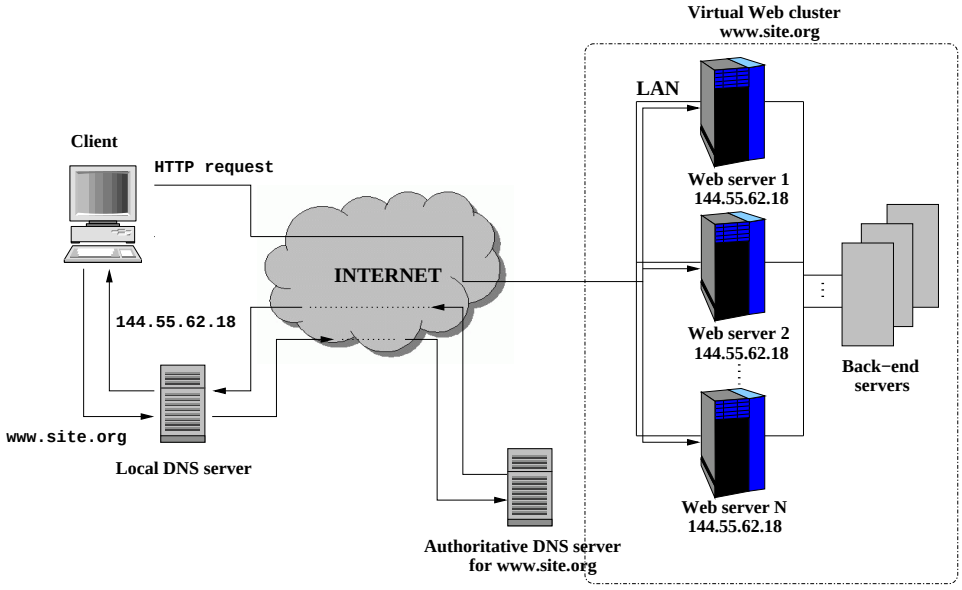

Figure 4: Architecture of a *virtual Web cluster*.

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 some literature reports.

2.2 Virtual Web clusters

The architecture of a virtual Web cluster is based on a fully distributed system design that does not use a front-end Web switch. Similarly to the previously described Web cluster architecture, a *virtual Web cluster* presents a single system image to the outside throughout the use of a single VIP address. The main difference is that this address is not assigned to a centralized front-end node that receives all incoming client requests and distributes them to a target server. In a virtual Web cluster, the VIP address is shared by all the server nodes in the cluster so that each node receives all inbound packets and filters them to decide whether to accept or discard them. The mechanisms are described in Section 3.3. A high-level view of a virtual Web cluster architecture is shown in Figure 4. It worth to note that this system removes the single point of failure and the potential bottleneck represented by the Web switch.

2.3 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

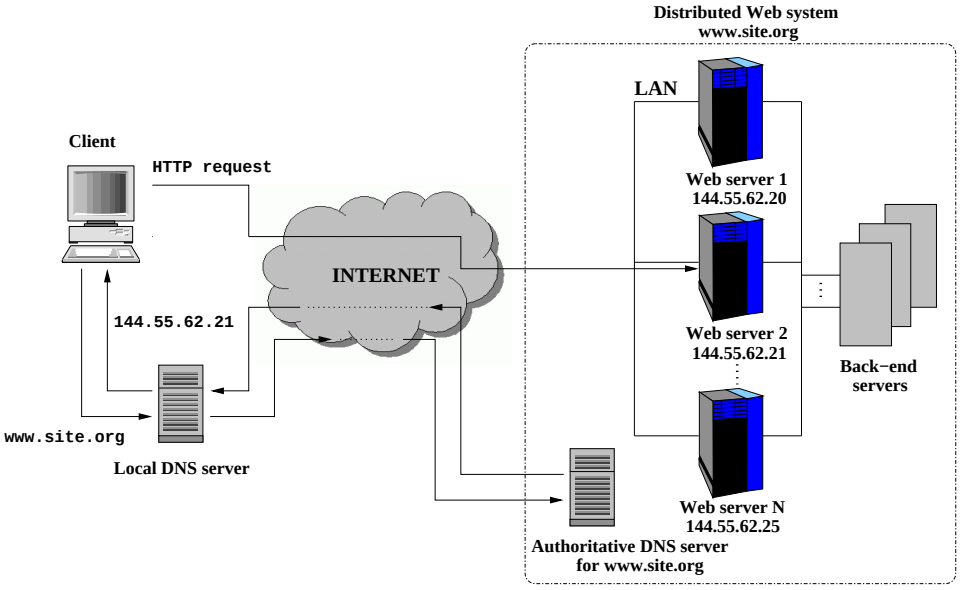

Figure 5: Architecture of a *distributed Web system*.

in Figure 5. (The authoritative DNS is included into the system box because this component plays a key role in request routing for distributed Web systems.) Unlike the cluster-based Web system, and similarly to the virtual Web cluster architecture, a distributed Web system does not rely on a front-end Web switch. 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 routing mechanisms in a distributed Web system are examined in Section 5.

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

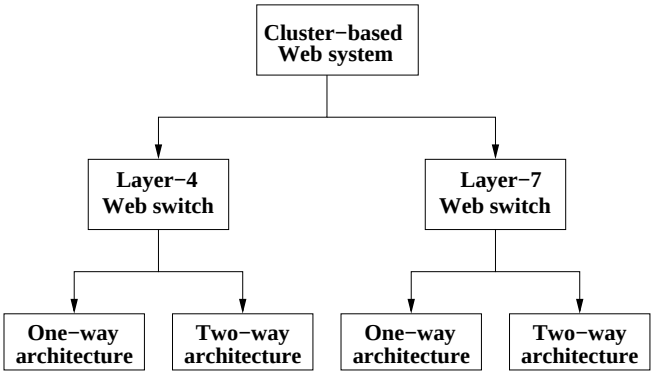

Figure 6: Taxonomy of cluster-based architectures.

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 either responds directly to the client (namely, *one-way* architectures) or returns its response to the Web switch, that in its turn sends the response back to the client (referred to as *two-way* architectures). Figure 6 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.

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

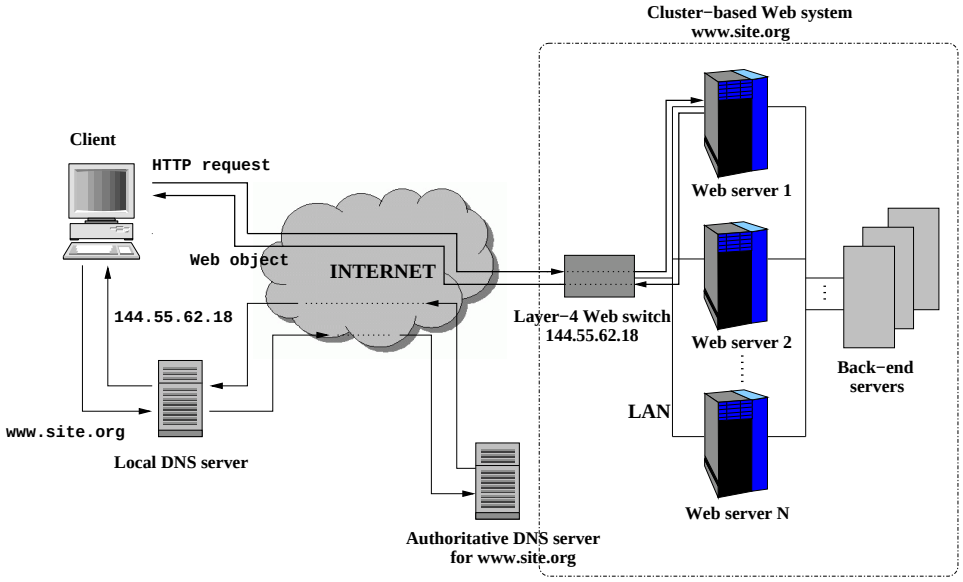

Figure 7: Layer-4 two-way architecture.

established one, or none of them. If the inbound packet is for a new connection (that is, 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 4-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. The client information in the 4-tuple is dynamic, whereas the server address/port is a static information which is known by the Web switch at the cluster initialization time.

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 server-to-client way. 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, that is, the private address is at IP level. Both inbound and outbound packets are rewritten at TCP/IP level by the Web switch. Figure 7 shows the packet flows.

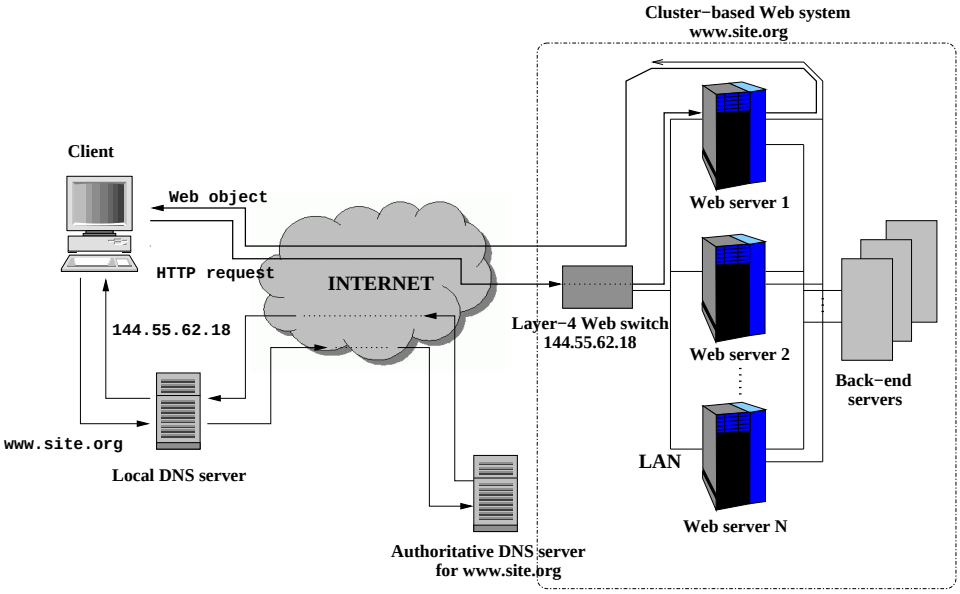

Figure 8: Layer-4 one-way architecture.

Packet rewriting is based on the IP Network Address Translation approach [114]. 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. 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.

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 outbound packets. Figure 8 shows an example where the outbound packets flow back to the client from the Web server 1. In this figure, the layer of the server private address is intentionally not specified as it can be either at the IP layer (layer-3) or MAC layer (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 the MAC layer (*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 the IP layer.

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 [54].

