Dynamic Load Balancing on Web-server Systems *

Valeria Cardellini
Università di Roma "Tor Vergata"
Roma, I-00133
cardellini@uniroma2.it

Michele Colajanni
Università di Modena e Reggio Emilia
Modena, I-41100
colajanni@unimo.it

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

Abstract

Popular Web sites can neither rely on a single powerful server nor on independent mirrored-servers to support the ever increasing request load. Scalability and availability can be provided by distributed Web-server architectures that schedule client requests among the multiple server nodes in a user-transparent way. In this paper we will review the state of the art in load balancing techniques on distributed Web-server systems. We will analyze the efficiency and limitations of the various approaches and their tradeoff.

Keywords: Distributed Systems, Load Balancing, Scheduling, Web-server, WWW.

1 Introduction

The explosive growth of traffic on the World Wide Web is causing a rapid increase in the request rate to popular Web sites. These sites can suffer from severe congestion, especially in conjunction with special events, such as Olympic Games and NASA Pathfinder. The administrators of popular sites constantly face the need of improving the capacity of their Web-servers to meet the demands of the users.

One approach to handle popular Web sites is based on the replication of information across a mirrored-server architecture, which provides a list of independent URL sites that have to be manually selected by the users. This solution has a number of disadvantages, including the not user-transparent architecture, and the lack of control on the request distribution by the Web-server system. A more promising solution is to rely on a distributed architecture that can route incoming requests among several server nodes in a user-transparent way. This approach can potentially improve throughput performance and provide Web-server systems with high scalability and availability. However, a number of challenges must be addressed to make a distributed server cluster function efficiently as a single server within the framework of the HTTP protocol and Web browsers.

In this paper we describe and discuss the various approaches on routing requests among the distributed Web-server nodes. We analyze the efficiency and the limitations of the different techniques and the tradeoff among the alternatives. The scope of this paper is not to describe the detailed technical features of each approach, for which we refer the reader to the appropriate literature. Its aim is rather to analyze the characteristics of each approach and its effectiveness on Web-server scalability.

In Section 2 we propose a classification of existing approaches based on the entity that dispatches the client requests among the distributed servers: client-based, DNS-based, dispatcher-based, and server-based. From Section 3 to Section 6 we describe and discuss in more details each of these approaches. In Section 7 we provide a comparison of the different solutions. In Section 8 we give some final remarks for future work.

2 Distributed Web-server systems

In this paper we refer to *user* as anyone who is accessing the information on the World Wide Web, while we define *client* as a program, typically a Web browser, that establishes connections to Internet for satisfying user requests. Clients are connected to the network through gateways; we will refer to the network sub-domain behind these local gateways as *domain*. The purpose of a Web *server* is to store information and serve client requests. To request a document from a Web-server host, each client first needs to resolve the mapping of the host-name contained in the URL to an IP address. The client obtains the IP address of a Web-server node through an address mapping request to the *Domain Name System* (DNS) server, which is responsible for the address resolution process. However, to reduce traffic load due to address resolutions, various entities (e.g., intermediate name servers, local gateways, and browsers) can cache some address mapping for a certain period.

In this paper, we will consider as a *distributed Web-server system* any architecture consisting of multiple Web-server hosts with some mechanism to spread the incoming client requests among the servers. The nodes may be locally or geographically distributed (namely, LAN and WAN Web-server systems, respectively). As assumed by existing distributed Web-server systems, each server in the cluster can respond to any client request. There are essentially two mechanisms for implementing information distribution among the server nodes: to replicate the content tree on the local disk of each server or to share information by using a distributed file system. The former solution can be applied to both LAN and WAN Web-server systems, the latter works fine only for LAN Web-server systems.

A distributed 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. Unlike the distributed Web-server system, the mirrored-server based architecture is visible to the user, thereby violating the main transparency requirement.

In this paper we classify the distributed Web-server architectures by focusing on the entity which distributes the incoming requests among the servers. In such a way, we identify four classes of methods, the first of which requires software modification on the client side, and the remaining three affect one or more components of the Web-server cluster.

- Client-based approach
- DNS-based approach
- Dispatcher-based approach (at network level)
- Server-based approach.

All approaches of interest to this paper satisfy the architecture transparency requirement. Other considered factors include *scalability*, *load balancing*, *availability*, *applicability* to existing Web standards (namely, *backward compatibility*), and *geographical scalability* (i.e., solutions applicable to both LAN and WAN distributed systems).

3 Client-based approach

The approach of routing the document requests from the client side can be applied to any replicated Web-server architecture even when the nodes are loosely or not coordinated at all. There are two main approaches that put the server selection mechanism on the client side by satisfying the user-transparency requirement: the routing to the Web cluster can be provided by the Web clients (i.e., the browsers) or by the client-side proxy servers.

3.1 Web clients

The Web clients can play an active role in the request routing, if we assume that the Web clients know the existence of the replicated servers of the Web-server system. Figure 1 shows the steps through which a client can directly route user requests to a Web-server node. This figure and all those in the following sections showing the main features of different approaches for distributing

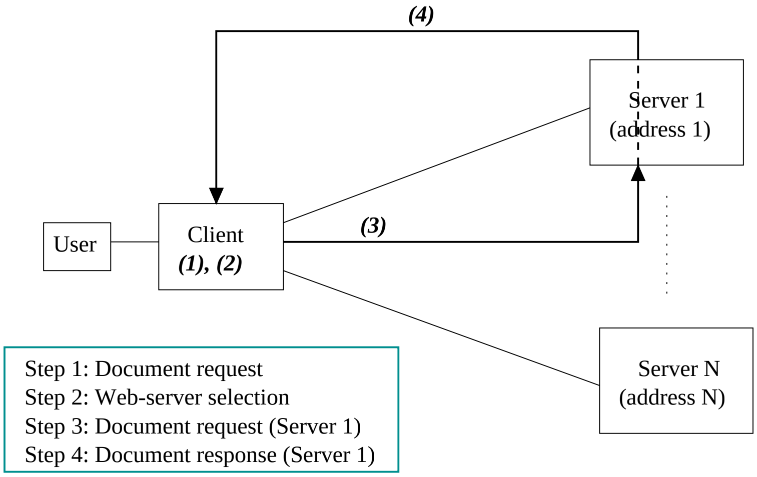

(4)
Server 1
(address 1)
User
Client
(1), (2)
(3)
Server N
(address N)
Step 1: Document request
Step 2: Web-server selection
Step 3: Document request (Server 1)
Step 4: Document response (Server 1)

Figure 1: Web clients based approach.

requests are based on a protocol-centered description: the labeled arcs denotes the main steps that are necessary to request a document, select a Web-server and receive a response.

In particular, Figure 1 shows the simplicity of the Web clients based approach. Upon received the user request (step 1), the Web client is able to select a node of the cluster (step 2) and, once resolved the address mapping (not shown in the figure because it is not relevant to the server selection), submits the request to the selected node (step 3). This Web-server is responsible for responding to the client (step 4). A successive request from the same client can reach another server. We consider two representative examples of the scheduling approach based on Web clients.

Netscape's approach. The way of accessing the Netscape Communication site through the Netscape Navigator browser has been the first example of client-based distributed Web-server architecture [20, 24]. The mechanism for spreading the load across the multiple nodes of the Netscape Web-server system is as follows. When the user accesses the Netscape home page (located at the URL www.netscape.com), Navigator selects a random number i between 1 and the number of servers and directs the user request to the node www$i$.netscape.com.

The Netscape approach is not generally applicable, as the number of Web sites that have the privilege to own a browser to distribute the load among their servers is very limited. (Such a solution could find a wider practical application in the Intranets of large corporations.) Furthermore, this architecture is not scalable unless the client browser is re-installed. Moreover, the random selection scheme, which is used for spreading the requests, does not guarantee server availability and load balancing among the servers.

Smart Clients. This solution [27] aims at migrating some server functionality to the client machine, in contrast with the traditional approach in which the Web client is not involved. The client request routing is achieved through a *Java applet* which is executed at the client side each time a user requests an access to the distributed Web-server system. As the applet knows the IP addresses of all server nodes in the Web cluster, it can send messages to each of the servers to get information on the node states and network delays. This information is analyzed by the applet to select the most appropriate server. Requests from the client node executing the applet are then forwarded to the selected server.

