FEATURE

DYNAMIC LOAD BALANCING ON WEB-SERVER SYSTEMS

Distributed architectures support increased loads on popular Web sites by dispatching client requests transparently among multiple server nodes. This article evaluates several possible approaches.

VALERIA CARDELLINI
University of Rome Tor Vergata
MICHELE COLAJANNI
University of Modena
PHILIP S. YU
IBM T.J. Watson Research Center

The overall increase in traffic on the World Wide Web causes a disproportionate increase in client requests to popular Web sites, especially in conjunction with special events, such as Olympic Games and the NASA Pathfinder. Site administrators constantly face the need to improve server capacity. One approach is to replicate information across a mirrored-server architecture. This load-balancing technique lets users manually select alternative URLs for a Web site. However, such architectures are not user-transparent, nor do they allow the Web-server system to control request distribution.

A more promising solution to load balancing is a distributed architecture that can route incoming requests transparently among several server nodes. Although this approach can improve throughput performance and Web-server system scalability, there are many challenges to making a distributed-server system function as a single server within the HTTP and Web browser framework.

A *distributed Web-server system* is any architecture consisting of multiple Web-server hosts, distributed on LANs and WANs, with a mechanism to spread incoming client requests among the servers. Each server in the system can respond to any client request. Information can be distributed among server nodes in two ways: content tree replication on a server's local disk, or information sharing by means of a distributed file system.

Replication works with both LAN and WAN Web-server systems; information sharing works only for LANs. Successful load-balancing approaches must be transparent to users, making a distributed system appear as a single host to the outside world. All the approaches we discuss

1089-7801/99/$10.00 ©1999 IEEE

do this. The mirrored-server-based architecture, on the other hand, is visible to users, which minimizes its usefulness and therefore is not one of the suggested solutions.

In this article we classify four distributed Web-server architectures according to the entity that distributes the incoming requests among the servers. The client-based solution we discuss requires software modification on the client side; the other three (DNS-based, dispatcher-based, server-based) affect one or more components of the Web-server system. We then evaluate the trade-offs among the alternative approaches and techniques.

CLIENT-BASED APPROACH

Document requests to popular Web sites can be routed from the client side in any replicated Web-server architecture even when the nodes are loosely (or not) coordinated. Routing to the Web cluster can be provided by Web clients or by client-side proxy servers. These approaches are not universally applicable, but we include them for completeness.

Web Clients

Web clients, if they are aware of the Web-server system's replicated servers, can actively route requests. After receiving a request, the Web client selects a node of the cluster and, after resolving the address mapping, submits the request to the selected node, which is then responsible for responding to the client. The same client can send another request and reach another server.

Netscape Communications' approach is one example of Web-client scheduling. The load-balancing mechanism for the Netscape Web-server system's multiple nodes is as follows. When a user accesses the Netscape home page (http://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. This approach, which has very limited practical applicability and is not scalable, might offer utility to corporate intranets.[1]

A second example of Web-client scheduling is via smart clients. Unlike the traditional approach that does not involve the Web client, this solution migrates server functionality to the client through a Java applet.[2] The increased network traffic due to the continued message exchanges among each applet and server node to monitor node states and network delays, however, is a major drawback. Moreover, although this solution provides scalability and availability, it lacks client-side portability.

Client-Side Proxies

From the Web cluster standpoint, *proxy servers* are similar to clients. The proxy server is an important Internet entity that can route client requests to Web-server nodes. Like all Web-client approaches, this one has limited applicability; however, Baentsch and colleagues describe an interesting method that combines caching and server replication.[3]

We do not further investigate proxy servers because any load balancing mechanism they carry out requires Internet component modification. Typically, the same institution or company that manages the distributed Web-server system does not control the modifications.

Document requests to popular Web sites can be routed from the client side in any replicated Web-server architecture.

DNS-BASED APPROACH

Distributed Web-server architectures that use request routing mechanisms on the cluster side are free of the problems of client-based approaches. Architecture transparency is typically obtained through a single virtual interface to the outside world, at least at the URL level. (Other approaches provide a single virtual interface even at the IP level, as we will explain).

The *cluster DNS*—the authoritative DNS server for the distributed Web system's nodes—translates the symbolic site name (URL) to the IP address of one server. This process allows the cluster DNS to implement many policies to select the appropriate server and spread client requests.

The DNS, however, has a limited control on the request reaching the Web cluster. Between the client and the cluster DNS, many intermediate name servers can cache the logical-name-to-IP-address mapping to reduce network traffic. Moreover, every Web client browser typically caches some address resolution.