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 [100]. 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 datagram contains the VIP address and the server IP address as source and destination address, respectively. The server private addresses are at the IP layer 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 must have one of their network interfaces physically linked by an uninterrupted LAN segment. Unlike packet single-rewriting and packet tunneling mechanisms, the server private addresses are now at the MAC layer. (For this reason, packet forwarding is also referred to as *MAC address translation*.) The same virtual IP address is shared by the Web switch and the servers in the cluster through the use of primary and secondary IP addresses. In particular, each server is configured with the VIP address as its secondary address through a 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, to avoid collisions, the server nodes have disabled the Address Resolution Protocol (ARP) mechanism. The Web switch forwards an inbound packet to the target server by writing the server MAC address in the layer-2 destination address and retransmitting the frame on the common LAN segment. This operation does not require any modification of the TCP/IP header of the forwarded packet. Since all nodes share the same VIP address, the server receiving this packet can recognize itself as a destination and can respond directly to the client. (Packet forwarding is also known as *direct server return* [23].) Rewriting and encapsulation operations at TCP/IP level, which are typical of packet rewriting and tunneling mechanisms, are unnecessary.

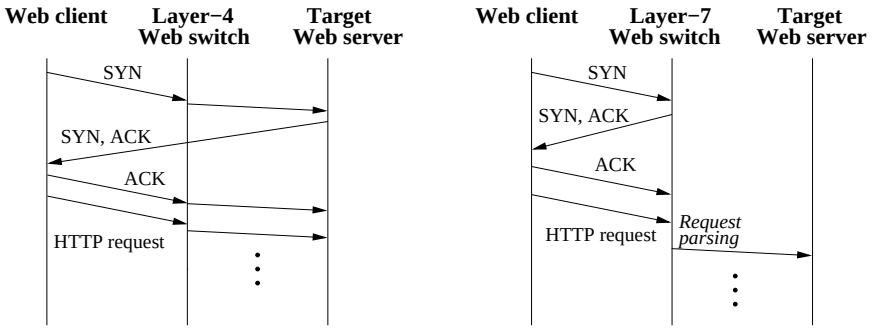

Figure 9: Operations of layer-4 routing (left) and layer-7 routing (right).

3.2 Solutions based on layer-7 switches

Layer-7 Web switches work at application layer, 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 (that is, 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 layer. 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 9 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.

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

TCP gateway In this architecture, a proxy running on the Web switch at the application layer mediates the communication between the client and the server. This proxy maintains open a TCP *persistent* connection with each Web server. When a request arrives, the proxy on the Web switch accepts the client connection and forwards the client request to the target server through the TCP persistent connection. When the response arrives from the server, the proxy forwards it to the client through the client-switch connection.

TCP splicing This mechanism aims to improve the TCP gateway approach that is computationally expensive because each request/response packet must flow up to the application layer.

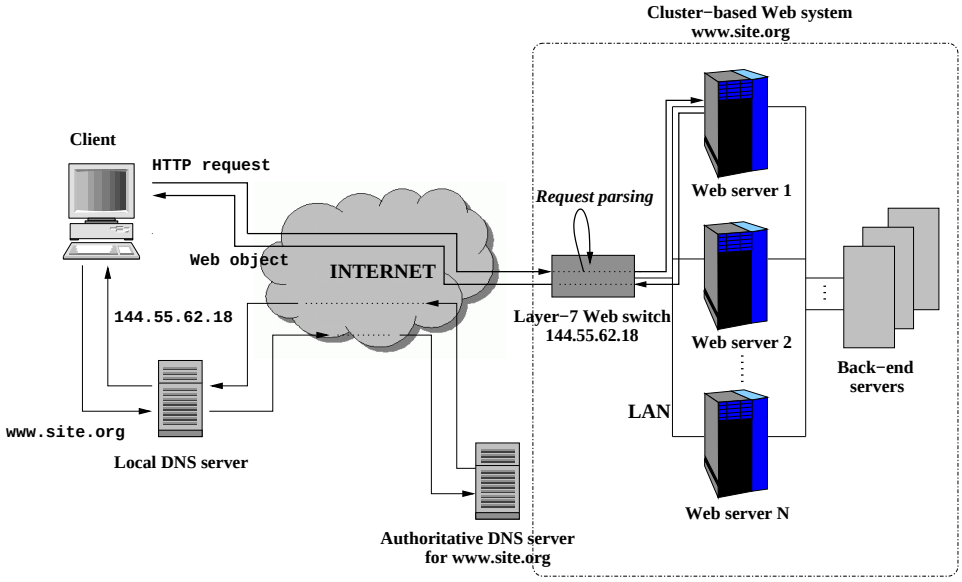

Figure 10: Layer-7 two-way architecture.

Similarly to TCP gateway, the Web switch maintains a persistent TCP connection with each Web server. Unlike the previous technique, TCP splicing forwards packets at the network layer between the network interface card and the TCP/IP stack. It is carried out directly by the operating system which must be modified at the kernel level [44, 85, 113].
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 (or patched) together. In such a way, IP packets are forwarded from one endpoint to the others without traversing the transport 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 (that is, it modifies the IP address and recalculates checksum) so that both the client and target server can recognize these packets as destined to them.
The TCP splicing mechanism can be also implemented by hardware-based switches (e.g., [7]) or by a library of functions defined at the socket level [107].

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 11.

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, which can communicate directly with the client [96].

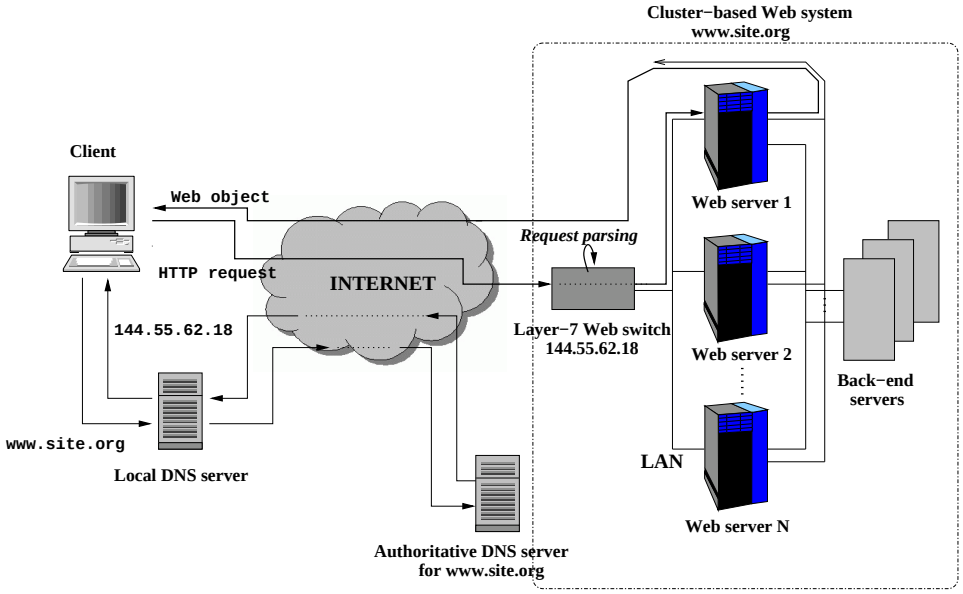

Figure 11: Layer-7 one-way architecture.

The TCP handoff mechanism remains transparent to the client, as packets sent by the servers appear to be coming from the Web switch. Incoming traffic on already established connections (that is, any acknowledgment packet sent by the client to the switch) is forwarded to the target server by an efficient module running at the bottom of the Web switch's protocol stack.

The TCP handoff mechanism requires modifications to the operating systems of both the Web switch and the servers. However, this is not a serious limitation because Web clusters are special solutions anyway. Moreover, TCP handoff allows to handle HTTP/1.1 persistent connections by letting the Web switch assign HTTP requests in the same connection to different target servers [11]. 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*). The integration of the TCP handoff mechanism with the SSL protocol to provide the reuse of SSL session identifiers is an active research area, but no result is yet available.

TCP connection hop. This a software-based proprietary solution implemented by Resonate as a TCP-based encapsulation protocol [104]. 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 a Resonate Exchange Protocol (RPX) packet and sending it to the server [104]. The connection hop operates at the network layer between the network interface card and the TCP/IP stack, thus minimizing the latency

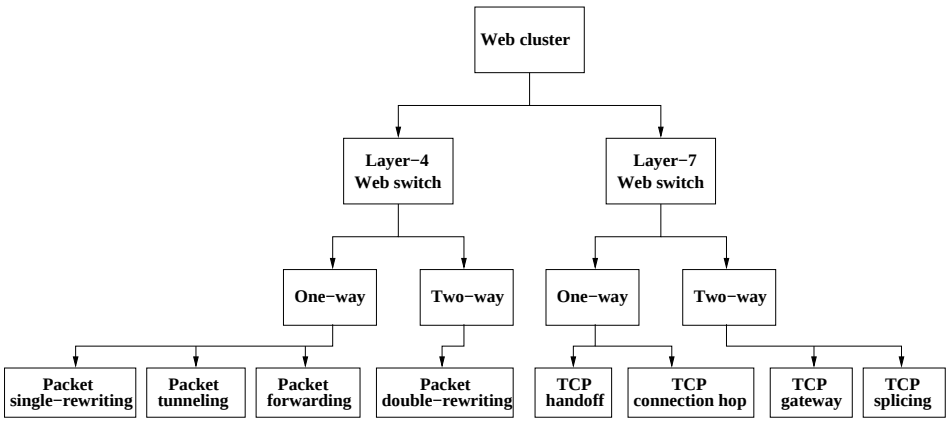

Figure 12: Detailed taxonomy of Web cluster architectures.

of incoming packets. The server can reply directly to the client because it shares the same VIP address of the Web switch. Acknowledgment packets and persistent session information from clients are managed by the Web switch.

3.3 A comparison of routing mechanisms for Web clusters

Figure 12 summarizes the classification for Web cluster by further detailing the taxonomy previously shown in Figure 6. We first discuss the different solutions for layer-4 and layer-7 routing, and then we compare the two classes of routing mechanisms.

3.3.1 Layer-4 routing mechanisms

An advantage of two-way solutions is that the server nodes may be in different LANs. The main constraint is that both inbound and outbound traffic must flow through the Web switch. The consequence 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. It is worth noting that the TCP header rewriting does not necessarily mean a recomputation of the entire packet checksum because the new checksum can be obtained by subtracting the old checksum [105].

A layer-4 Web switch that uses a one-way solution can sustain a larger throughput than two-way architecture 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.

This requires modifications to the kernel of the server operating system, but it does not seem to be a serious drawback because Web clusters are proprietary solutions anyway.

Packet forwarding mechanisms aim to limit the overhead of packet rewriting thanks to two mechanisms. The Web switch processes only inbound packets and it works at the MAC layer that avoids expensive checksum recalculations. A limitation 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. They require the servers to support IP tunneling which is not yet a standard for current operating systems. However, as for the other solutions based on special purpose systems, this is not considered a serious limit in the reality.