A major drawback of this approach is the increase in network traffic due to the message exchanges to monitor server load and network delay at each server node. Moreover, although this approach provides scalability and availability, it lacks of portability on the client side.

3.2 Client-side proxies

The proxy server is another important Internet entity that may dispatch client requests to Web-server nodes. We include these approaches in this section because from the Web cluster point of view the proxy servers are quite similar to clients. An interesting proposal that combine caching and server replication is described in [4]. Their Web Location and Information Service (WLIS) implemented at the client-side proxy can keep track of replicated URL addresses and route client requests to the most appropriate server.

We do not further investigate this class of solutions because they are affected by the same problem that limits the applicability of all Web client approaches. Indeed, any load balancing mechanism carried out by the client-side proxy servers requires some modifications on Internet components, that typically are not controlled by the same institution/company that manages the distributed Web-server system.

4 DNS-based approach

The distributed Web-server architectures that use request routing mechanisms on the cluster side do not suffer from the limited applicability problems of the client-based approaches. Typically, the architecture transparency is obtained through a single virtual interface to the outside world at least at the URL level. (We will see that other approaches provide a single virtual interface even at the IP level). In the first implementation of a cluster side solution, the responsibility of spreading the requests among the servers is delegated to the *cluster DNS*, that is, the authoritative DNS server for the domain of the cluster nodes. Through the translation process from the symbolic name (URL) to IP address, the cluster DNS can select any node of the Web-server cluster. In particular, this translation process allows the cluster DNS to implement a large set of scheduling policies to select the appropriate server. On the other hand, it should be observed that the DNS has a limited control on the request reaching the Web cluster. Indeed, between the client and the cluster DNS, there are many intermediate name servers that can cache the logical name to IP address mapping (i.e. *network-level address caching*) to reduce network traffic. Moreover every Web client browser typically caches some address resolution (i.e. *client-level address caching*).

Besides providing the IP address of a node, the DNS also specifies a validity period, known as *Time-To-Live* (TTL), for caching the symbolic to IP address mapping. After the expiration of the TTL, the address mapping request is forwarded to the cluster DNS for assignment to a Web-server node. Otherwise, the address mapping request is handled by some intermediate name server. These two alternative ways of URL address resolution are shown in Figure 2. If an intermediate

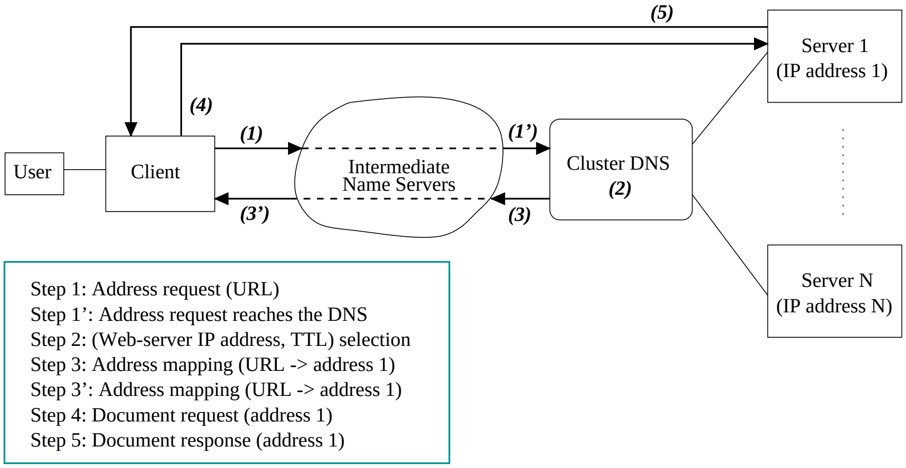

(5)
Server 1
(IP address 1)
(4)
(1)
(1')
User
Client
Intermediate
Name Servers
Cluster DNS
(2)
(3')
(3)
Server N
(IP address N)
Step 1: Address request (URL)
Step 1': Address request reaches the DNS
Step 2: (Web-server IP address, TTL) selection
Step 3: Address mapping (URL -> address 1)
Step 3': Address mapping (URL -> address 1)
Step 4: Document request (address 1)
Step 5: Document response (address 1)

Figure 2: DNS-based approach.

name server holds a valid mapping for the cluster URL, it can resolve the address mapping request without forwarding it to another intermediate name server or to the DNS (loop 1, 3'). Otherwise, the address request reaches the cluster DNS (step 1, 1'), which selects the IP address of a Web-server and the TTL (step 2). The URL to IP address mapping and the TTL value are forwarded to all intermediate name servers along the path (step 3) and to the client (step 3').

The DNS control on the address caching is limited by the following factors. First of all, the TTL period does not work on the browser caching. Moreover, the DNS may not be able to reduce the TTL to values close to zero because of the presence of non-cooperative intermediate name servers that ignore very small TTL periods. On the other hand, the limited control on client requests prevents DNS from becoming a potential bottleneck.

We distinguish the DNS-based architectures through the scheduling algorithm that the cluster DNS uses to share the load among the Web-server nodes. We first consider various policies where the DNS makes server selection considering some system state information, and then assigns the same TTL value to all address mapping requests (*constant TTL algorithms*). We next describe an alternative class of algorithms that adapt the TTL values on the basis of dynamic information from servers and/or clients (*dynamic TTL algorithms*).

4.1 Constant TTL algorithms

The constant TTL algorithms are classified on the basis of the system state information (i.e., none, client load or client location, server load, or combination of them) the DNS uses for the Web-server choice.

System stateless algorithms. The most important example of this class is the Round Robin DNS (namely, *DNS-RR*) approach implemented by the National Center for Supercomputing Applications (NCSA) [23]. This was the first distributed homogeneous Web-server architecture, and represents the basic approach for various subsequent enhancements. To support the fast growing demand to its site, NCSA developed a Web cluster that is made up of the following entities: a group of loosely-coupled Web-server nodes to respond to HTTP requests, an AFS distributed file system that manages the entire WWW document tree and one primary DNS for the entire Web-server system.

The DNS for the NCSA domain is modified in order to implement a round-robin algorithm for address mapping. The load distribution under the DNS-RR is not very balanced mainly because of the address caching mechanism that lets the DNS control only a very small fraction of the requests. This is further affected by the uneven distribution of client requests from different domains. So a large number of clients from a single domain can be assigned to the same Web-server, leading to load imbalance and overloaded server nodes [13, 18]. Additional drawbacks of this approach are related to the fact that the DNS-RR ignores both server capacity and availability. If a server is overloaded or out of order, there is no mechanism to stop the clients from continuing to try to access the Web site by using the cached address of that server. Moreover, the round-robin mechanism treats all servers equally, i.e. assuming a homogeneous server environment. The poor performance of the RR-DNS policy motivates the study of alternative DNS routing schemes that require additional system information to better balance the load of a Web-server cluster.

Server state based algorithms. Knowing server state conditions is mandatory if we want an available Web-server system that can exclude unreachable servers because of faults or congestion. It was found in [13] that the DNS policies combined with a simple feedback alarm mechanism from highly utilized servers is very effective to avoid overloading the Web-server system. This allows to exclude those servers from further request assignments during the overload period. A similar approach combined with the DNS-RR policy is implemented by the SunSCALR framework [26].

The scheduling decision of *lbmnamed* proposed in [25] is based on the current load on the Web-server nodes. When an address request reaches the DNS, it selects the server with the lightest load. To inhibit address caching at name servers, the DNS of the lbmnamed system sets the TTL value to zero. This choice imposes a limitation to the applicability of this approach as discussed in Section 4.3.

Client state based algorithms. Two main kinds of information can come from the client side: the typical load that arrives to the Web-server system from each connected domain, and the geographical location of the client.

A measure of the average number of data requests sent from each domain to a Web site during the TTL caching period following an address mapping request is referred to as the *hidden load weight* [13]. (A normalized hidden load weight represents the domain request rate.) Various DNS scheduling policies were proposed in [13] that mainly use this client information so as to assign the requests to the most appropriate server. In particular, these policies aim at identifying the requesting domain and the hidden load weight that this domain imposes on the server. One example of hidden load weight algorithm is the *multi-tier round-robin* policy, where different round-robin chains are used for requests in different ranges of hidden load weight.