Besides providing a node's IP address, the DNS also specifies a validity period (Time-To-Live, or TTL) for caching the result of the logical name resolution. When the TTL expires, the address-mapping request is forwarded to the cluster DNS for

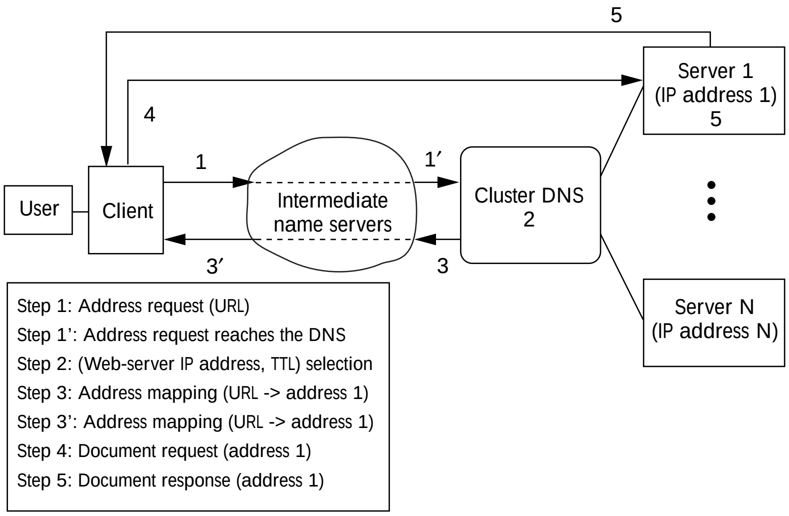

5
Server 1
(IP address 1)
5
4
1
1′
User
Client
Intermediate
name servers
Cluster DNS
2
3′
3
Server N
(IP address N)
Step 1: Address request (URL)
Step 1': Address request reaches the DNS
Step 2: (Web-server IP address, TTL) selection
Step 3: Address mapping (URL -> address 1)
Step 3': Address mapping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

Figure 1. DNS-based approach to load balancing.

assignment to a Web-server node; otherwise, an intermediate name server handles the request—Figure 1 shows both resolutions. This figure, like those in the following sections, show the different approaches for distributing requests on the basis of a protocol-centered description.

If an intermediate name server holds a valid mapping for the cluster URL, it resolves the address-mapping request without forwarding it to another name server. Otherwise, the address request reaches the cluster DNS, which selects the IP address of a Web server and the TTL. The URL-to-IP-address mapping and the TTL value are forwarded to all intermediate name servers along the path and to the client.

Several factors limit the DNS control on address caching. First, the TTL period does not work on the browser caching. Moreover, the DNS might be unable to reduce the TTL to values close to zero because of noncooperative intermediate name servers that ignore very small TTL periods. On the other hand, the limited control on client requests prevents the DNS from becoming a potential bottleneck.

We distinguish the DNS-based architectures by the scheduling algorithm that the cluster DNS uses to balance the Web-server nodes' load. With *constant TTL algorithms*, the DNS selects servers on the basis of system state information and assigns the same TTL value to all address-mapping requests. Alternatively, *adaptive TTL algorithms* adapt the TTL values on the basis of dynamic information from servers and/or clients.

Constant TTL Algorithms

These algorithms are classified by the system state information that the DNS uses to select a Web-server node—none, client load or client location, server load, or a combination.

System-stateless algorithms. The Round Robin DNS (RR-DNS) approach, first implemented by the National Center for Supercomputing Applications (NCSA) to handle increased traffic at its site, is for a distributed homogeneous Web-server architecture.[4] NCSA developed a Web cluster comprising the following entities: a group of loosely coupled Web-server nodes to respond to HTTP requests; a distributed file system that manages the entire WWW document tree; and one primary DNS for the entire Web-server system.

NCSA modified the primary DNS for its domain to map addresses by a round-robin algorithm. The load distribution under the RR-DNS is unbalanced because the address-caching mechanism lets the DNS control only a small fraction of requests. An uneven distribution of client requests from different domains further adds to the imbalance such that many clients from a single domain can be assigned to the same Web server, which overloads server nodes.[5,6]

Additional drawbacks result because the algorithm ignores both server capacity and availability. With an overloaded or nonoperational server, no mechanism can stop the clients from continuing to try to access the Web site by its cached address. The RR-DNS policy's poor performance needs research into alternative DNS routing schemes that require additional system information.