Thanks to hardware improvements and one-way mechanisms, the scalability of Web clusters based on layer-4 solutions is primarily limited by the capacity of the network link to Internet.

3.3.2 Layer-7 routing mechanisms

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 [12]. However, one-way solutions operating at layer-7 require modifications to the operating system of both the Web switch and the servers. Typically, two-way architectures work also on commercial off-the-shelf (COTS) systems when combined with layer-4 operations, while they may or not require special purpose operating systems when combined with layer-7 solutions (see for example, TCP splicing and TCP gateway, respectively). However, as already observed, the research community and the market feel acceptable to use special purpose solutions for Web clusters, especially if they guarantee better performance. An advantage of two-way architectures combined with a layer-7 Web switch is that caching solutions 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.

The main advantage of the TCP gateway approach is the simplicity of its implementation that can be done on any operating system. The main drawback is the overhead that this solution adds on both request and response packets flowing through the Web switch up to the application layer. TCP splicing reduces TCP gateway overhead, because it eliminates the expensive copying and context switching operations that result from the use of an application layer proxy. However, even in this instance, the Web switch can easily become the bottleneck of the cluster as it needs to modify the TCP and IP packet headers.

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 6.3 that through these policies it is possible to achieve high disk cache hit rates, to partition the Web content among

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

	Layer-4 two-way	**Layer-4** one-way	**Layer-7** two-way	**Layer-7** one-way
Dispatching	content-blind	content-blind	content-aware	content-aware
Dispatching granularity	TCP connection	TCP connection	HTTP request	HTTP request
Web switch data flow	in/outbound	inbound	in/outbound	inbound
Routing characteristic(s)	fast	fast	slowest (gateway) slow (splicing)	slow, complex
System scalability	high	highest	lowest	medium
Supported net applications	HTTP, FTP, ...	HTTP, FTP, ...	HTTP	HTTP

the servers, to employ specialized server nodes, to assign subsequent SSL sessions to the same server, and to achieve a fine grain request distribution 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 [12, 112]. As an example, [12] show 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 improve scalability of layer-7 architectures, alternative solutions for scalable Web-server systems, which combine content-blind and content-aware request distribution, have been proposed. They are described in Section 8.

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

4 Request routing mechanisms for virtual Web clusters

In this section, we analyze the mechanisms for routing requests in a virtual Web cluster, where the architecture does not use a front-end device with a single VIP address. As the name of the Web site is mapped by the DNS into the virtual IP address that is shared by all Web servers, the decision on client request routing is not designated to one entity but it is fully distributed among the nodes of the cluster. All inbound packets reach each Web server that has to filter out the packet to determine whether it is the target or not. Only one server must accept the packets from the same client and the others must refuse them. The packet filtering occurs between the data link layer and the IP layer. It is carried out either by a specific driver interposed between these two layers or by the modified device driver of the server.

The request routing is *content-blind*, because the target server identifies itself only by examining the information at TCP/IP level, such as the client IP address and port. Typically, the filter implemented on each server node computes a hash function on the source IP address and sometimes also on the port number. If the hash value matches the server own value, the packet is accepted, otherwise it is discarded.

Since the same VIP address is shared by all the server nodes, request routing occurs at the MAC layer. Two MAC address assignments to the server nodes are feasible: *unicast MAC address* and

multicast MAC address that are described below.

Request routing at the MAC layer allows a virtual Web cluster to host any IP based service. Another advantage of this architecture is that it avoids the single point of failure and the potential system bottleneck represented by the Web switch. However, it has its own disadvantages too. The MAC routing mechanism, similarly to the packet forwarding mechanism (see Section 3.1), requires all the server nodes to be on the same subnet. But the most serious limitation of the virtual Web cluster architecture is represented by the request dispatching. Not only the request routing in a virtual Web cluster cannot take advantage of content-aware dispatching, but also the packet filtering based on a hash function is not able to adapt itself to dynamic conditions when the client requests unevenly load the servers. In this survey, we will not further investigate the dispatching policies for virtual Web clusters because they are limited to the application of a hash function.

4.1 Unicast MAC address

The unicast MAC address approach requires the assignment of a unique MAC address (namely, *cluster MAC address*) to all the Web server nodes in the cluster [87, 117]. To let this mechanism operate, we must change the unique burned-in MAC address of the network adapter of each server into the cluster MAC address. The frames addressed to the cluster MAC address are received by each server node, which filters out the incoming packets to determine whether it is the intended recipient.

Since the servers share both the same IP address and the same MAC address, intra-cluster network communications are impossible unless a second network adapter is used for each node. In this case, the server dedicated IP address is mapped to the MAC address of the second adapter.

The unicast MAC address approach is the default mode of operation of the Network Load Balancing [87], and is also used in the prototype realized by [117]. The Network Load Balancing works as a filter between the network adapter and the TCP/IP protocol stack. It applies a hash function on the source IP address and the port number, and passes to the upper network layer only the packets that are destined to the local node. The cluster nodes can periodically exchange messages to maintain a coherent view of the mapping and for reliability monitoring, even if this requires to add a second network adapter to each server.

4.2 Multicast MAC address

The alternative mode of operation of the Network Load Balancing [87] relies on the use of a multicast MAC address. In this architecture, all the server nodes are configured to receive Ethernet frames via an identical multicast MAC address [87]. The unicast VIP address of the Web site is resolved to the multicast MAC address. When the access router sends an ARP request to find out the MAC address of the VIP address, the system returns the MAC multicast address, and the frames addressed to this address are received by all the servers. Each node inspects the client packet and one of the servers chooses to reply to the request based on some filtering mechanism. Typically, the filtering mechanism is based on a hash function applied to the client IP address and the port

in the TCP/IP packet. Unlike the solution adopted for the unique MAC address mechanism, here each server retains its unique built-in MAC address. This is resolved to the specific IP address of the server that is used for intra-cluster traffic.

The multicast MAC address mechanism allows the use of one network adapter to handle both client-to-cluster traffic and server's dedicated traffic. However, the use of a multicast MAC address as a response to an ARP query for a unicast IP address may not be accepted by some routers (e.g., Cisco routers). This problem can be solved by a manual configuration of a static ARP entry within the router.

5 Request routing mechanisms for distributed Web systems

In this section, we analyze the mechanisms for routing requests in distributed Web systems where server IP addresses are visible to the clients because a front-end Web switch is not used. The decision on client request routing occurs at the DNS and/or server level. We analyze in Section 5.1 DNS-based mechanisms where the routing takes place during the address resolution phase, and in Section 5.2 (re-)routing mechanisms that are carried out by the Web servers. A DNS-based 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 distributed and used for the second-level routing or re-routing.

5.1 DNS routing mechanisms

DNS-based routing is the first solution to handle multiple Web servers hosting a Web site. Proposed around 1994, it was originally conceived for locally distributed Web systems even if now it is commonly used in geographically distributed Web systems [76]. 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 [25]. 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 hostname-address mapping that is typically cached in the name servers along the path from the A-DNS to the local name server 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 [46, 54]. (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 may 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 browser caching. 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 the network level and allow for a more fine-grained 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 [108]. Finally, the TTL period chosen by the A-DNS does not work on browser 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.

5.2 Web server routing mechanisms

Some routing mechanisms can be implemented also by the Web servers that can decide to (re)direct a client request to another node. Specifically, we consider the *triangulation* mechanism implemented at the TCP/IP layer, and *HTTP redirection* and *URL rewriting* mechanisms that work at the application layer. Unlike the previous routing mechanisms that typically have a centralized activation, Web server mechanisms are distributed. Indeed, when a new request arrives from the Web switch, each Web server can decide to serve it locally or to redirect it, although some centralized coordination can be used.

5.2.1 Triangulation

The triangulation mechanism is based on packet tunneling [14], which has been described in Section 3.1. When this routing is activated, 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 layer, by encapsulating the original datagram into a new datagram. The target node recognizes that the datagram has been re-routed and responds directly to the client (see one-way mechanisms in Section 3). 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.

5.2.2 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 [21, 60]. 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 13 shows the flow of requests and response when HTTP redirection is activated.

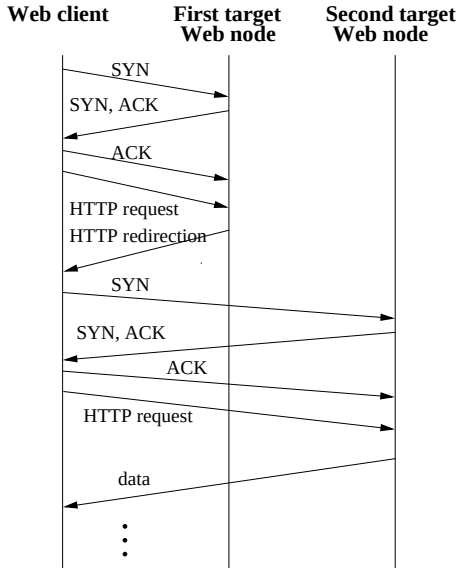

Figure 13: HTTP redirection.

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, because the first server receiving the HTTP request can take into account the content of the request when it selects another appropriate node.

The main drawback is that this mechanism consumes resources of the first contacted server and 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. Nevertheless, it is not automatic that a redirected page experiences a slower response time, because 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 yet redirection as specified by the HTTP standard, for example, when requiring the browser to display the originally requested URL instead of the redirected URL. Most browsers still 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.

5.2.3 URL rewriting

URL rewriting is a more recent mechanism to implement Web server re-routing. When redirection is activated, the first contacted node changes dynamically the links for the embedded objects within the requested Web page so that they point to another node [79]. Such redirection mechanism integrated with a multiple-level DNS routing technique is also used by some Content Delivery Networks, such as [2], [88], and [55].

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 IP address of the second target node. It has been demonstrated that address lookup might take substantially longer than network round-trip time [45]. On the other hand, URL rewriting has also some advantages over the other solutions. For example, this approach can be handled completely at the user level, and the redirection mechanism can redirect further communications of a client to a specific Web server with one request redirection only.

5.3 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 more fine-grained request distribution, but it could make the A-DNS a potential system bottleneck and increase the latency time perceived by users. Because of the presence of non-cooperative name servers that use their own lower bounds for the TTL, the range of the applicability of this solution is limited or not well predictable.

Although initially conceived for locally distributed architectures (e.g., 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 DNS problems is to add a second-level routing carried out by the Web servers through a re-routing mechanism operating at the TCP or HTTP layer. 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 an HTTP/1.1 persistent connection.