One of the two alternative modes of the Cisco *DistributedDirector* [12] takes into account the location of the client/domain to make server selection. In this mode, DistributedDirector acts as a primary DNS. It basically takes relative client-to-server topological proximity and client-to-server link latency into account to determine the most suitable server. As an approximation, the client location is evaluated using the IP address of the client's local DNS server.

The Internet2 Distributed Storage Infrastructure Project (I2-DSI) proposes a smart DNS that implements address resolution criteria based on network proximity information, such as round trip delays [6].

These geographic DNS-based algorithms do not work if URL to IP address mapping is always cached by the intermediate name servers. To make these mechanisms work, the cluster DNS sets the TTL to zero. However, this solution has a limited effect because of many non-cooperative name servers.

Server and client state based algorithms. Most DNS algorithms achieve best results when they use both client and server state conditions. For example, the *DistributedDirector* DNS algorithm actually uses server availability information in addition to client proximity.

Similarly, the hidden load weight may not always be sufficient to predict the load conditions at each Web-server node. An asynchronous alarm feedback from over-utilized servers allows the design of new DNS policies that exclude those servers from request assignments during the overload period and use the client state based algorithms to select an eligible server from the non-overloaded servers [13]. As the enhanced DNS scheduling algorithms require various kinds of system state information, efficient state estimators are needed for their actual implementation. Some dynamic approaches for collecting state information at the DNS are proposed in [9].

4.2 Dynamic TTL algorithms

To balance the load across multiple Web-server nodes, the DNS has actually two control knobs: the policy for scheduling the server and the algorithm for the TTL value selection. Just exploring the scheduling component alone (*constant TTL* algorithms) is often inadequate to address client request skew and probable heterogeneity of server capacities. For this reason, a new class of *dynamic TTL* scheduling policies was proposed in [14]. They use some server and client state based DNS policy for the selection of the server, and dynamically adjust the TTL value based on different criteria.

The first set of algorithms (*variable TTL*) mainly addresses the problem of the limited control of the DNS. For this purpose, it increases the DNS control when there are many overloaded servers through a dynamic reduction of the TTL value. However, the strategies based on this approach do not work much better than constant TTL algorithms. Instead of just reducing the TTL value to give more control to the DNS, a better alternative is to address the unevenly distributed domain load and heterogeneous server capacities by assigning a different TTL value to each address request (*adaptive TTL* algorithms). The rationale for this approach comes from the observation that the hidden load weight, independently of the domain, increases with the TTL value. Therefore, by properly selecting the TTL value for each address request, the DNS can control the subsequent requests to reduce the load skews that are the main cause of overloading [14].

The *adaptive TTL* uses a two-step decision process. In the first step, the DNS selects the Web-server node similarly to the hidden load weight algorithms. In the second step, the DNS chooses the appropriate value for the TTL period. The uneven domain load distribution is handled by using TTL values inversely proportional to the domain request rate, i.e., the address requests coming from very popular domains will receive a TTL value lower than the requests originated by small domains. As the popular domains have higher domain request rate, a shorter TTL interval will even out the total number of subsequent requests generated. The (probable) system heterogeneity is addressed either during the server selection or, again, through adjusting the TTL values. The DNS can assign a higher TTL value when the DNS chooses a more powerful server, and a lower TTL value when the requests are routed to a less capable server. This is due to the fact that for the same fraction of server capacity, the more powerful server can handle a larger number of requests, or take requests for a longer TTL interval. Furthermore, the adaptive TTL approach can take server availability into account by using feedback mechanisms from the servers.

The adaptive TTL algorithms can easily scale from LAN to WAN distributed Web-server system because they only require information that can be dynamically gathered by the DNS [9], i.e., the request rate associated with each connected domain and the capacity of each server.

4.3 DNS-based architecture comparison

In [13] it was shown that the DNS policies such as *lbmnamed* based on detailed server state information (e.g., present and past load) are not effective in balancing the load across the servers. This is because with address caching, each address mapping can cause a burst of future requests to the selected server and make the current load information become obsolete quickly and poorly correlate with future load. The hidden load weight provides an estimate of the impact of each address mapping and is a more useful information to guide routing decisions.

Scheduling algorithms based on the hidden load weight and alarms from the overloaded servers can lead to much better load balancing than RR-DNS and maintain high Web site availability [13]. However, they give less satisfactory results when generalized to a heterogeneous Web-server system through probabilistic routing [14].

Adaptive TTL algorithms are the most robust and effective in balancing the load among distributed Web-server systems, even in the presence of skewed loads and non-cooperative name servers. However, they do not take the client-to-server distance into account in making scheduling decision. A policy that uses adaptive TTL assignment combined with information on geographical client location may achieve a better performance, even though no existing DNS-based system has yet considered this approach.

Policies, such as the DistributedDirector, I2-DSI and lbmnamed, that require the setting of the TTL value to zero have serious limitations on general applicability. They make DNS a potential bottleneck. Furthermore, they do not take the client-level address caching into account, because of which subsequent requests from the same client (browser) are sent to the same server. Some problems exist even at the network-level address caching because most intermediate name servers are configured in a way that do not accept very low TTL values.

5 Dispatcher-based approach

An alternative approach to DNS-based architectures aims to achieve full control on client requests and mask the request routing among multiple servers. To this purpose, they extend the address virtualization, done by the DNS-based approaches at the URL level, to the IP level. This approach provides the Web-server cluster with a *single virtual IP address* (IP-SVA). In fact, this is the IP address of a *dispatcher* that acts as a centralized scheduler and, in contrast to the DNS, has complete control on the routing of all client requests. To distribute the load among the Web-server nodes, the dispatcher is able to uniquely identify each server in the cluster through a *private address* that can be at different protocol levels depending on the proposed architecture. We differentiate the dispatcher-based architectures through the mechanism they use to route the requests reaching the dispatcher toward the 'hidden' selected Web-server. The two main classes of routing are through a packet rewriting mechanism or HTTP redirection.

The existing dispatcher-based architectures typically use simple algorithms for the selection of the Web-server (e.g., round-robin, server load) because the dispatcher has to manage all incoming flow and the amount of processing for each request has to be kept to minimum. However, it is possible to integrate this architecture with some more sophisticated assignment algorithms proposed for distributed Web-server systems having a centralized dispatcher with full control on client requests. An example is the *SITA-V* algorithm [16] that takes into account the heavy-tailed task size distributions [3, 15] in the selection of the appropriate server.

5.1 Packet single-rewriting by the dispatcher

We first consider architectures where the dispatcher reroutes client-to-server packets by rewriting their IP address. An example of this solution is the basic *TCP router* mechanism described in [18]. The distributed Web-server cluster is made up by a group of nodes and a TCP router that acts as an IP address dispatcher. The mechanism is outlined in Figure 3, where *address* i is now the private IP address of the i-th Web-server node. All HTTP client requests reach the TCP router because the IP-SVA is the only public address (step 1). The routing to a server is achieved by rewriting the destination IP address of each incoming packet (and also the TCP and IP checksums, since they both depend on the destination IP address): the dispatcher replaces its IP-SVA with the IP address of the selected server (step 3). The server choice for each HTTP request is done by the dispatcher through a round-robin algorithm (step 2). Since a request consists of several IP packets, the TCP router keeps track of the source IP address for every established TCP connection in an address table. In such a way, the TCP router can always route packets pertaining to the same connection to the same Web-server node (step 4).