Server-state-based algorithms. Knowledge of server state conditions is essential for a highly available Web-server system to exclude servers that are unreachable because of faults or congestion. DNS policies, combined with a simple feedback alarm mechanism from highly utilized servers, effectively avoid Web-server system overload.[5] The SunSCALR framework implements a similar approach combined with the RR-DNS policy.[7]

Schemers based his proposed lbmnamed algorithm's scheduling decision[8] on the current Web-server nodes' load. The DNS, after receiving an address request, selects the least-loaded server. To inhibit address caching at name servers, the lbmnamed algorithm requires that the DNS sets the TTL value to zero. This choice limits applicability, as discussed later.

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

The *hidden load weight* index measures the average number of data requests sent from each connected domain to a Web site during the TTL caching period following an address-mapping request.[5] (A normalized hidden load weight represents the domain request rate.) Proposed DNS scheduling policies, which chiefly use this information to assign requests to the most appropriate server,[5] try to identify the requesting domain and the hidden load weight imposed by this domain. One example of this algorithm is the *multitier round-robin policy*, which uses different round-robin chains for requests in domains of different hidden load weights.

A commercially available alternative is Cisco Systems' DistributedDirector, an Internet appliance in the version that considers client/domain location before selecting a server. In this mode, the DistributedDirector acts as a primary DNS that determines the most suitable server on the basis of relative client-to-server topological proximity and client-to-server link latency. The DistributedDirector evaluates the approximate client location using the IP address of the client's local DNS server.

A third alternative is the Internet2 Distributed Storage Infrastructure Project (I2-DSI), which proposes a smart DNS that implements address resolution criteria on the basis of network proximity information, such as round-trip delays.[9]

These geographic DNS-based algorithms do not work well if URL-to-IP-address mapping is always cached by the intermediate name servers. To make them work, the cluster DNS sets the TTL to zero. However, this solution is limited by noncooperative name servers.

Server- and client-state-based algorithms. DNS algorithms are most effective when they use both client and server state conditions. For example, the DistributedDirector DNS algorithm uses server availability information along with client proximity. Similarly, the hidden load weight may be insufficient to predict the load conditions at each Web-server node. An asynchronous alarm feedback from overutilized servers lets newer DNS policies exclude those servers from request assignments during overload conditions and instead use the client-state-based algorithms to select an eligible server from the nonoverloaded servers.[5] To acquire the system state information needed by the enhanced DNS scheduling algorithms requires efficient state estimators, and several dynamic approaches have been suggested.[10]

DNS algorithms are most effective when they use both client and server state conditions.

Adaptive TTL Algorithms

To balance the load across multiple Web-server nodes, the DNS can exert control through both the policy for server scheduling and the algorithm for selecting the TTL value. Constant TTL algorithms cannot adequately address client request skew and probable heterogeneity of server capacities, so we have devised dynamic (adaptive) TTL scheduling policies.[10] The algorithms use some server- and client-state-based DNS policy to select the server, and dynamically adjust the TTL value.

The best alternative addresses the unevenly distributed domain load and heterogeneous server capacities by assigning a different TTL value to each address request. The rationale is 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 primarily cause overloading.[10]

Adaptive TTL algorithms use a two-step decision process. First, the DNS selects the Web-server node similarly to the hidden load weight algorithms. Second, the DNS chooses the appropriate value for the TTL period. To resolve the uneven domain load distribution, 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 a higher domain request rate, a shorter TTL interval evens out the total requests generated.

Adaptive TTL algorithms can easily scale from LANs to WANs because they require only information that can be dynamically gathered by the

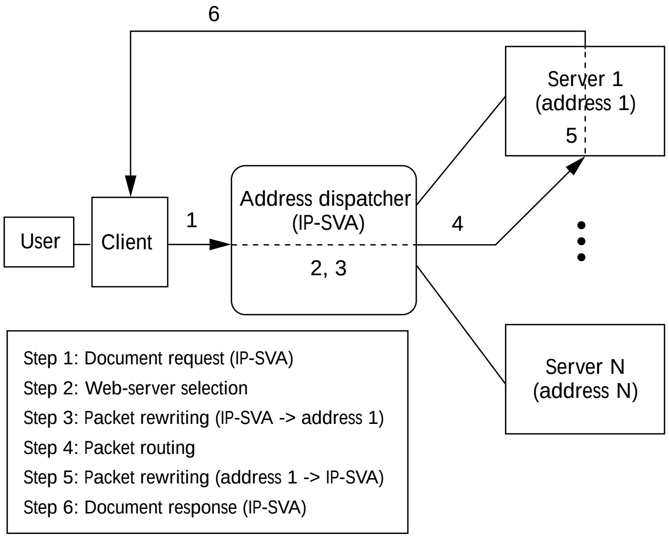

6
Server 1
(address 1)
5
User
Client
1
Address dispatcher
(IP-SVA)
2, 3
4
Server N
(address N)
Step 1: Document request (IP-SVA)
Step 2: Web-server selection
Step 3: Packet rewriting (IP-SVA -> address 1)
Step 4: Packet routing
Step 5: Packet rewriting (address 1 -> IP-SVA)
Step 6: Document response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

Figure 2. Packet single-rewriting by the dispatcher.