Unlike triangulation-based solutions, application-layer mechanisms, such as HTTP redirection

and URL rewriting, do not require the modification of packets reaching or leaving the Web-server system. This allows the server 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-based service to HTTP requests only and increases network traffic, because any redirected request needs two TCP connections prior to be serviced. There is another tradeoff when comparing URL rewriting and HTTP redirection for multiple requests to the same Web site. The former mechanism requires additional DNS lookups but no request redirection after the first one, while the latter solution might save address lookup overheads at the price of multiple request redirections.

Application-layer mechanisms can avoid ping-pong effects that occur when an already re-routed request is further selected for reassignment. A cookie can be 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. Besides the use of cookies, other means for session identification can be used (e.g., session encoding in URLs), either to identify browsers that do not support cookies or to avoid the misuse of long-living cookies. The triangulation mechanism does not suffer from these side effects, because the destination node can deduce if the request has already been re-routed by simply inspecting the source packet address.

Table 2 outlines and compares the main characteristics of the routing mechanisms for distributed Web systems.

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

	DNS	**Triangulation**	**HTTP redirection**	**URL rewriting**
Dispatching	content-blind	content-blind	content-aware	content-aware
Dispatching granularity	session	TCP connection	page/object	page/object
Client/server data flow	direct	triangular	redirection	redirection
Overhead(s)	None	operations at the first contacted server	round-trip times	server operations address lookups
Supported net applications	HTTP, FTP, ...	HTTP, FTP, ...	HTTP	HTTP

6 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 influences both performance experienced by the users and system scalability.

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 different criteria [34, 109, 110, 121]. 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

[1] We use the definition of *global scheduling* given in [34] and *dispatching* as synonymous.

can be actually applied to Web cluster dispatching. 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 objective of load sharing is to smooth out transient peak overload periods while load balancing algorithms strive to equalize the loads of the servers [75, 110], a dispatching algorithm at the 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 take immediate decisions 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 constraints for dispatching algorithms that we will analyze in this section.

1. Low computational complexity, because dispatching decisions are required in real-time.
2. Full compatibility with existing Web standards and protocols.
3. All state information needed by a dispatching policy has to be accessible to 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.

6.1 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 investigated in literature of distributed systems 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 [110]. 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 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 the TCP/IP layer, and *content-aware dispatching*, if the switch works at the application layer.

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

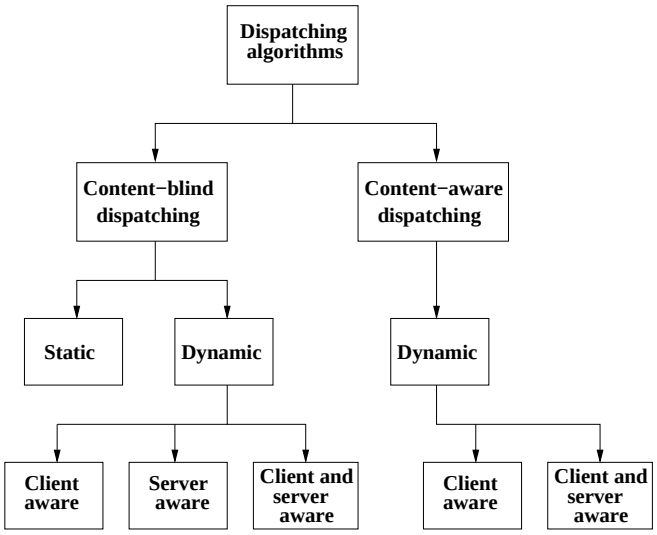

Figure 14: Taxonomy of dispatching policies in Web clusters.

at the TCP/IP layer. Indeed, 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-layer* 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 consider information about 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 14 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 the TCP/IP layer.

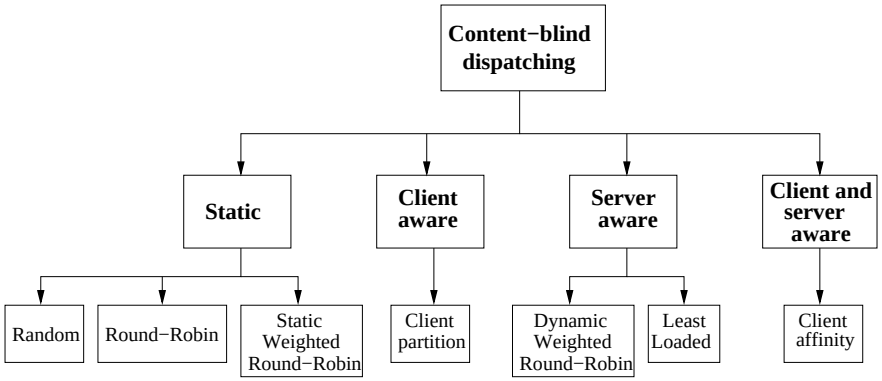

Figure 15: Content-blind dispatching algorithms.

6.2 Content-blind dispatching policies

In this section, we describe the main content-blind dispatching policies according to the taxonomy shown in Figure 14, and detailed in Figure 15 with some representative algorithms for each category at the bottom level.

6.2.1 Static algorithms

Static policies do not consider any system state information. Typical examples are **Random** and **Round-Robin** algorithms. Random distributes the incoming requests uniformly through the server nodes with equal probability of reaching any server. Round-Robin 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, Round-Robin utilizes only information on past assignment decision.

Both Random and Round-Robin policies can be easily extended to treat servers with different processing capacities by making the assignment probabilistic on the basis of the server capacity [46]. 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 Round-Robin 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.

Different processing capacities can be also treated by using the so-called **static Weighted Round-Robin**, 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 [80]. 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 $S_1\, S_1\, S_2\, S_1\, S_2\, S_3$.

6.2.2 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, that is, the IP source address and TCP port numbers. This coarse client information can be used by dispatching algorithms that statically partition the server nodes and assign the same clients to the same servers, typically through a hash function applied to the client IP address.

6.2.3 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 [50, 59]. The note in Section 6.4.2 is devoted to the analysis of some solutions 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 **Least Loaded** approach denotes a class of policies that depend on the chosen server load index. For example, in the **Least Connections** algorithm, which is usually adopted in commercial products (e.g., LocalDirector [42], BIG-IP [58]) 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 [80]. 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. The **Fastest Response Time** policy (e.g., LocalDirector [42] and ServerIron [61]) is another example of least loaded policy, where the Web switch assigns the new connection to the server which is responding fastest, that is, showing the smallest object latency time in the last observation interval.

The **dynamic Weighted Round-Robin** algorithm is a variation of the version that considers static information (server capacity) as a weight. This policy associates each server with a dynamically evaluated weight that is proportional to the server load state [72]. Periodically, the Web switch gathers load index information from the servers and computes the weights, that are dynamically incremented for each new connection assignment. The reader should be aware that the Weighted Round-Robin algorithm (WRR) is often referred in literature without specifying whether it is the static or dynamic version.

6.2.4 Client and server state aware algorithms

Server state information can be combined with some client information available at a layer-4 Web switch (e.g., the source IP address and the service port number within the TCP header). Performance or functional reasons motivate the choice of assigning consecutive multiple connections from the same client to the same server, to provide the so called *client affinity* [72, 80].

For example, to avoid time- and resource-consuming operations for SSL key negotiation and generation, it would be convenient to assign consecutive SSL connections from the same client to the same server during the life span of the SSL key. Indeed, the reuse of SSL keys cached in the server's main memory can significantly reduce the latency time [66]. An FTP session is an example where client affinity assignment is not a performance but a functional requirement, because this protocol uses two connections (control and data) that must be established with the same server.

As regards the reuse of cached SSL keys, it has to be noted that a layer-4 Web switch can only identify a client issuing requests on a secure channel through its source IP address. As different clients behind the same proxy or firewall share the same IP address, using only this information it can happen that clients which are not part of the session are routed to the same server.

In proposed client affinity policies, client information usually overrides server information for assignment decisions. This means that client past assignments have more importance than server state conditions.

6.2.5 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. To limit these risks, some authors have proposed the integration of a static algorithm at the Web switch with a second-level re-routing mechanism carried out by the server nodes. The server dispatching algorithm is typically content-aware and aims to improve load sharing and caching [31, 38, 41].

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 6.4.2). 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 [50, 89], yet the least loaded approach is commonly used in commercial products. On the other hand, many experiments and simulation results have demonstrated that the dynamic Weighted Round-Robin policy compromises simplicity with efficacy at best [33, 72].

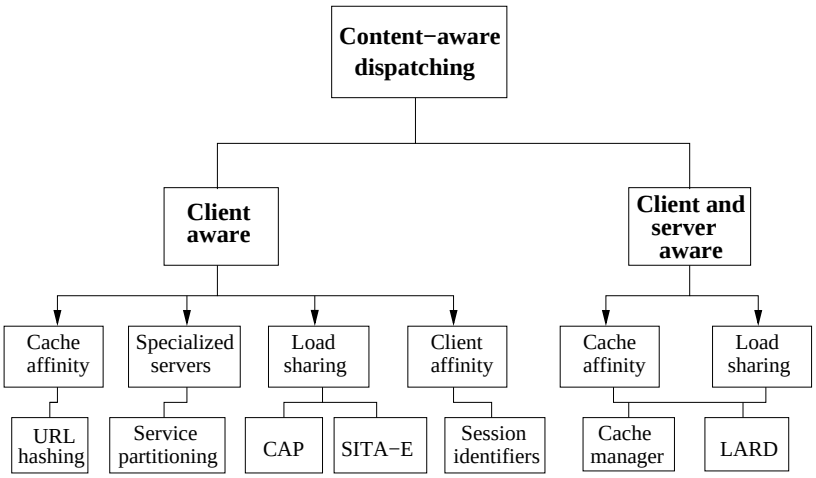

Figure 16: Content-aware dispatching algorithms.

6.3 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 14 with an additional level that considers the main goal of the dispatching policies. Figure 16 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-based 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 the TCP/IP layer (although with a lower degree of accuracy as explained in Section 6.2). Indeed, stateful services can be identified through the service port (e.g., 443 for SSL) and a connection reuse timeout can be set [72].

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.

6.3.1 Client state aware algorithms

Let us first consider the left part of the taxonomy in Figure 16. In cache affinity policies, the file space is typically partitioned among the server nodes. A hash function applied to the URL (or to a substring of the URL) can be used to perform a static partitioning of the files. The dispatching policy running on the Web switch (namely, **URL hashing** algorithm) uses the same function. This scheme exploits the locality of references in the server nodes and achieves the best cache hit rate. However, the solution combining Web object partitioning and hash function work well for 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., [9, 48]), the server nodes serving those critical files will be more loaded than others. These critical load peaks can be lowered by architectures that integrate proxy server solutions, but the other problem remains.

For Web sites providing heterogeneous Web-based 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 [125]. We refer to this policy as to **Service Partitioning**. Most commercial content-aware switches deploy this type of approach (for example, BIG-IP [58], Central Dispatch [104]).