Furthermore, the Web-server, before sending the response packets to the client (step 6), needs to replace its IP address with the IP-SVA of the dispatcher (step 5). Therefore, the client is not

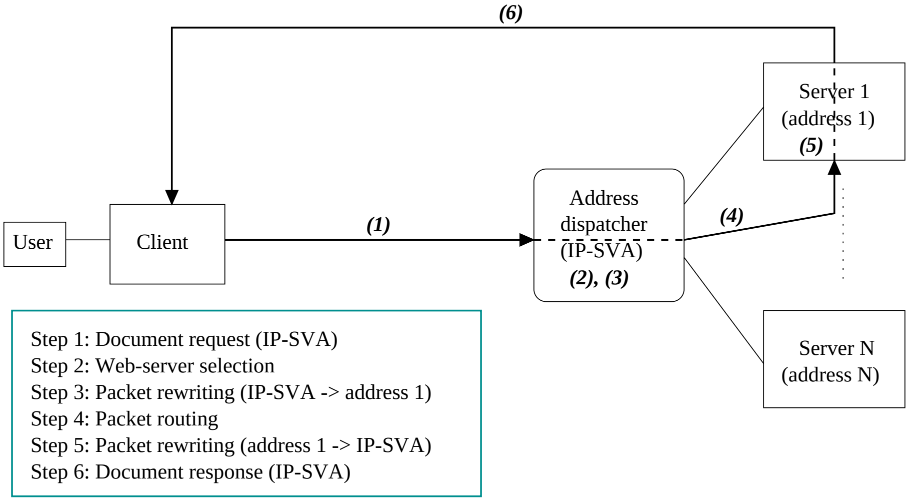

(6)
Server 1
(address 1)
(5)
User
Client
(1)
Address
dispatcher
(IP-SVA)
(2), (3)
(4)
Server N
(address N)
Step 1: Document request (IP-SVA)
Step 2: Web-server selection
Step 3: Packet rewriting (IP-SVA -> address 1)
Step 4: Packet routing
Step 5: Packet rewriting (address 1 -> IP-SVA)
Step 6: Document response (IP-SVA)

Figure 3: Packet single-rewriting by the dispatcher.

aware that its requests are handled by some hidden Web-server.

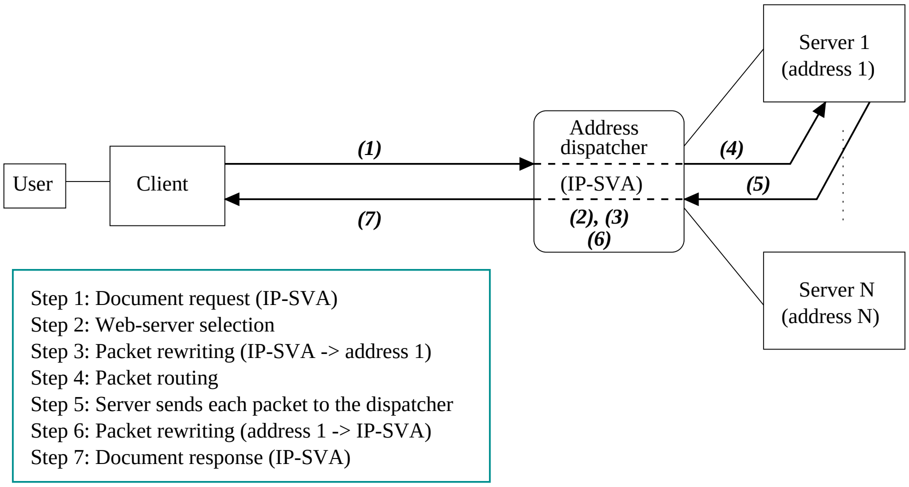
Although this solution is transparent to the client, the TCP router architecture does require modification of the kernel code of the servers, since the IP address substitution occurs at the TCP/IP level. On the other hand, this approach provides high system availability, because upon failure of a front-end node, its address can be removed from the router's table so that no client request will be routed to it anymore. The TCP router architecture can be combined with a DNS-based solution to scale from LAN to WAN distributed Web system.

5.2 Packet double-rewriting by the dispatcher

This class of Web-server clusters also relies on the presence of a centralized dispatcher that acquires all client requests and distributes them among the servers. The difference from the TCP router is in the modification of the source address of the server-to-client packets. Now, the modification of all IP addresses, including that in the response packets, are carried out by the dispatcher. This packet double-rewriting mechanism is at the basis of the Network Address Translation (NAT) defined in [19] and shown in Figure 4. When the dispatcher receives a client request, it selects the Web-server node (step 2) and modifies the IP header of each incoming packet (step 3). Furthermore, the dispatcher has to modify the outgoing packets that compose the requested Web document (step 6).

Two architectures that are based on this approach (with a server fault detection mechanism) are the *Magicrouter* [1] and the *LocalDirector* [11].

Magicrouter. Magicrouter uses a mechanism of *fast packet interposing*, where a user level process, acting as a switchboard, intercepts client-to-server and server-to-client packets and modifies them by changing the addresses and checksum fields. To balance the load among the Web-server nodes, three algorithms are considered: round-robin, random and incremental load, which is similar to selecting the lowest load server based on the current load estimate and



Step 1: Document request (IP-SVA)
Step 2: Web-server selection
Step 3: Packet rewriting (IP-SVA -> address 1)
Step 4: Packet routing
Step 5: Server sends each packet to the dispatcher
Step 6: Packet rewriting (address 1 -> IP-SVA)
Step 7: Document response (IP-SVA)

Figure 4: Packet double-rewriting by the dispatcher.

the per TCP connection load adjustment. Unlike the TCP router working at kernel level, the Magicrouter works at application level.

LocalDirector. The LocalDirector from Cisco Systems [11] rewrites the IP information header of each client-to-server packet according to a dynamic mapping table of connections between each session and the server to which it has been redirected. The routing policy selects the server with the least number of active connections.

5.3 Packet forwarding by the dispatcher

A different set of solutions uses the dispatcher to forward client packets to the servers instead of rewriting their IP addresses.

Network Dispatcher. An extension of the basic TCP router mechanism is provided by the IBM *Network Dispatcher* [22]. Let us distinguish its LAN implementation from WAN implementation.

The LAN Network Dispatcher solution assumes that the dispatcher and the server nodes are on the same local network. All them share the same IP-SVA address, however the client packets reach the Network Dispatcher because the Web nodes have disabled the ARP resolution mechanism. The dispatcher can forward these packets to the selected server using its physical address (MAC address) on the LAN without modifying the IP header. Unlike the basic TCP router mechanism, neither the dispatcher nor the Web-servers of the LAN Network Dispatcher are required to modify the IP header of the response packets. The mechanism is similar to that described in Figure 3. In this case, *address i* is the private hardware MAC address of the *i*-th Web-server node. This solution is transparent to both the client and server because it does not require packet rewriting. The scheduling policy used by the dispatcher can be dynamically based on server load and availability.

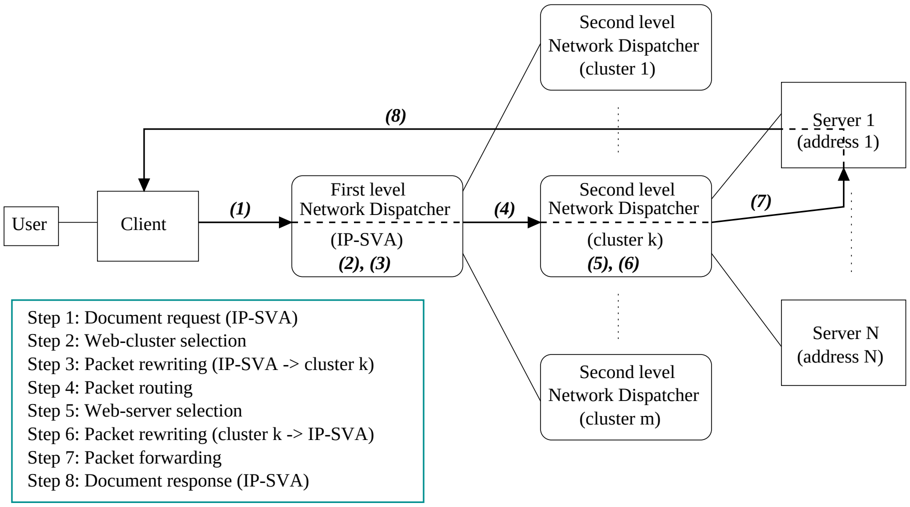


Figure 5: WAN Network Dispatcher architecture.