DNS; namely, the request rate associated with each connected domain and the capacity of each server.[10]

DNS-Based Architecture Comparison

DNS policies based on detailed server state information (for example, present and past load) do not effectively balance client requests across servers.[5] The policies are ineffective because with address caching, each address mapping can cause a burst of future requests to the selected server and quickly obsolete the current load information. The domain request rate estimates the impact of each address mapping and is more useful to guide routing decisions.

Scheduling algorithms based on the domain request rate and alarms from overloaded servers can lead to better load balancing than RR-DNS and maintain high Web site availability.[5] However, they give less satisfactory results when generalized to a heterogeneous Web-server system through probabilistic routing.

To balance requests among distributed Web-server systems, adaptive TTL algorithms are the most robust and effective, despite skewed loads and noncooperative name servers. The algorithms, however, do not consider the client-to-server distance in making scheduling decisions. A policy that uses adaptive TTL assignment combined with information on geographical client location may perform better, although no DNS-based system has yet considered this approach.

Policies requiring a zero TTL value, such as lbmnamed and that used by the DistributedDirector and in the I2-DSI, seriously limit general applicability and make DNS a potential bottleneck. Furthermore, such policies do not consider the client-level address caching, resulting in subsequent requests from the same client (browser) being sent to the same server. Problems exist even at the network-level of address caching because most intermediate name servers are configured such that they reject very low TTL values.

DISPATCHER-BASED APPROACH

To centralize request scheduling and completely control client-request routing, a network component of the Web-server system acts as a dispatcher. Request routing among servers is transparent—unlike DNS-based architectures, which deal with addresses at the URL level, the dispatcher has a single, virtual IP address (IP-SVA).

The dispatcher uniquely identifies each server in the system through a private address that can be at different protocol levels, depending on the architecture. We differentiate dispatcher-based architectures by routing mechanism—packet rewriting (single-rewriting or double-rewriting), packet forwarding, or HTTP redirection.

Dispatcher-based architectures typically use simple algorithms to select the Web server (for example, round-robin, server load) to manage incoming requests, as simple algorithms help minimize request processing.

Packet Single-Rewriting

In some architectures the dispatcher reroutes client-to-server packets by rewriting their IP address, such as in the basic TCP router mechanism.[6] The Web-server cluster consists of a group of nodes and a TCP router that acts as an IP address dispatcher. Figure 2 outlines the mechanism, in which address *i* becomes 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. The dispatcher selects a server for each HTTP request through a round-robin algorithm and achieves routing by rewriting each incoming packet's destination IP address. The dispatcher replaces its IP-SVA with the selected server's IP address. Because a request consists of several IP packets, the TCP router tracks the source IP address for every established TCP connection in an address table. The TCP router can thereby always route packets regarding the same connection to the same Web-server node.

Furthermore, the Web server must replace its IP address with the dispatcher's IP-SVA before sending the response packets to the client. Therefore, the client is not aware that its requests are handled by a hidden Web server.

This approach provides high system availability because, when a front-end node fails, its address can be removed from the router's table to prevent further request routing. Moreover, the TCP router architecture can be combined with a DNS-based solution to scale from a LAN- to a WAN-distributed Web system.

Packet Double-Rewriting

This mechanism also relies on a centralized dispatcher to schedule and control client requests but differs from packet single-rewriting in the source address modification of the server-to-client packets. Instead, the dispatcher modifies all IP addresses, including those in the response packets.

Packet double-rewriting underlies the Internet Engineering Task Force's Network Address Translator, shown in Figure 3. The dispatcher receives a client request, selects the Web-server node and modifies each incoming packet's IP header, and also modifies the outgoing packets that compose the requested Web document.

Two solutions using this approach (with a server fault-detection mechanism) are the Magicrouter[11] and Cisco Systems' LocalDirector. The Magicrouter architecture uses a fast packet interposing mechanism. 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, Magicrouter can use a round-robin, random, or incremental load algorithm. The last is similar to selecting the lowest-load server based on the current load estimate and the per-TCP-connection load adjustment.