The third main goal of the content-aware dispatching algorithms is to improve load sharing among the servers. These strategies do not require static partitioning of the file space and the Web-based services. Two policies belong to this class: **Size Interval Task Assignment with Equal load** (SITA-E) [68] and **Client-Aware Policy** (CAP) [33]. The former is more oriented to Web sites providing static information, the latter to sites providing Web-based services with different computational impact on system resources.

The **SITA-E** policy partitions dynamically the Web content among the servers according to the file size distribution. The Web switch selects the target server on the basis of the size of the requested file. The goal is to separate services for light tasks from those for heavy tasks [68]. The SITA-E policy founds on theoretical demonstrations, but it assumes that the service time of a request is proportional to its size. This assumption is valid for static Web content only (indeed, pre-determining the service time of a dynamic request remains an interesting open problem). Furthermore, the SITA-E policy does not consider that caching and possible optimizations at the kernel and TCP connection level (e.g., `sendfile` in Unix/Linux operating systems) have a stronger effect on small files.

The other dispatching policies, which do not consider static files only, typically manage hetero-

geneous services through a static partitioning of the Web content. A quite different approach is taken by the Client-Aware Policy (**CAP**) proposed in [33]. The basic observation is that when the Web site provides heterogeneous services, each client request could stress a different Web system resource, e.g., CPU, disk, network. Although the Web switch cannot estimate the service time of a static or dynamic request accurately, it can distinguish the class of the request from the URL and estimate its impact on main Web system resources. A feasible classification for CAP is to consider disk bound, CPU bound, and network bound services, but other choices are possible depending on placement of the Web content. To improve load sharing in Web clusters that provide multiple services, the Web switch manages a circular list of server assignments for each class of Web-based services. The CAP goal is to share multiple load classes among all servers so that no component of a node is overloaded. CAP in its initial form is a client aware policy, however it can be easily combined with some server state information.

Client affinity policies that assign all Web transactions from the same client to the same server can use more detailed information than that discussed in 6.2.2 when they are applied to a layer-7 Web switch. For example, **session identifiers**, such as cookies and SSL identifiers, are commonly exploited by current dispatchers. These 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.

6.3.2 Client and server state aware algorithms

Dispatching algorithms implemented at the application layer 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 16, the client and server aware policies belong to both classes of goals 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 [11, 96]. 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 in the HACC cluster [128].

In LARD it is the Web switch that maintains the mapping from a file to the set of nodes containing it, while the **Cache manager** dispatching policy relies on a cache manager that is aware of cache content of all Web servers [26]. 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 caching the object, provided that its

load is within a threshold over the least loaded server [26].

6.3.3 Considerations on content-aware dispatching

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 6.4.2). 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.

Static partitioning algorithms tend to ignore load sharing, as it is almost impossible to partition the Web-based services in such a way that the requests are balanced out. Indeed, a well-known characteristic of Web workload is that a small set of files account for a large fraction of requests. 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 services, content-aware policies, such as CAP, that aim to share the load among all (or most) of server components, can provide best performance [32].

6.4 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.

6.4.1 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 dynamic Weighted Round-Robin strategy when considering static content [11, 33].

On the other hand, operations at layer-7 are expensive, hence client state aware policies must limit 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 require many TCP segments. If the Web switch has to inspect every cookie before assigning the client request to a server, the latency time increases and the Web switch can easily become the system bottleneck.

It is important that new content-aware dispatching algorithms consider also the heterogeneity of Web-based 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.

6.4.2 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, memory, 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, that is, the completion time of an object request at the Web cluster side. Unlike load indexes such as utilization referring to a well known range, most other indexes must be referred to the system capacity under examination.

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 client 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 lifetime of a connection and the number of active connections, that provide 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 (for example, for the Fastest Response Time policy), number of active connections, and 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 uti-

lization, always need a process monitor on the servers and a communication mechanism from the servers to the Web switch.

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 strategy for interpreting stale load information that can be applied to layer-4 Web switches has been proposed in [50].

7 Classification of products and prototypes

In the last years, the size of the market for Web-server systems has rapidly expanded. Many companies, from big ones to startups, have invested large amounts of money for the research in cluster-based Web systems. The consequence is that new ideas and interesting solutions that contributed to the advancement of the research in this field came from both the academic and the industrial world.

For these reasons, we found correct and useful to classify together and compare research prototypes and commercial products concerning cluster-based Web systems.

In this section, we refer to the architecture taxonomy outlined in Section 3 and illustrated in Figure 12. We can anticipate that commercial products often use simple dispatching policies because they are more stable even if they do not guarantee best performance. On the other hand, research prototypes tend to propose and investigate more sophisticated solutions. However, this is just a tendency, and counter-examples exist.

Another premise is important before reading this section. The names of the companies and products tend to change frequently because of continuous merges and acquisitions in this field. For example, in October 1999 [74] acquired IPivot, a technology spin-off; in June 2000 Cisco Systems Cisco acquired ArrowPoint, one of the first companies to commercialize layer-7 Web switches; in January 2001 [93] entered content-aware dispatching market by acquiring Alteon WebSystems that was one of the market leaders. Because of this turbolence, we will maintain a Web page[2] with updated information about Web cluster prototypes and products.

7.1 Products based on a layer-4 Web switch

Table 3 classifies some commercial products and research prototypes that work at the TCP/IP layer. Some products, such as LVS [80], appear in multiple table entries because they can be configured to support more than one request routing mechanism.

[2] `http://weblab.ce.uniroma2.it/webcluster/links.html`

Table 3: Layer-4 Web clusters.

Two-way	**One-way**		
Packet double-rewriting	*Packet single-rewriting*	*Packet tunneling*	*Packet forwarding*
LocalDirector [42]	TCP Router [54]	LVS [80]	Network Dispatcher [72, 73]
Magicrouter [5]			LVS [80]
LVS [80]			BIG-IP [58]
LSNAT [115]			LSMAC [64]
BIG-IP [58]			NetStructure Traffic Director [74]
ServerIron [61]			Alteon 180 [93]
Network Dispatcher [73]			WSD Pro [103]
Equalizer [47]			ServerIron [61]
NetBalancer [3]			ONE-IP [51]

We compare some prototypes and products based on a layer-4 Web switch by first considering two-way architectures and then one-way architectures.

7.1.1 Two-way solutions

Layer-4 Web switches based on double-packet rewriting are often implemented on specialized hardware because of their relative simplicity. Examples are LocalDirector [42] and ServerIron [61]. However, pure-software implementations are also possible, such as LVS [80] and the Magicrouter prototype from Berkeley [5].

Two-way architectures typically offer various solutions for the dispatching policies implemented at the Web switch, ranging from static algorithms to client and server state aware algorithms. For example, LocalDirector [42] is provided with the *least connections* policy that selects the server with the least number of active connections, and the *Fastest Response Time* algorithm that dispatches the request to the server that responded as the fastest to previous connection requests. Magicrouter [5], besides the stateless round-robin and random algorithms, offers also the *incremental load* policy, which 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.

Two-way architectures (e.g., LocalDirector [42] and ServerIron [61]) can also support some stateful services, such as SSL, through the use of a sticky flag. This is achieved by directing multiple connections from the same client to the same server within a period of time that is set to 5 minutes by default. However, SSL session reuse provided by layer-4 Web switches may lead to load imbalance among the server nodes because these switches are not able to distinguish among different clients that are behind the same proxy or firewall.

7.1.2 One-way solutions

Most existing products and prototypes based on one-way architectures provide request routing through the *packet forwarding* mechanism. Among the others, we recall the Network Dispatcher [72, 73] which is the load balancing component of the IBM WebSphere Edge Server, and ONE-IP from Bell Labs [51] which is one of the first implementations of a layer-4 Web switch based on packet forwarding. In these systems, the switch forwards packets destined to the Web cluster to a selected server by using its MAC address on the LAN, without modifying the TCP/IP headers. This is possible because all the server nodes share the same VIP address which is bound to the loopback interface (for example, through the `ifconfig` command in Unix/Linux systems). It is important to observe that some commercial products define the packet forwarding mechanism through different names. This is the case of nPath [58] and SwitchBack [61].

A small subset of solutions use request routing mechanisms that differ from packet forwarding. For example, the TCP router [54] implements packet single-rewriting which allows the Web servers to respond directly to clients by modifying outbound packets, while one of the three request routing mechanism supported by LVS [80] is based on IP packet tunneling.

Similarly to two-way architectures, one-way solutions offer many alternatives for the dispatching policies implemented at the Web switch. As explained in Section 6.4.2, the main difference between the two architectures is in the server state information directly available at the Web switch. For example, the Network Dispatcher uses a dynamic Weighted Round-Robin algorithm to distribute connections among the server nodes [72]. ONE-IP supports two different system architectures, called *routing-based dispatching* and *broadcast-based dispatching* [51]. In both solutions, the destination server is selected by applying a hash function that maps the client IP address into a server identifier. The difference between the two solutions lies in the cluster component that applies the hash function. In routing-based dispatching, the Web switch selects the target server using the hash function, while in broadcast-based dispatching the Web switch broadcasts the packets destined to the VIP address to every server in the cluster; each server evaluates whether it is the actual destination of the packets by applying the hash function. The main advantage of the ONE-IP approach is that the Web switch does not need to keep track of any system state information. The weak point of the ONE-IP approach is the use of a hash function to select the server, based on the client IP address.

One-way solutions are also able to support stateful services, such as SSL session reuse, although the same limitation of two-way architectures applies. For example, the Network Dispatcher provides a *client affinity* mechanism that is similar to Cisco's sticky flag.

7.2 Products based on a layer-7 Web switch

In Table 4 we classify the cluster-based Web systems that work at the application layer. Some products listed herein, such as ServerIron [61] and BIG-IP [58], have already been considered in Table 3 as they can be configured to support both layer-4 and layer-7 routing mechanisms.

Table 4: Layer-7 Web clusters.

Two-way		**One-way**	
TCP gateway	*TCP splicing*	*TCP handoff*	*TCP connection hop*
Network Dispatcher proxy-level CBR [73]	Alteon Web OS [93]	ScalaServer [11] [96]	Central Dispatch [104]
ClubWeb [33]	Web Switch [81]	[116]	
HACC [128]	CSS [42]	ClubWeb [6]	
	ServerIron [61]		
	BIG-IP [58]		
	WSD Pro [103]		
	Load Balancer [127]		
	L5 [7]		
	[125]		
	Array 500 [13]		
	Network Dispatcher kernel-level CBR [73]		

7.2.1 Two-way solutions

Request routing in two-way architectures can be implemented in hardware (e.g., CSS [42], L5 [7], Alteon Web OS [93], and WSD Pro [103]), completely in software (e.g., the Network Dispatcher CBR product [73], the Harvard Array of Cheap Computers (HACC) [128] and the CLUster-Based Web (ClubWeb) prototypes [33]), or through a hardware switch combined with some control functions implemented in software (for example, BIG-IP [58] consists of a hardware box with a modified BSDi-Unix kernel, while Web Switch [81] is a Linux-based platform).