The extension of the Network Dispatcher to a WAN distributed architecture requires a different solution based on two levels of Network Dispatchers. The mechanism is outlined in Figure 5, where *cluster i* is the IP address of the second level dispatcher for the *i*-th cluster. The first level acts now quite similarly to a packet single-rewriting mechanism by the dispatcher, because it replaces the IP-SVA address with the IP address of the second level Network Dispatcher (step 2, 3) that coordinates the chosen cluster. The second level Network Dispatcher, in its turn, changes this IP address back to the original IP-SVA and selects a Web-server (step 5, 6). As the second level Network Dispatcher and the Web-server nodes of this cluster are on the same local network, the packet forwarding mechanism based on MAC address can be used to assign the client requests to one Web-server (step 7). This solution limits the rewriting by the dispatcher to the client-to-server packets that in a Web environment are typically much less than server-to-client packets, thereby solving the main problem of the other architectures in Section 5.2 that require packet rewriting in both directions.

ONE-IP address. A different kind of forwarding approach uses the `if config alias` option to configure a Web-server system with multiple machines [17]. This solution publicizes the same secondary IP address of all Web-server nodes as IP-SVA of the Web-server cluster (here called ONE-IP). Starting from this basic approach, two techniques are described in [17].

The *routing-based dispatching* requires that all the packets directed to the ONE-IP address are first rerouted by the subnetwork router to the IP address dispatcher of the distributed Web-server system. The dispatcher selects the destination server based on a hash function that maps the client IP address into the primary IP address of a server and then reroutes the packet to the selected server. This approach provides a static assignment, however it guarantees that all packets belonging to the same TCP connection are directed to the same server.

The *broadcast-based dispatching* uses a different mechanism to distributed client requests. The subnetwork router broadcasts the packets having ONE-IP as destination address to every server in the Web-server cluster. Each server evaluates whether it is the actual destination by applying a hash function to the client IP address and comparing the result with its assigned identifier.

Using a hash function to select the server based on the client IP address is the weak point of the ONE-IP address approach. Although the hash function could be dynamically modified to provide fault tolerance, this approach does not allow dynamic load balancing based on server load. Moreover, the proposed hash function does not take into account server heterogeneity. To further scale the system, the ONE-IP approach can be combined with a DNS-based solution.

5.4 HTTP redirection by the dispatcher

In the HTTP redirection approach a centralized dispatcher receives all incoming requests and distributes them among the Web-server nodes through the redirection mechanism provided by HTTP. This protocol allows the dispatcher to redirect a request by specifying the appropriate status code in the response and indicating in its header the server address to which the client can get the desired document. Such redirection is transparent to the users that can at most perceive an increase in the response time. Unlike most dispatcher-based solutions, the HTTP redirection does not require the modification of the IP address of the packets reaching or leaving the Web-server cluster. Two main proposals of this class are:

Server state based dispatching. The Distributed Server Groups approach [21] adds some new methods to HTTP protocol to administer the cluster and exchange messages between the dispatcher and the servers. Since the dispatcher needs to be aware of the load on the servers, each server periodically reports both the number of processes in its run queue and the number of received requests per second. Based on this information, the dispatcher selects the least loaded server.

Location based dispatching. The Cisco DistributedDirector [12] provides two alternative modes on location based dispatching. The former, discussed in Section 4.1, uses the DNS approach with client and server state information, while the latter takes the HTTP redirection approach. The DistributedDirector that acts as a dispatching facility uses some measure to estimate the proximity of a client to the servers and the node availability (as in the DNS case), determines the most suitable server for the request and redirects the client to that node.

5.5 Dispatcher-based architecture comparison

The solutions based on the packet double-rewriting by the dispatcher have the major disadvantage of requiring the dispatcher to rewrite not only the incoming but also outgoing packets (which are typically in larger number than the incoming request packets).

The packet single-rewriting by the dispatcher mechanism provided by the TCP router architecture has the same overhead of rewriting in both directions, however it reduces the bottleneck risks at the dispatcher because the more numerous server-to-client packets are rewritten by the Web-servers. Even more efficient is the WAN Network Dispatcher solution that rewrites (twice) only the client-to-server packets.

However, packet rewriting remains a big overhead. Packet forwarding mechanisms aim at addressing this issue. Problems of the ONE-IP approach can arise from the static scheduling algorithm which does not take the server state into account for the routing decision. While the *routing-based dispatching* requires double re-routing of each packet, the *broadcast-based dispatching* broadcasts all packets to every servers, thus causing more server overhead. This latter solution also requires the modification of each server device driver in order to properly filter the packets.

The LAN Network Dispatcher architecture avoids most of the network traffic problems of the ONE-IP solutions, and overheads of TCP router and double rewriting approaches. However, it lacks geographical scalability because it requires that the dispatcher and the Web-server nodes are directly connected by the same network segment.

The HTTP request redirection approach can be finely applied to LAN and WAN distributed Web-server systems, however it duplicates the number of necessary TCP connections. Furthermore, the Distributed Server Group technique requires new methods added to HTTP protocol for the communications among the dispatcher and the Web-server nodes.

6 Server-based approach

The server-based techniques use a two-level dispatching mechanism: client requests are initially assigned by the cluster DNS to the Web-servers; then, each server may reassign a received request to any other server of the cluster. Unlike the DNS-based and dispatcher-based centralized solutions, the distributed scheduling approach allows all servers to participate in balancing the load of the cluster through the request (re-)assignment mechanism. The integration of the DNS-based approach with some redirection techniques carried out by the Web-servers aims to solve most of the problems that affect DNS scheduling policies, e.g., the non-uniform distribution of client requests among the domains and limited control over the requests reaching the Web cluster.

The server-based proposals differ in the way the redirection decision is taken and implemented. We consider two main classes of solutions: those based on the packet rewriting mechanism, and those that take advantage of the redirection facility provided by the HTTP protocol.

6.1 HTTP redirection by the server

Scalable server World Wide Web (SWEB) system [2] and other architectures proposed in [10] use a two-level distributed scheduler (see Figure 6). Client requests, initially assigned from the DNS to a Web-server (step 1, 1', 2, 3, 3'), may be reassigned to any other server of the cluster through the redirection mechanism of the HTTP protocol. Figure 6 shows the case where the server 1 receiving the client request (step 4) decides to redirect the request (step 5, 6) to server 2 instead of serving it. As in Figure 2, we show that the first-level Web-server selection done by the DNS can be prevented by the intermediate name servers caching a valid address mapping.

The decision to serve or to redirect a request can be based on various criteria. A large set

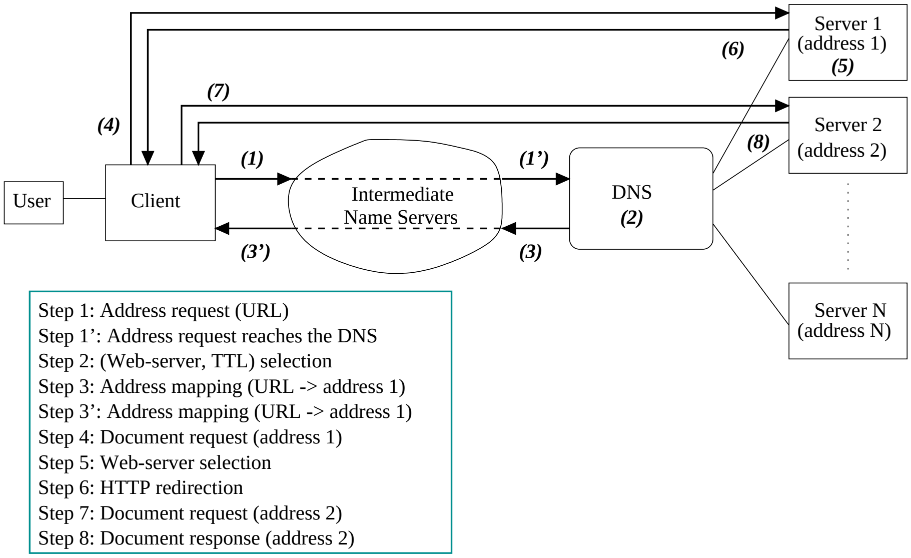
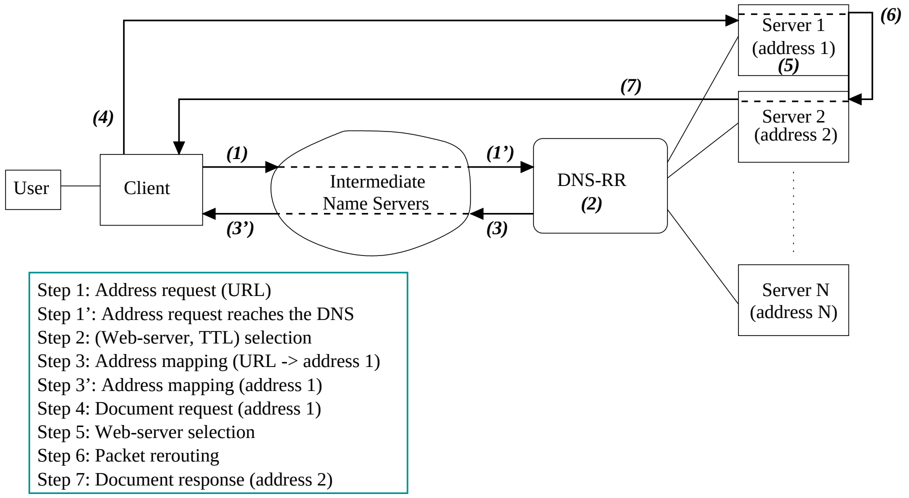