The LocalDirector Internet appliance 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.

Packet Forwarding

Other solutions use the dispatcher to forward client packets to the servers instead of rewriting their IP addresses.

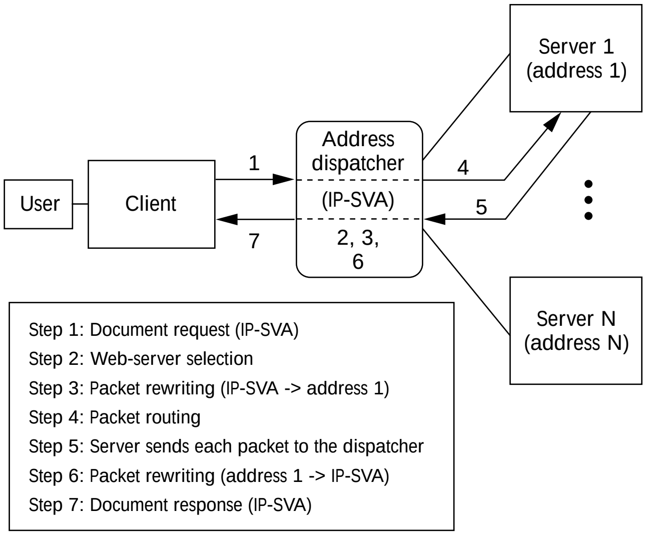

Server 1
(address 1)
Address
dispatcher
(IP-SVA)
2, 3,
6
User
Client
1
7
4
5
Server N
(address N)
Step 1: Document request (IP-SVA)
Step 2: Web-server selection
Step 3: Packet rewriting (IP-SVA -> address 1)
Step 4: Packet routing
Step 5: Server sends each packet to the dispatcher
Step 6: Packet rewriting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

Figure 3. Packet double-rewriting by the dispatcher.

Network Dispatcher. IBM's Network Dispatcher[12] extends the basic TCP router mechanism. The Network Dispatcher works with both LANs and WANs. The LAN Network Dispatcher assumes that the dispatcher and the server nodes are on the same local network. All share an IP-SVA address; however, the client packets reach the dispatcher because the Web nodes have disabled the Address Resolution Protocol (ARP) mechanism. The dispatcher can forward these packets to the selected server using its physical (MAC) address on the LAN without IP header modification. Unlike the basic TCP router mechanism, neither the LAN Network Dispatcher nor its Web servers need modify the response packets' IP header. With this mechanism, similar to that shown in Figure 2, address *i* is the private hardware MAC address of the *i*-th Web-server node. This solution is both client- and server-transparent because it does not require packet rewriting. The dispatcher's scheduling policy can be dynamically based on server load and availability.

Extending the dispatcher to a WAN-distributed architecture requires two levels. In Figure 4 (next page), cluster *i* is the IP address of the second-level dispatcher for the *i*-th cluster. The first level acts like a packet single-rewriting mechanism because it replaces the IP-SVA address with the IP address of the second-level Network Dispatcher that coordinates the chosen cluster.

The second-level dispatcher then changes this IP address back to the original IP-SVA and selects a Web server. As the second-level Network Dis-

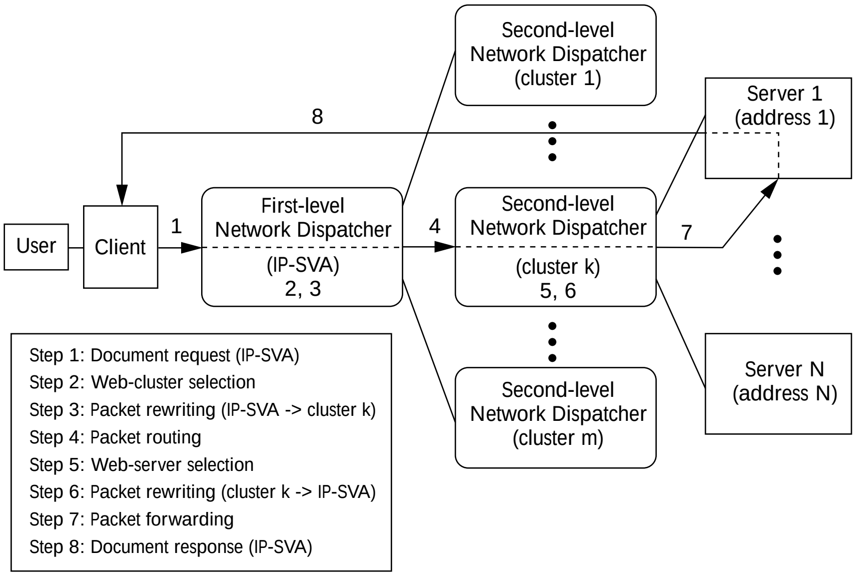


Figure 4. WAN Network Dispatcher architecture.