The request routing mechanism can be implemented at the application layer by configuring the Web switch to work as an enhanced TCP gateway or reverse proxy (e.g., Network Dispatcher proxy-level CBR [73], HACC [128], and ClubWeb [33]). An alternative solution is to route client requests at a lower layer by letting the Web switch splice the TCP connection established with the client to a connection with a selected server. Alteon Web OS [93], BIG-IP [58], and Web Switch [44, 81] use the TCP splicing mechanism. To this purpose, the Web switch 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. In Web Switch [81] two components implement the switching functionality on the Web switch. The first component is a user-level proxy that accepts connections from clients, parses the HTTP requests, and selects a target server. The proxy then removes itself from the data path by requesting a kernel module to splice the TCP connections together. To limit the bottleneck risks at the Web switch, the splicing functions can be implemented on specialized

hardware. For example, Alteon Web OS [93] relies on a dedicated network multi-processor integrated with a parallel-processing operating environment.

As regards the dispatching algorithms in two-way layer-7 Web switches, the most sophisticated policies are typically implemented in research prototypes. For example, in the HACC prototype [128] the target server node is selected on the basis of the locality properties and capacity of the nodes. The Web switch monitors the server loads and 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. To improve reference locality in disk cache, subsequent requests for the same object are assigned to the same server. In the ClubWeb prototype [33], the Web switch selects the target server on the basis of the CAP policy described in Section 6.3.

Commercial products typically parse the HTTP request either to partition the servers according to the service type they handle or to provide persistent session support, based on cookies or SSL identifiers. Typically, each commercial product provides a set of content-aware dispatching policies. The companies use different names, but the substance is similar. Let us give some examples. In Alteon Web OS [93] the content-aware policy is basically a service partitioning algorithm that allows specialized servers to store specific object types or provide specific services. Hence, the client request is assigned by the Web switch to a target server that is selected on the basis of URL (or URL substring) matches. ServerIron [61] supports URL hashing, in which the Web switch examines information in the HTTP request and maps this information to one of the servers. All the HTTP requests that contain the same information are always assigned to the same server. BIG-IP [58] can use cookie information stored in an HTTP request to direct requests from the same client to the same server. A more sophisticated algorithm that resembles the LARD policy [96] and aims at improving the locality of reference in request streams runs on the Load Balancer [127].

7.2.2 One-way solutions

One-way solutions at layer-7 are typically software implementations that require modifications at the operating system level on both the Web switch and the servers. These interventions can be done in the internal kernel. An alternative is to realize the supported mechanism as a loadable kernel module not requiring modifications to the native kernel. For example, to support the TCP handoff protocol described in Section 3.2 the ScalaServer prototype [11, 96] from Rice University modifies the FreeBSD operating system, while the ClubWeb prototype [6] modifies the Linux kernel. A set of dynamically loadable modules that provide the TCP handoff mechanism in a STREAMS-based TCP/IP implementation has been proposed in [116].

To the best of our knowledge, Central Dispatch [104] is the only commercial product that implements a one-way architecture with a layer-7 Web switch. This solution requires the installation of a kernel module on both the Web switch and the servers to support the proprietary TCP connection hop mechanism (see Section 3.2).

It is worth noting that all the proposed request routing mechanisms are transparent to the HTTP server software running on the Web server nodes. Incoming traffic on already established

connections is forwarded to the target server through an efficient forwarding module layered between the network interface card and the TCP/IP stack. A distinguishing feature of the request routing mechanism deployed by ScalaServer is the support to HTTP/1.1 persistent connections which allows to assign HTTP requests to different servers through the same TCP connection. This is achieved through a *back-end forwarding* mechanism, that allows the original target node to forward a request to a second server selected by the Web switch [11]. Assigning individual HTTP requests traveling on the same TCP connection to multiple servers appears to be also a feature of Request Switch [92], but no information is currently provided about the routing mechanism because of patent pending reasons.

As regards the server selection, one-way solutions working at layer-7 typically analyze the HTTP header content prior to dispatching the request to an appropriate server. In the ScalaServer [96] and ClubWeb prototype [6] the Web switch selects a server on the basis of the LARD and CAP policy, respectively. Both policies are described in Section 6.3.

In Central Dispatch [104], the Web dispatching algorithm uses client information in addition to server performance and availability. Specifically, upon receipt of an HTTP request, the Resonate switch parses the URL to determine the requested content and applies some dispatching rule that may chosen by the system administrator. If more than one node is available to serve the request, the Web switch transfers the TCP connection to the least loaded server.

8 Integrated dispatching mechanisms

Reliability, scalability and security are critical qualities in Web-based information services. Indeed, 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 reliability issue can be addressed by a replication of the switch device integrated with some heartbeat technique (e.g., [82]) or by some distributed dispatching mechanism (e.g., [14, 87, 117]). This section focuses on the scalability issues and presents some extensions to the basic Web cluster architecture described in Section 2. The most interesting proposals for improving cluster scalability combine two request routing mechanisms and exploit some system caching techniques.

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 [54] where the Web switch operated at the TCP layer, and it is now adopted by several geographically distributed Web systems. Indeed, when all strategies have been implemented, and the Web cluster scalability and reliability is primarily limited by the network connection to Internet, the best alternative for a content provider that does not want to refer to outsourcing solutions is to distribute multiple Web clusters over different Internet zones.

An interesting idea for improving Web cluster scalability is to combine the performance of a

content-blind dispatcher (DNS, layer-4 Web switch) with the caching features of a content-aware dispatcher, that can be implemented by a layer-7 switch or by a Web server.

A change of the basic Web cluster architecture presented in Section 2 integrates a layer-4 Web switch with two or more layer-7 Web switches that take content-aware dispatching decisions and provide some caching functionality. Indeed, the caches store frequently accessed Web objects and respond to requests for these objects, thus relieving that workload from the Web servers. The first proposal where caching is integrated into one layer-7 Web switch is in [77], the so called *Web server accelerator*. A first improvement was to use sophisticated mechanisms that allow caching of dynamic objects too [35]. An evolution of the basic Web server accelerator architecture adds a set of Web server accelerators between the Web switch and the Web servers [112, 111], 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 the requests, it routes them to the cache nodes regardless of URL contents. 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.

A different approach to improve Web cluster efficiency is to perform content-aware dispatching or caching through the Web servers instead of additional layer-7 Web switches. The first dispatching level carried out by a layer-4 Web switch (or by the authoritative DNS if we refer to a distributed Web system) selects one Web server typically by means of a static algorithm. There are various proposals in this sense, that basically differ for the way the Web content is distributed (that is, replicated [12, 31] or partitioned [38]) and the system information is shared among the nodes.

If each node can access all Web documents, the server selected by the Web switch may redirect the request for load sharing or caching reasons, for example it is aware that another server is lightly loaded or holds the requested object in the disk cache. This system information may be centralized in a dispatcher component [12] or distributed among all the servers [38, 31].

In the prototype proposed by [12] the layer-4 Web switch selects one server through the dynamic Weighted Round-Robin algorithm. The server accepts the connection, parses the client request, and contacts a central dispatcher located in the internal LAN that decides about (re-)assignment on the basis of the LARD policy. If the dispatcher selects a different server, the first contacted server hands off the connection towards the other server, that responds directly to the client. When a TCP connection handoff occurs, the server 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, while the second-level routing is distributed and carried out by each server. The motivation is that the processing overhead on a layer-7 switch

is caused by the routing mechanism and not by the server selection task. Although the authors demonstrate that the dispatcher is not the system bottleneck [12], a distributed dispatching algorithm carried out by each server node could avoid the communication overheads with the centralized dispatcher. This alternative is chosen by [31] that implement a Web cluster with a global caching mechanism. Depending on the size of the requested object, and its presence in the local cache or in the caches of other servers, the first contacted Web server replies to the client immediately or after having retrieved the file from another server. Server load and caching information are periodically broadcasted by each server.

In the prototype proposed by [38] the Web content is not entirely replicated. Just a small set of the most popular files (namely, *core*) can be accessed by any server, while the other files are partitioned among the nodes. If the client request is for a core file, the request is processed by the first contacted server, otherwise the server redirects it to the designated server node. The global information about document location is stable, because the core is determined by analyzing periodically (e.g., daily) the workload access patterns.

9 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 in one section of this survey. We outline main ideas and give references for further reading by distinguishing static content from content that is dynamically generated at the time of a client request.

9.1 Distribution of static content

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 content among the Web servers of the cluster: to replicate the content tree across independent file systems running on the servers; to share information by means of a distributed file system, such as Andrew File System (AFS) and 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 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 periods of low traffic (if any), 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. Besides that, for very popular sites that receive millions of requests per day from around the world, any maintenance, system upgrade and information update has to be done while the Web cluster is online.

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 among the servers and may increase the response time because 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, that is, replicating the content tree on each server node or sharing 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 [58, 104, 125]. On the other hand, content partitioning can lead to load imbalance produced by the uneven distribution of Web document 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, [102] propose a sophisticated mechanism that simultaneously use several strategies for replicating Web content. 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 server nodes.

9.2 Dynamic Web content

In the old days, the Web was largely based on static and read-only information, but now a large percentage of Web sites provide information and services 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 service time. Nevertheless, dynamic pages and services are becoming essential in

modern Web sites where Web-based technologies have emerged as a valid alternative to traditional client-server computing. Indeed, the Web simplicity and its compatibility with existing protocols 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-based services, databases and other (legacy) applications are typically hosted on a set of servers behind 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 17. A front-end Web switch is located between the Internet and the first set of Web server nodes (*presentation layer*) that run the HTTP daemons. They 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 other servers.

A so called Web Application Server layer (*middle layer*) can be interposed between the Web servers and the back-end servers (*data layer*). (In less complex architectures where the middle layer is thin, Web application processes can run on the Web server nodes). The Web application servers run the software that handles all operations between browser-based clients and a company's back-end database. The Web 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 Web application servers.