Server 1
(address 1)
(5)
(6)
(7)
Server 2
(address 2)
(4)
(8)
User
Client
(1)
Intermediate
Name Servers
(1')
DNS
(2)
(3')
(3)
Server N
(address N)
Step 1: Address request (URL)
Step 1': Address request reaches the DNS
Step 2: (Web-server, TTL) selection
Step 3: Address mapping (URL -> address 1)
Step 3': Address mapping (URL -> address 1)
Step 4: Document request (address 1)
Step 5: Web-server selection
Step 6: HTTP redirection
Step 7: Document request (address 2)
Step 8: Document response (address 2)

Figure 6: HTTP redirection by the server.

of alternative mechanisms, including *synchronous* (periodic) vs. *asynchronous* activation of the redirection mechanism, and granularity of the redirected entities (i.e., individual clients vs. entire domains) are discussed and compared in [10]. The asynchronous activation is triggered when the server chosen by the DNS considers that is more appropriate to let another server answer the client request, e.g. the former server is overloaded, or in the cluster there is a less loaded server and/or a server closer to the client domain. We have seen that the redirection of individual client connections is crucial to better balancing the load for asynchronous schemes, however it has to be combined with domain redirection when synchronous policies are adopted. Although these latter algorithms give the best results for a wide set of system parameters, the performance difference with the asynchronous approaches is not appreciable unless the Web-server cluster is subject to very heavy loads.

The SWEB architecture uses a round robin DNS policy as a first-level scheduler and a purely distributed asynchronous scheme as a second-level scheduler. Each Web-server decides about redirection on the basis of a server selection criterion that aims to minimize the response time of the client request. The estimation of this time value is done by taking the processing capabilities of the servers and Internet bandwidth/delay into account.

All the proposed re-assignment mechanisms imply an overhead of intra-cluster communications, as every server needs to periodically transmit some status information to the cluster DNS [10] or to other servers [2]. However, such cost has a negligible effect on the network traffic generated by the client requests. Indeed, on the user point of view, the main drawback of the HTTP redirection mechanism is that it increases the mean response time since each redirected request requires a new client-server connection.



Step 1: Address request (URL)
Step 1': Address request reaches the DNS
Step 2: (Web-server, TTL) selection
Step 3: Address mapping (URL -> address 1)
Step 3': Address mapping (address 1)
Step 4: Document request (address 1)
Step 5: Web-server selection
Step 6: Packet rerouting
Step 7: Document response (address 2)

Figure 7: Packet redirection by the server.

6.2 Packet redirection by the server

Distributed Packet Rewriting (DPR) uses a round robin DNS mechanism to carry out the first scheduling of the requests among the Web-servers [7]. The server reached by a client request is able to reroute the connection to another server through a packet rewriting mechanism that, unlike HTTP redirection, is quite transparent to the client. Figure 7 shows the initial request distribution achieved through the DNS (step 1, 1', 2, 3, 3') and the case where the server 1 decides to redirect the received request (step 5, 6) to the server 2 without affecting the client.

Two load balancing algorithms are proposed to spread clients requests. The first policy uses a static (stateless) routing function, where the destination server of each packet is determined by a hash function applied to both the sender's IP address and the port number. However, this simple policy is not practicable because IP packet fragmentation does not provide the port information in each fragment. The second stateful algorithm relies on periodic communications among servers about their current load. It tends to redirect the requests to the least loaded server. DPR can be applied to both LAN and WAN distributed Web-server systems. However, the packet rewriting and redirecting mechanism causes a delay that can be high in WAN distributed Web-server systems.

7 Comparison of the different approaches

7.1 Qualitative comparison

Load balancing is critical in managing high performance Web-server clusters. Request load must be spread across the Web-server nodes, so as to reduce the response time and provide WWW users with the best available quality of service. Load balancing among the servers can be achieved by several approaches with different degrees of effectiveness.

A summary of the policy features and main tradeoff of the various approaches is outlined in Table 1. Below we give additional explanations.

Client-based approach. The client-based approaches have the advantage of reducing the load on Web-server nodes by implementing the routing service at the client side. However, they lack general applicability since the client must be aware that the Web site is distributed.

DNS-based approach. Unlike other solutions, the DNS-based approaches do not directly route client requests. They only affect the destination of client requests through address mapping. However, due to address caching mechanisms, DNS can control only a very small fraction of address mapping requests. That is to say, the address caching permits only a coarse-grained load balancing mechanism if the assigned TTL is greater than zero. So the DNS scheduler needs to use sophisticated algorithms to achieve acceptable performance. They are typically based on additional state information such as hidden load weight from each domain, client location, and server load conditions. Furthermore, by adaptively setting the TTL value for each address mapping request, the performance can be greatly improved and applied to heterogeneous server environments.

The DNS-based architecture does not present risk of bottleneck and can be easily scaled from LAN to WAN distributed Web-server systems. On the other hand, this approach cannot use more than 32 Web-servers for each public URL because of UDP packet size constraints [22].

Dispatcher-based approach. The main drawback of dispatcher-based solutions comes from the presence of a single decision entity which can become a bottleneck when the system is subject to ever growing request rate. Furthermore, in a centralized approach the system can be disabled by the failure of the dispatcher. On the other hand, the dispatcher acting as a centralized scheduler with full control on the client requests can achieve fine grained load balancing. Furthermore, the single virtual IP address prevents problems arising from client or network level address caching that affect DNS-based solutions. As to server topology, it should be noted that the solutions that adopt the packet rewriting mechanism (with the exception of the WAN Network Dispatcher) is most applicable to server clusters on a LAN or high speed Intranet. Otherwise, the delay caused by the modification and rerouting of each IP packet that flows through the dispatcher can degrade the performance.

Server-based approach. The distributed scheduler solution proposed in the server-based architectures can provide scalability and does not introduce a single point of failure or potential bottleneck in the system. Furthermore, it can achieve the same fine grained control on the request assignment as the dispatcher-based solutions do. On the other hand, the redirection mechanism that characterizes the server-based architectures typically increases the latency time perceived by the users.

To face the high variability of both offered load and system configuration, the scheduling technique should be able to take the heterogeneity of the computing resources into consideration and to implement a dynamic assignment of incoming requests among the clustered nodes. An evaluation of the previously discussed scheduling mechanisms used by the various approaches is outlined in Table 2. The factors considered include control granularity, state information overhead, general applicability to popular Web sites subject to continue clients requests, bottleneck risk, geographi-

Approach	**Scheduling**	**Pros**	**Cons**
Client-based	client-side	no server overhead	limited applicability
	distributed	LAN and WAN solution	medium-coarse grained balancing
DNS-based	cluster-side	no bottleneck	partial control
	centralized	LAN and WAN solution	coarse grained balancing
Dispatcher-based	cluster-side	fine grained balancing	dispatcher bottleneck
	centralized	full control	(typically) LAN solution
			packet rewriting overhead
Server-based	cluster-side	distributed control	latency time increase (HTTP)
	distributed	fine grained balancing	packet rewriting overhead (DPR)
		LAN and WAN solution	

Table 1: Main pros and cons of the approaches.

cal scalability, and availability which is the ability to avoid routing requests to failed or overloaded servers in the cluster.