patcher and the Web-server nodes of this cluster are on the same LAN, the packet forwarding mechanism based on the MAC address can assign the client requests to one Web server. This solution limits the dispatcher's rewriting to the client-to-server packets that in a Web environment are typically much smaller than server-to-client packets. This solves most dispatcher-based architectures, which require packet rewriting in both directions.

ONE-IP address. Another forwarding approach uses the if config alias option to configure a Web-server system with multiple machines.[13] This solution publicizes the Web-server system's IP-SVA as the same secondary IP address of all nodes and can be implemented with two techniques.[13]

Routing-based dispatching requires that all 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 a server's primary IP address, 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.

Broadcast-based dispatching requires that the subnetwork router broadcast the packets having ONE-IP as a destination address to every server in the Web system. 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.

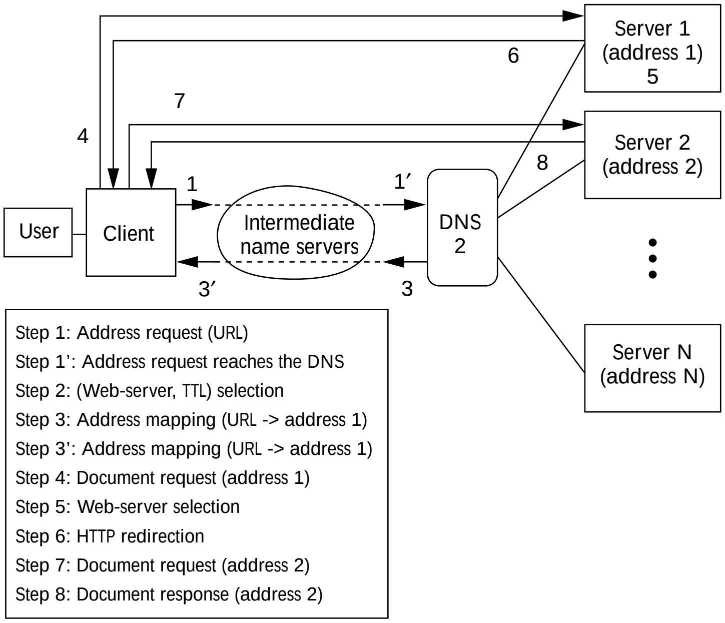
Using a hash function to select the server based on the client IP address is the weak point of the ONE-IP approach. Although the hash function could be dynamically modified to provide fault tolerance, this approach disallows dynamic load balancing based on server load. To further scale the system, the ONE-IP approach can be combined with a DNS-based solution.

HTTP Redirection

A centralized dispatcher receives all incoming requests and distributes them among the Web-server nodes through the HTTP's redirection mechanism. The dispatcher redirects a request by specifying the appropriate status code in the response, indicating in its header the server address where the client can get the desired document. Such redirection is largely transparent; at most, users might notice an increased response time. Unlike most dispatcher-based solutions, HTTP redirection does not require IP address modification of packets reaching or leaving the Web-server system. HTTP redirection can be implemented with two techniques.

Server-state-based dispatching, used by the Distributed Server Groups architecture,[14] adds new methods to HTTP protocol to administer the Web system and exchange messages between the dispatcher and the servers. Since the dispatcher must be aware of the server load, each server periodically reports the number of processes in its run queue and the number of received requests per second. The dispatcher then selects the least-loaded server.

Location-based dispatching, used by Cisco Systems' DistributedDirector appliance, provides two dispatching modes. The first applies the DNS-based approach with client and server state information; the second, HTTP redirection. The DistributedDirector estimates a client's server proximity and the node availability with algorithms that apply to the DNS-based solution. Client requests are redirected to the server that is evaluated as most suitable for each request at a certain time.



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 5. HTTP redirection by the server.

Dispatcher-Based Architecture Comparison

Packet double-rewriting by the dispatcher presents problems because the dispatcher must rewrite incoming as well as outgoing packets, and outgoing packets typically outnumber incoming request packets.