In the reality, a multi-tier architecture is even more complex than that shown in Figure 17. Indeed, when a Web cluster hosts business applications, strict security strategies are employed to protect the safety of these systems. 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 Web content opens several new issues that are beyond the scope of this survey. The alternative solutions depend also on the application software, the chosen middleware and database technology. For example, commercial Web-based service software, 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 applica-

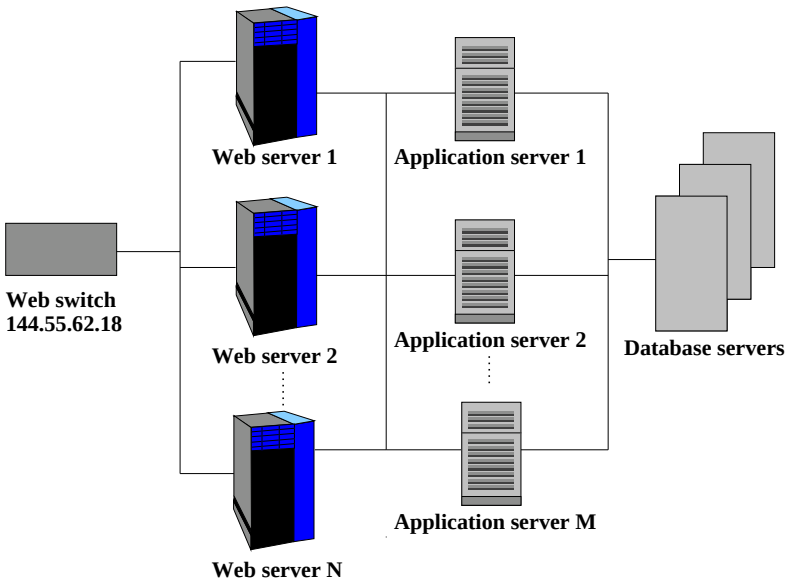

Figure 17: An example of multi-tier architecture for a Web cluster.

tions. A wide spectrum of new technologies is coming on the scene, such as Cold Fusion, Domino, WebBuilder, IBM DB2 WWW Connection, Oracle9*i* 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.

[129] 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 [24, 62]. A different solution for load balancing based on CORBA middleware

technology is proposed in [95] 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 to improve performance of the multi-tier systems are not oriented towards load balancing algorithms, but to mechanisms for caching query results and dynamic content at different layers [27, 52, 94, 101, 124].

10 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 security and fault-tolerance, 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 [1, 17, 22, 39, 57, 78, 86, 99, 118] and this research topic has been moved towards cluster-based platforms only recently. A multi-node Web architecture integrated with admission control and performance isolation mechanisms has been proposed in [10]. Some recent research efforts focus on

how providing QoS support through the Web switch, by taking into account request information at layer-4 switches [29] as well as at layer-7 switches [37, 131], 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 [28, 131].

While layer-4 Web cluster architectures may be considered an almost 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 fully explored yet. Some companies commercialize layer-7 Web switches with very simple dispatching mechanisms that are mainly oriented to statically partition 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 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 commercial 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 users 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 [56]. Finally, we observe that most of the topics and algorithms analyzed in this paper change completely if we assume that the multiple servers (or Web clusters) of the content provider are distributed over the world rather than grouped in a local area.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments and suggestions that have improved the presentation and correctness of this survey. The first three authors also acknowledge the support of MIUR-Cofin2001 in the framework of the project "High quality Web systems".

References[3]

[1] T. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel and Distributed Systems*, 13(1):80–96, Jan. 2002.

[3] All URLs are current as of January 2002.

[2] Akamai Tech., 2002. `http://www.akamai.com`.

[3] Allot Communications, 2002. `http://www.allot.com`.

[4] J. Almeida, M. Dabu, A. Manikntty, and P. Cao. Providing differentiated levels of service in Web content hosting. In *Proceedings of Workshop on Internet Server Performance (Madison, WI, June)*, 1998.

[5] 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.

[6] M. Andreolini, E. Casalicchio, M. Colajanni, and M. Mambelli. Performance analysis of layer-7 switches for cluster-based Web servers. Technical Report RR-01.24, Univ. of Roma Tor Vergata, Computer Engineering Dept., 2001.

[7] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha. Design, implementation and performance of a content-based switch. In *Proceedings of the 19th IEEE International Conference on Computer Communications (INFOCOM 2000) (Tel Aviv, Israel, March)*, pages 1117–1126, Los Alamitos, CA, 2000. IEEE Computer Soc. Press.

[8] 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.

[9] 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.

[10] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2000) (Santa Clara, CA, June)*, pages 90–101, New York, June 2000. ACM Press.

[11] M. Aron, P. Druschel, and Z. Zwaenepoel. Efficient support for P-HTTP in cluster-based Web servers. In *Proceedings of the 1999 USENIX Annual Technical Conference (Monterey, CA, June)*, pages 185–198, Berkeley, CA, 1999. USENIX Assoc.

[12] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *Proceedings of the 2000 USENIX Annual Technical Conference (San Diego, CA, June)*, Berkeley, CA, 2000. USENIX Assoc.

[13] Array Networks, 2002. `http://www.arraynetworks.net`.

[14] L. Aversa and A. Bestavros. Load balancing a cluster of Web servers using Distributed Packet Rewriting. In *Proceedings of the 19th IEEE International Performance, Computing, and Communication Conference (Phoenix, AZ, Feb.)*, pages 24–29, Los Alamitos, CA, 2000. IEEE Computer Soc. Press.

[15] 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.

[16] G. Banga, P. Druschel, and J. C. Mogul. Better operating system features for faster network servers. *ACM Performance Evaluation Review*, 26(3):23–30, Dec. 1998.

[17] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation (New Orleans, LA, Feb.)*, pages 45–58, Berkeley, CA, 1999. USENIX Assoc.

[18] N. Bansal and M. Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proceedings of the 2001 ACM/IFIP Joint International Conference on Measurement and Modeling of Computer Systems (Cambridge, MA, June)*, pages 279–290, New York, June 2001. ACM Press.

[19] P. Barford and M. E. Crovella. Critical path analysis of TCP transactions. *IEEE/ACM Trans. Networking*, 9(3):238–248, June 2001.

[20] G. Barish and K. Obraczka. World Wide Web caching: Trends and techniques. *IEEE Commun.*, 38(5):178–184, May 2000.

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

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

[23] T. Bourke. *Server Load Balancing.* O'Reilly and Associates, Sebastopol, CA, 2001.

[24] E. A. Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, July/Aug. 2001.

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

[26] R. B. Bunt, D. L. Eager, G. M. Oster, and C. L. Williamson. Achieving load balance and effective caching in clustered Web servers. In *Proceedings of the 4th International Web Caching Workshop (San Diego, CA, Apr.)*, pages 159–169, Apr. 1999.

[27] K. S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal. Enabling dynamic content caching for database-driven Web sites. In *Proceedings of 2001 ACM SIGMOD International Conf. on Management of Data (Santa Barbara, CA)*, pages 532–543, New York, 2001. ACM Press.

[28] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli. Web switch support for differentiated services. *ACM Performance Evaluation Review*, 29(2):14–19, Sept. 2001.

[29] V. Cardellini, E. Casalicchio, M. Colajanni, and S. Tucci. Mechanisms for quality of service in Web clusters. *Computer Networks*, 36(6):759–769, Nov. 2001.

[30] 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.

[31] E. V. Carrera and R. Bianchini. Efficiency vs. portability in cluster-based network servers. In *Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (Snowbird, UT, June)*, pages 113–122, New York, June 2001. ACM Press.

[32] E. Casalicchio, V. Cardellini, and M. Colajanni. Content-aware dispatching algorithms for cluster-based Web servers. *Cluster Computing*, 5(1):67–76, Jan. 2002.

[33] E. Casalicchio and M. Colajanni. A client-aware dispatching algorithm for Web clusters providing multiple services. In *Proceedings of the 10th International World Wide Web Conference (Hong Kong, May)*, pages 535–544, New York, May 2001. ACM Press.

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

[35] J. Challenger, A. Iyengar, and P. Dantzig. A scalable system for consistently caching dynamic Web data. In *Proceedings of the 18th IEEE Conference on Computer Communications (INFOCOM 1999) (New York, NY, March)*, pages 294–303, Los Alamitos, CA, 1999. IEEE Computer Soc. Press.

[36] 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, Heidelberg, 2001.

[37] X. Chen and P. Mohapatra. Providing differentiated service from an Internet server. In *Proceedings of the 8th IEEE International Conference on Computer Communications and Networks (Boston, MA, Oct.)*, pages 214–217, Los Alamitos, CA, Oct. 1999. IEEE Computer Soc. Press.

[38] L. Cherkasova and M. Karlsson. Scalable Web server cluster design with WARD. In *Proceedings of the 3rd International Workshop on Advanced issues of E-Commerce and Web-Based Information Systems (San Jose, CA, June)*, pages 212–221, Los Alamitos, CA, 2001. IEEE Computer Soc. Press.

[39] L. Cherkasova and P. Phaal. Session based admission control: a mechanism for improving performance of commercial Web sites. In *Proceedings of the International Workshop on Quality of Service (London, UK, June)*, 1999.

[40] L. Cherkasova and S. Ponnekanti. Optimizing the "content-aware" load balancing strategy for shared Web hosting service. In *Proceedings of the 8th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2000) (San Francisco, CA, Aug./Sept.)*, pages 492–499, Los Alamitos, CA, 2000. IEEE Computer Soc. Press.

[41] G. Ciardo, A. Riska, and E. Smirni. EQUILOAD: a load balancing policy for clustered Web servers. *Performance Evaluation*, 46(2-3):223–239, Oct. 2001.

[42] Cisco Systems, 2002. `http://www.cisco.com/`.

[43] K. G. Coffman and A. M. Odlyzko. Internet growth: Is there a "Moore's Law" for data traffic? In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*. Kluwer Academic Publ., Dordrecht, The Netherlands, 2001.

[44] A. Cohen, S. Rangarajan, and H. Slye. On the performance of TCP splicing for URL-aware redirection. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (Boulder, CO)*, Berkeley, CA, Oct. 1999. USENIX Assoc.

[45] E. Cohen and H. Kaplan. Proactive caching of DNS records: Addressing a performance bottleneck. In *Proceedings of the 2001 Symposium on Applications and the Internet (San Diego, CA, Jan.)*, pages 85–94, Los Alamitos, CA, 2001. IEEE Computer Soc. Press.

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

[47] Coyote Point Systems, 2002. http://www.coyotepoint.com.

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

[49] M. E. Crovella, R. Frangioso, and M. Harchol-Balter. Connection scheduling in Web servers. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (Boulder, CO, Oct.)*, Berkeley, CA, 1999. USENIX Assoc.

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

[51] 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.

[52] L. Degenaro, A. Iyengar, I. Lipkind, and I. Rouvellou. A middleware system which intelligently caches query results. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000) (New York, NY, April)*, pages 24–44, Heidelberg, 2000. Springer-Verlag.

[53] 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, 1999.

[54] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available Web server. In *Proceedings of the 41st IEEE Computer Society International Conference (San Jose, CA, Feb.)*, pages 85–92, Los Alamitos, CA, Feb. 1996. IEEE Computer Soc. Press.

[55] Digital Island, 2002. http://www.digitalisland.net.