Approach	*Control granularity*	*State information overhead*	*General applicability*	*Bottleneck risk*	*Geographical scalability*	*Availability*
Client-based						
Netscape	coarse	no	no	no	yes	no
Smart Clients	fine	very high	no	no	yes	yes
Client-side proxies	fine	high	no	no	yes	yes
DNS-based						
RR-DNS	coarse	no	yes	no	yes	no
lbmnamed	medium	low	no (TTL=0)	medium	yes	no
SunSCALR	medium	low	yes	medium	yes	yes
Hidden load weight	coarse	high	yes	no	yes	yes
Geographic	medium	very high	no (TTL=0)	medium	yes	yes
Adaptive TTL	coarse	high	yes	no	yes	yes
Dispatcher-based						
TCP-router	fine	low	yes	high	no	yes
Magicrouter	fine	low	yes	very high	no	yes
LocalDirector	fine	medium	yes	very high	no	yes
Network Dispatcher (LAN)	fine	low	yes	high	no	yes
Network Dispatcher (WAN)	fine	low	yes	high	yes	yes
ONE-IP	fine	no	yes	high	no	partial
Distributed Server Groups	fine	high	yes	high	yes	yes
DistributedDirector	fine	high	yes	high	yes	yes
Server-based						
Synchronous, Asynchronous	fine	high	yes	no	yes	yes
SWEB	fine	high	no (TTL=0)	medium	yes	yes
DPR-stateless	fine	no	no	no	no	no
DPR-stateful	fine	high	yes	no	yes	yes

Table 2: Scheduling mechanisms.

7.2 Quantitative comparison

In the previous section we examine the load balancing characteristics of the different approaches with some brief analysis of the response time implication. A detailed quantitative analysis of the various features is beyond the scope of this paper. Here, we focus on investigating the impact of the scheduling algorithms on avoiding that some server node becomes overloaded, while others are underutilized. Indeed, the main goal of the Web site management is to minimize the worst case scenario, i.e., to avoid that some server node drops requests and eventually crashes. To this purpose, we define the *cluster maximum utilization* at a given instant as the highest server utilization at that instant among all servers in the cluster. Specifically, our performance criterion is the *cumulative frequency* of the cluster maximum utilization, i.e., the probability (or fraction of time) that the cluster maximum utilization is below a certain value. By focusing on the highest utilization among all Web-servers in the cluster, we can deduce whether some node of the Web-server cluster is overloaded. Hence, we developed a simulator to evaluate the performance of the various load balancing approaches by tracking at periodic intervals the cluster maximum utilizations observed during the simulation runs and plotting their cumulative frequencies as in [13].

The simulation experiments are carried out with an offered load equal to 2/3 of the cluster capacity. Since the performance are evaluated from the Web-server cluster's perspective, we did not model the Internet traffic, but we consider major WWW components that impact the performance of the Web-server clusters. These include the intermediate name servers and all the details concerning a client session, i.e., the entire period of access to the Web site from a single user. The clients have a (set of) local name server(s) and are connected to the network through gateways. We assume that clients are partitioned among these network sub-domains based on a Zipf's distribution. Two main models have been considered for representing the client load: the *exponential distribution model* and the *heavy-tailed distribution model.*

In the exponential distribution model, the number of page requests per session and the time between two page requests from the same client are assumed to be exponentially distributed. Analogously, the hit service time (i.e., the time to serve each object of an HTML page) and the inter-arrival time of hit requests to the servers are also assumed to be exponentially distributed. The parameters of the distributions and other details about the experiments are similar to those reported in [13].

On the other hand, the heavy-tailed distribution model can incorporate all main characteristics of real Web workload, and in particular the self-similar nature of Web traffic [15]. The high variability in the client load is represented through heavy-tailed functions, such as the Pareto and Weibull distributions. The adopted model following the Barford and Crovella results [5] is similar to that described in [9].

Below we consider some representative (ideal) version of the approaches described in the previous sections.

DNS-based approach. We implement two DNS-based algorithms: the constant TTL algorithm with server and client information, and the adaptive TTL algorithm. Moreover, for comparison purposes we consider even the DNS-RR used by NCSA. For all these approaches the percentage of requests that need to have address mapping resolved by the cluster DNS is kept below 5%.

Dispatcher-based approach. We consider a distributed Web cluster where the scheduler has full control on the incoming requests. Here we do not investigate the bottleneck issue, i.e., the dispatcher is assumed to have sufficient processing capacity. Nevertheless, in the real scenario, to reduce the likelihood that the dispatcher becoming a bottleneck, one needs to keep the amount of processing per request carried out by the dispatcher to a minimum. Thus, the load balancing algorithms cannot be too complicated. Typically, stateless algorithms such as round-robin or hash function, and simple algorithms such as least-loaded node are used. Our implementation of the dispatcher-based architecture uses round-robin which in our simulation experiment has demonstrated even better performance than the least-loaded node approach.

Served-based approach. The implemented server-based solution is a simplified version of the policies discussed in Section 6. A server node replies to a client request unless its load is over an alarm watermark. In such a case, it redirects the requests to the least loaded server in the cluster. This policy requires that each server be kept informed of the load on every other server. We observed in the simulation that the frequency of this information exchange should be rather high to achieve acceptable results.

For each of these approaches under the *exponential distribution model*, Figures 8 shows the cluster maximum utilization and its cumulative distribution on the x-axis and y-axis, respectively. Clearly, the (idealized) dispatcher-based architecture outperforms all other approaches because it always keeps the utilization of Web-server nodes below 0.8. Nevertheless, the DNS-based with adaptive TTL and the server-based policies also work quite well, because for all them the $Prob(ClusterMaxUtilization < 0.90)$ is almost 1, i.e., there is no overloaded server. On the other hand, the DNS-based architecture with constant TTL has at least one Web node overloaded (i.e., utilized above 0.90) for almost 20% of the time. However, both constant and adaptive TTL DNS-based architectures have performance results much better than the basic DNS-RR solution that causes at least one overloaded server for more than 70% of the time.

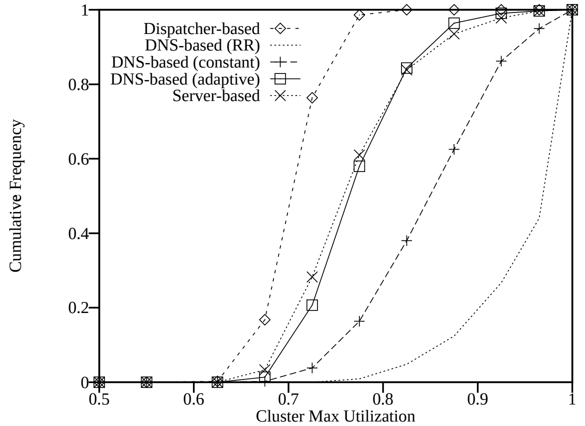


Figure 8: Performance of various distributed Web-server architectures (*exponential distribution model*).

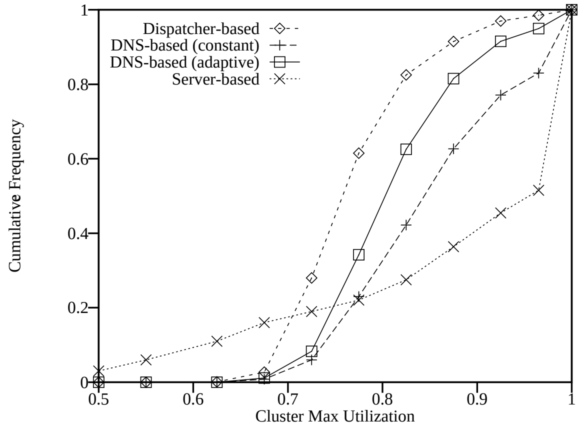


Figure 9: Performance of various distributed Web-server architectures (*heavy-tailed distribution model*).