Packet single-rewriting, which the TCP router architecture uses, sustains the same overhead of rewriting in both directions but it reduces dispatcher bottlenecks because the Web servers rewrite the more numerous server-to-client packets. The WAN Network Dispatcher’s more efficient solution rewrites (twice) only the client-to-server packets.

Packet-forwarding mechanisms are an effort to resolve the overhead of packet rewriting. The ONE-IP approach, however, can have problems with the static scheduling algorithm, which does not consider the server state for routing decisions. While routing-based dispatching requires double rerouting of each packet, broadcast-based dispatching broadcasts all packets to every server, thus causing even higher server overhead.

The LAN Network Dispatcher architecture avoids most of ONE-IP’s traffic problems, and TCP router and double-rewriting overheads, but it lacks geographical scalability as it requires the same network segment to connect the dispatcher and the Web-server nodes.

HTTP redirection can be finely applied to LANs and WANs, but it duplicates the number of necessary TCP connections.

SERVER-BASED APPROACH

These techniques use a two-level dispatching mechanism. The primary DNS of the Web system initially assigns client requests to the Web-server nodes; then, each server may reassign a received request to any other system server. Unlike the DNS-based and dispatcher-based centralized solutions, the distributed scheduling approach lets all servers participate in load balancing the system through the request reassignment mechanism. Integrating the DNS-based approach with redirection techniques executed by the Web servers solves most DNS scheduling problems, such as unevenly distributed client requests among the domains and limited control over the requests reaching the Web system.

Server-based proposals differ in redirection decision implementation. One solution involves HTTP redirection; the other, packet redirection.

HTTP Redirection

The scalable server World Wide Web (SWEB) system[15] and similar architectures[16] use a two-level distributed scheduler, as shown in Figure 5. Client requests, initially assigned by the DNS to a Web server, can be reassigned to another server via HTTP redirection. Figure 5 shows server 1 receiving the client request, then redirecting the request to server 2. As was also shown in Figure 1, the first-level Web server selected by the DNS can be prevented by the intermediate name servers’ caching a valid address mapping.

A request is served or redirected depending on several factors. Redirection mechanisms are synchronously or asynchronously activated, and redirected entities can be individual clients or entire domains.[16] Asynchronous activation occurs when the DNS-selected server determines that another server would better answer the client request—perhaps one server is overloaded, or another server is closer to the client domain.

Redirecting individual client connections is crucial to better load balancing at a fine granularity level.

Table 1. Pros and cons of load-balancing approaches.

Approach	Scheduling	Pros	Cons
Client-based	Client side	No server overhead	Limited applicability
	Distributed	LAN and WAN solution	Medium-coarse-grained balancing
DNS-based	Web system side	No bottleneck	Partial control
	Centralized	LAN and WAN solution	Coarse-grained balancing
Dispatcher-based	Web system side	Fine-grained balancing	Dispatcher bottleneck
	Centralized	Full control	(typically) LAN solution packet rewriting overhead
Server-based	Web system side	Distributed control	Latency time increase (HTTP)
	Distributed	Fine-grained balancing	Packet-rewriting overhead (DPR)
		LAN and WAN solution	

In most instances, however, it is preferable to combine client redirection with domain redirection.[16]

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 redirects requests according to server selection that minimizes the client request's response time, a value estimated on the basis of server processing capabilities and Internet bandwidth/delay.

These mechanisms imply an overhead of intracluster communications, as every server must periodically transmit status information to the cluster DNS[16] or other servers,[15] but such cost only negligibly affects client-request-generated network traffic. To users, HTTP redirection's main drawback is increased response time, since each redirected request requires a new client-server connection.

Packet Redirection

Distributed Packet Rewriting (DPR) by the server uses a round-robin DNS mechanism to schedule the requests among the Web servers.[17] The server reached by a request reroutes the connection to another server through a packet-rewriting mechanism that, unlike HTTP redirection, is transparent to the client.

Two load-balancing algorithms spread client requests. The first uses static (stateless) routing, where a hash function applied to both the sender's IP address and the port number determines each packet's destination server. However, this simple policy is impractical because IP packet fragmentation does not provide the port information in each fragment.

The second algorithm uses periodic server communications to determine the servers' current load. It typically redirects the requests to the least-loaded server. DPR can be applied to both LAN- and WAN-distributed Web-server systems, but the packet-rewriting and -redirecting mechanism causes a delay that can be significant in WAN-distributed Web-server systems.

COMPARING THE APPROACHES

Table 1 outlines the features and trade-offs of the various approaches we have discussed.

Approach Trade-off Summary

Client-based approaches, which reduce Web-server loads by implementing routing at the client side, lack general applicability because the client must be aware that the Web site is distributed.