[56] R. Doyle, J. S. Chase, S. Gadde, and A. M. Vahdat. The trickle-down effect: Web caching and server request distribution. In *Proceedings of the 6th International Workshop on Web Caching and Content Delivery (Boston, MA)*, Amsterdam, June 2001. Elsevier Science.

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

[58] F5 Networks, 2002. http://www.f5labs.com/.

[59] D. Ferrari and S. Zhou. An empirical investigation of load indices for load balancing applications. In *Proceedings of the 12th International Symposium on Computer Performance Modeling, Measurement, and Evaluation (Brussels, Belgium)*, pages 515–528, Amsterdam, 1987. Elsevier Science.

[60] 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.

[61] Foundry Networks. Foundry networks' serveriron, 2002. http://www.foundrynet.com/products/webswitches/serveriron/.

[62] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (Saint-Malo, France, Oct.)*, pages 78–91, New York, 1997. ACM Press.

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

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

[65] G. Gilder. Fiber keeps its promise: Get ready. Bandwidth will triple each year for the next 25. *Forbes*, 1997. 7 April.

[66] A. Goldberg, R. Buff, and A. Schmitt. Secure Web server performance dramatically improved by caching SSL session keys. In *Proceedings of Workshop on Internet Server Performance (Madison, WI, June)*, 1998.

[67] J. Gray and P. Shenoy. Rules of thumb in data engineering. In *Proceedings of the 16th IEEE International Conference on Data Engineering (San Diego, CA, Apr.)*, pages 3–10, Los Alamitos, CA, Apr. 2000. IEEE Computer Soc. Press.

[68] 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.

[69] J. Hennessy. The future of system research. *IEEE Computer*, 32(8):27–33, Aug. 1999.

[70] Y. Hu, A. Nanda, and Q. Yang. Measurement, analysis and performance improvement of Apache Web server. In *Proceedings of the 18th IEEE International Performance, Computing, and Communications Conference (Phoenix, AZ, Feb.)*, Los Alamitos, CA, Feb. 1999. IEEE Computer Soc. Press.

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

[72] 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.

[73] IBM. IBM WebSphere Edge Server, 2002. `http://www.ibm.com/software/webservers/edgeserver/`.

[74] Intel. Intel NetStructure, 2002. `http://www.intel.com/network/idc/products/traffic_equipment.htm`.

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

[76] 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.

[77] E. Levy-Abegnoli, A. Iyengar, J. Song, and D. Dias. Design and performance of a Web server accelerator. In *Proceedings of the 18th IEEE International Conference on Computer Communications (INFOCOM 1999) (New York, NY, March)*, pages 135–143, Los Alamitos, CA, Mar. 1999. IEEE Computer Soc. Press.

[78] K. Li and S. Jamin. A measurement-based admission-controlled Web server. In *Proceedings of the 19th IEEE International Conference on Computer Communications (INFOCOM 2000) (Tel Aviv, Israel, March)*, pages 651–659, Los Alamitos, CA, Mar. 2000. IEEE Computer Soc. Press.

[79] Q. Li and B. Moon. Distributed Cooperative Apache Web server. In *Proceedings of the 10th International World Wide Web Conference (Hong Kong, May)*, pages 555 – 564, New York, 2001. ACM Press.

[80] Linux Virtual Server. Linux Virtual Server project, 2002. `http://www.linuxvirtualserver.org/`.

[81] Lucent Tech. Lucent Web Switch, 2002. `http://www.bell-labs.com/project/webswitch/`.

[82] M.-Y. Luo and C.-S. Yang. Constructing zero-loss Web services. In *Proceedings of the 20th IEEE International Conference on Computer Communications (INFOCOM 2001) (Anchorage, AK, Apr.)*, pages 1781–1790, Los Alamitos, CA, Apr. 2001. IEEE Computer Soc. Press.

[83] M.-Y. Luo and C.-S. Yang. System support for scalable and reliable and highly manageable Web hosting service. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (San Francisco, CA, March)*, Berkeley, CA, 2001. USENIX Assoc.

[84] A. M. Luotonen. *Web Proxy Servers*. Prentice Hall, Englewood Cliffs, NJ, 1997.

[85] D. Maltz and P. Bhagwat. Application layer proxy performance using TCP splice. Technical Report RC 21139, IBM T. J. Watson Research Center, 1998.

[86] 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.

[87] Microsoft. Network load balancing, 2002. `http://www.microsoft.com/windows2000/techinfo/howitworks/cluster/nlb.as%p`.

[88] Mirror Image Internet, 2002. `http://www.mirror-image.com/`.

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

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

[91] E. M. Nahum, T. Barzilai, and D. D. Kandlur. Performance issues in WWW servers. *IEEE/ACM Trans. Networking*, 10(2):2–11, Feb. 2002.

[92] NetScaler. Netscaler's Request Switch, 2002. `http://www.netscaler.com`.

[93] Nortel Networks. Nortel Networks Web OS, 2002. `http://www.nortelnetworks.com/products/01/alteon/`.

[94] Oracle. Oracle9*i*AS Web Cache, 2002. `http://www.oracle.com/ip/deploy/ias/caching/index.html`.

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

[96] 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 *Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems (San Jose, CA, Oct.)*, pages 205–216, New York, 1998. ACM Press.

[97] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable Web server. In *Proceedings of the 1999 USENIX Annual Technical Conference (Monterrey, CA, June)*, pages 199–212, Berkeley, CA, 1999. USENIX Assoc.

[98] V. S. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: A unified I/O buffering and caching system. *ACM Trans. Comput. Syst.*, 18(1):37–66, Feb. 2000.

[99] R. Pandey and R. Barnes, J. F. Olsson. Supporting quality of service in HTTP servers. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (Puerto Vallarta, Mexico, June)*, pages 247–256, New York, June 1998. ACM Press.

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

[101] Persistence Software. Persistence Dynamai, 2002. `http://www.persistence.com/products/dynamai/index.php`.

[102] G. Pierre, van Steen M., and A. S. Tanenbaum. Dynamically selecting optimal distribution strategies for Web documents. *IEEE Trans. Comput.*, 51, Feb. 2002. To appear in 2002.

[103] Radware, 2002. `http://www.radware.com/`.

[104] Resonate, 2002. `http://www.resonate.com/`.

[105] A. Rijsinghani. *Computation of the Internet checksum via incremental update.* RFC 1624, May 1994.

[106] D. Rosu, A. Iyengar, and D. Dias. Web proxy acceleration. *Cluster Computing*, 4(4):307–317, Oct. 2001.

[107] M.-C. Rosu and D. Rosu. Evaluation of TCP splice benefits in Web proxy servers. In *Proceedings of the 11th International World Wide Web Conference (Honolulu, HI, May)*, New York, 2002. ACM Press.

[108] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of DNS-based server selection. In *Proceedings of the 20th IEEE International Conference on Computer Communications (INFOCOM 2001) (Anchorage, AK, April)*, pages 1801–1810, Los Alamitos, CA, 2001. IEEE Computer Soc. Press.

[109] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE Computer Soc. Press, Los Alamitos, CA, 1995.

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

[111] J. Song, A. Iyengar, E. Levy-Abegnoli, and D. Dias. Architecture of a Web server accelerator. *Computer Networks*, 38(1):75–97, Jan. 2002.

[112] J. Song, E. Levy-Abegnoli, A. Iyengar, and D. Dias. Design alternatives for scalable Web server accelerators. In *Proceedings of the 2000 IEEE International Symposium on Performance Analysis of Systems and Software (Austin, TX, April)*, pages 184–192, Los Alamitos, CA, 2000. IEEE Computer Soc. Press.

[113] O. Spatscheck, J. S. Hansen, J. H. Hartman, and L. L. Peterson. Optimizing TCP forwarder performance. *IEEE/ACM Trans. Networking*, 8(2):146–157, Apr. 2000.

[114] P. Srisuresh and K. Egevang. *Traditional IP Network Address Translator (Traditional NAT).* RFC 3022, Jan. 2001.

[115] P. Srisuresh and D. Gan. *Load sharing using IP Network Address Translation.* RFC 2391, Aug. 1998.

[116] W. Tang, L. Cherkasova, L. Russell, and M. W. Mutka. Modular TCP handoff design in STREAMS-based TCP/IP implementation. In *Proceedings of the 1st International Conference on Networking (Colmar, France, July)*, volume 2094 of *Lecture Notes in Computer Science*, pages 71–80, Heidelberg, 2001. Springer-Verlag.

[117] S. Vaidya and K. Christensen. A single system image server cluster using duplicated MAC and IP addresses. In *Proceedings of the IEEE 26th Conference on Local Computer Networks (Tampa, FL, Nov.)*, pages 206–214, Los Alamitos, CA, 2001. IEEE Computer Soc. Press.

[118] 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.

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

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

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

[122] D. Wessels. *Web Caching.* O'Reilly and Associates, Sebastopol, CA, 2001.

[123] J. L. Wolf and P. S. Yu. On balancing the load in a clustered Web farm. *ACM Trans. Internet Technology*, 1(2):231–251, Nov. 2001.

[124] K. Yagoub, D. Florescu, V. Issarny, and P. Valduriez. Caching strategies for data-intensive Web sites. In *Proceedings of the 24th International Conference on Very Large Databases (Cairo, Egypt, Sept.)*, pages 188–199, San Francisco, Apr. 2000. Morgan Kaufmann.

[125] C.-S. Yang and M.-Y. Luo. A content placement and management system for distributed Web-server systems. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (Taipei, Taiwan, April)*, pages 691–698, Los Alamitos, CA, 2000. IEEE Computer Soc. Press.

[126] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using Smart Clients to build scalable services. In *Proceedings of the 1997 USENIX Annual Technical Conference (Anaheim, CA, Jan.)*, pages 105–117, Berkeley, CA, 1997. USENIX Assoc.

[127] Zeus Tech., 2002. `http://www.zeus.com/`.

[128] X. Zhang, M. Barrientos, J. B. Chen, and M. Seltzer. HACC: An architecture for cluster-based Web servers. In *Proceedings of the 3rd USENIX Windows NT Symposium (Seattle, WA, July)*, pages 155–164, Berkeley, CA, 1999. USENIX Assoc.

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

[130] H. Zhu and H. Tang. Class-based cache management for dynamic Web content. In *Proceedings of the 20th IEEE International Conference on Computer Communications (INFOCOM 2001) (Anchorage, AK, April)*, pages 1215 –1224, Los Alamitos, CA, Apr. 2001. IEEE Computer Soc. Press.

[131] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In *Proceedings of the 20th IEEE International Conference on Computer Communications (INFOCOM 2001) (Anchorage, AK, April)*, pages 679 –688, Los Alamitos, CA, Apr. 2001. IEEE Computer Soc. Press.