Figures 9 shows the performance of the same architectures under the more realistic *heavy-tailed distribution model*. As expected, the results degrade for all approaches. Although the offered load in average is still kept to 2/3 of the cluster capacity, it can now have more frequent server overloading as most of the used distributions have infinite variance. Nevertheless, dispatcher-based and DNS-based with adaptive TTL continue to provide good performance. The DNS-based approach can be a realistic alternative to the dispatcher-based solution as it shows slightly worse results with no risks of bottleneck. The server-based approach has very poor performance (at least one server is overloaded for more than 50% of the time). However, we note that the implemented server-based approach shown here redirects the requests to the least loaded server. Although this policy works fine when the client load has a limited variability such as in the exponential distribution model, it becomes unacceptable when the past and future node load are poorly correlated as in the heavy-tailed distribution model. That is to say, to make the server-based architecture work even under this high variability scenario, more sophisticated scheduling policies have to be devised, such as those proposed in [10].

8 Conclusions

Load balancing is critical in operating high performance distributed Web-server systems. It can be achieved by various approaches with different degrees of effectiveness. In this paper, we have proposed a classification of existing solutions based on the entity that dispatches the client requests among the distributed Web-servers: client-based, DNS-based, dispatcher-based and server-based. The different techniques are evaluated primarily with respect to compatibility to Web standards, geographical scalability, and to what extent they achieve load balancing. We did not consider other Internet entities that may dispatch client requests, such as intelligent routers or intermediate name servers, because they are affected by the same problem that limits the portability of client-based approaches.

A more detailed quantitative comparison of the various architectures would be necessary to have major insights on the tradeoffs of the different mechanisms discussed in this paper. Furthermore, the performance constraints may often due to the network bandwidth than the server node capacity. Thus LAN distributed Web-server clusters are a limited solution to handle the increasing demand of client requests. The more effective solution is to distribute the Web-servers geographically so that they reside on separate networks. In such a case, the role of the dispatching algorithm can be ever more critical because the scheduler could then also take network load and client proximity into account when dispatching requests. A difficult issue that these approaches have to address is determining the client proximity and bandwidth availability in a dynamic way as they change very frequently in the Internet environment.

Acknowledgments

We thank the anonymous referees for their constructive comments and suggestions, which helped to improve the quality of the final paper. The first two authors acknowledge the support of the Ministry of University and Scientific Research (MURST) in the framework of the project “Design Methodologies and Tools of High Performance Systems for Distributed Applications” (MOSAICO).

References

[1] E. Anderson, D. Patterson, E. Brewer, "The Magicrouter, an application of fast packet interposing", University of California, Berkeley, May 1996. Available online at http://www.cs.berkeley.edu/~eanders/projects/magicrouter/osdi96-mr-submission.ps

[2] D. Andresen, T. Yang, V. Holmedahl, O.H. Ibarra, "SWEB: Toward a scalable World Wide Web-server on multicomputers", *Proc. of 10th IEEE Int'l. Symp. on Parallel Processing*, Honolulu, pp. 850–856, April 1996.

[3] M.F. Arlitt, C.L. Williamson, "Web-server workload characterization: The search for invariants", *IEEE/ACM Trans. on Networking*, vol. 5, no. 5, pp. 631–645, Oct. 1997.

[4] M. Baentsch, L. Baum, G. Molter, "Enhancing the Web's infrastructure: From caching to replication", *IEEE Internet Computing*, vol. 1, no. 2, pp. 18–27, Mar.-Apr. 1997.

[5] P. Barford, M. Crovella, "Generating representative Web workloads for network and server performance evaluation", *Proc. of ACM Sigmetrics '98*, Madison, WI, pp. 151-160, June 1998.

[6] M. Beck, T. Moore, "The Internet2 Distributed Storage Infrastructure project: An architecture for Internet content channels", *Proc. of 3rd Workshop on WWW Caching*, Manchester, England, June 1998.

[7] A. Bestavros, M. E. Crovella, J. Liu, D. Martin, "Distributed Packet Rewriting and its application to scalable Web server architectures", *Proc. of 6th IEEE Int'l. Conf. on Network Protocols* (ICNP'98), Austin, TX, Oct. 1998.

[8] P. Boyle, "Web site traffic cops: Load balancers can provide the busiest Web sites with nonstop performance", *PC magazine*, Feb. 1997.

[9] V. Cardellini, M. Colajanni, P.S. Yu, "DNS dispatching algorithms with state estimators for scalable Web-server clusters", *World Wide Web Journal*, Baltzer Science Publ., vol. 2, no. 3, pp. 101-113, July-Aug. 1999.

[10] V. Cardellini, M. Colajanni, P.S. Yu, "Redirection algorithms for load sharing in distributed Web-server systems", *Proc. of 19th IEEE Int'l. Conf. on Distributed Computing Systems* (ICDCS'99), Austin, TX, pp. 528-535, June 1999.

[11] Cisco's LocalDirector.
Available online at http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/index.shtml

[12] Cisco's DistributedDirector.
Available online at http://www.cisco.com/warp/public/cc/cisco/mkt/scale/distr/index.shtml

[13] M. Colajanni, P.S. Yu, D.M. Dias, "Analysis of task assignment policies in scalable distributed Web-server systems", *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, no. 6, pp. 585-600, June 1998.

[14] M. Colajanni, P.S. Yu, V. Cardellini, "Dynamic load balancing in geographically distributed heterogeneous Web-servers", *Proc. of 18th IEEE Int'l. Conf. on Distributed Computing Systems* (ICDCS'98), pp. 295-302, Amsterdam, The Netherlands, May 1998.

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

[16] M.E. Crovella, M. Harchol-Balter, C.D. Murta, "Task assignment in a distributed system: Improving performance by unbalancing load", *Proc. of ACM Sigmetrics '98*, Madison, WI, June 1998.

[17] O.P. Damani, P.Y. Chung, Y. Huang, C. Kintala, Y.-M. Wang, "ONE-IP: Techniques for hosting a service on a cluster of machines", *Journal of Computer Networks and ISDN Systems*, v. 29, pp. 1019-1027, Sept. 1997.

[18] D.M. Dias, W. Kish, R. Mukherjee, R. Tewari, "A scalable and highly available Web-server", *Proc. of 41st IEEE Computer Society Int'l. Conf.* (COMPCON'96), pp. 85–92, Feb. 1996.

[19] K. Egevang, P.Francis, "The IP Network Address Translator (NAT)", RFC 1631, May 1994. Available online at http://www.ietf.org/rfc/rfc1631.txt

[20] S.L. Garfinkel, "The wizard of Netscape", *WebServer Magazine*, pp. 58–64, July-Aug. 1996.

[21] M. Garland, S. Grassia, R. Monroe, S. Puri, "Implementing Distributed Server Groups for the World Wide Web", Tech. Rep. CMU-CS-95-114, Carnegie Mellon University, School of Computer Science, Jan. 1995.

[22] G.D.H. Hunt, G.S. Goldszmidt, R.P. King, R. Mukherjee, "Network Dispatcher: A connection router for scalable Internet services", *Proc. of 7th Int'l. World Wide Web Conf.* (WWW7), Brisbane, Australia, Apr. 1998.

[23] T.T. Kwan, R.E. McGrath, D.A. Reed, "NCSA's World Wide Web server: Design and performance", *IEEE Computer*, no. 11, pp. 68–74, Nov. 1995.

[24] D. Mosedale, W. Foss, R. McCool, "Lesson learned administering Netscape's Internet site", *IEEE Internet Computing*, vol. 1, no. 2, pp. 28–35, Mar.-Apr. 1997.

[25] R.J. Schemers, "lbmnamed: A load balancing name server in Perl", *Proc. of the 9th Systems Administration Conference* (LISA'95), Monterey, Sep. 1995.

[26] A. Singhai, S.-B. Lim, S.R. Radia, "The SunSCALR framework for Internet servers", *Proc. of IEEE Fault Tolerant Computing Systems* (FTCS'98), Munich, Germany, June 1998.

[27] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, D. Culler, "Using Smart Clients to build scalable services", *Proc. of Usenix 1997*, Jan. 1997.