DNS-based approaches minimize bottlenecks and can be easily scaled from LAN- to WAN-distributed Web-server systems. This approach cannot, however, use more than 32 Web servers for each public URL because of UDP packet size constraints.[12] DNS-based architectures determine the destination of client requests through address mapping. Address caching, however, permits only a coarse-grained load-balancing mechanism if the assigned TTL is greater than zero, so the DNS scheduler requires sophisticated algorithms to achieve acceptable performance. Algorithms are typically based on additional state information, such as hidden load weight from each domain, client location, and server load conditions. Furthermore, adaptively setting the TTL value for each address-mapping request greatly improves the performance applied to heterogeneous server environments.

Dispatcher-based approaches are hampered by a single-decision entity, which can be a bottleneck with increased requests, and in centralized scheduling, a failed dispatcher can disable the system. The dispatcher can, however, achieve fine-grained load balancing, and the single virtual IP address prevents client- or network-level address caching problems that affect DNS-based solutions. Regarding server topology, solutions that adopt the packet rewriting mechanism (except for the WAN Network Dispatcher) are 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 performance.

In the server-based approach, distributed scheduling provides scalability without introducing a single point of failure or potential bottleneck. It also achieves the same fine-grained control on request assignments as dispatcher-based solutions. The characteristic server-based redirection mechanism, however, typically increases users' perceived latency.

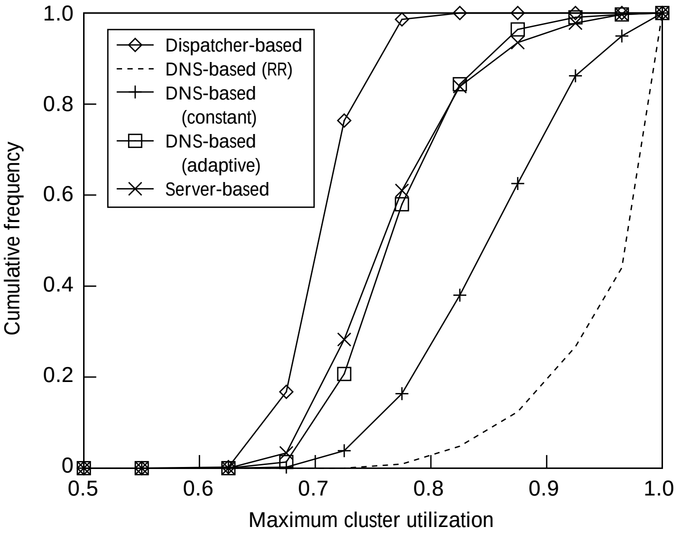


Figure 6. Exponential distribution model showing performance of distributed Web-server architectures.

Performance Evaluation

Our performance criterion is the cumulative frequency of *maximum cluster utilization*—the probability (or fraction of time) that the maximum cluster utilization is below a certain value. By focusing on the highest utilization among all Web servers in the cluster, we can determine if the Web-server system is overloaded.

We developed a simulator to run experiments so that we could evaluate the load-balancing performance of the various approaches. We did this by tracking the maximum cluster utilization observed, at intervals, and plotting their cumulative frequencies.[5]

We conducted the simulation experiments with an average offered load equal to two-thirds of the cluster capacity. Since we evaluated performance from the Web-server cluster's perspective, we didn't model the Internet traffic but did consider major Web components affecting the Web-server cluster performance. Components included the intermediate name servers and any details concerning a given client session. We assumed that the clients, having local name servers and connecting to the network through gateways, were partitioned among these network subdomains based on a Zipf distribution. We considered both the exponential distribution model and the heavy-tailed distribution model to represent the client load.

In the exponential distribution model, the number of page requests per session and the time between two page requests from the same client were assumed to be exponentially distributed. The hit service time (time to serve each object of an HTML page) and the interarrival time of hit requests to the servers were also assumed to be exponentially distributed.[5]

The heavy-tailed distribution model incorporated a real Web workload's key characteristics, particularly those concerning Web traffic. The high client-load variability was represented through heavy-tailed functions, such as the Pareto and Weibull distributions. We based our workload model on Barford and Crovella's work.[18,10]

Our simulations represented best-case scenarios for the DNS-based, dispatcher-based, and server-based load-balancing approaches. To evaluate the DNS-based approach, we implemented the constant TTL algorithm with server and client information, and the adaptive TTL algorithm. We also considered the RR-DNS solution. We kept the percentage of requests needing address mapping resolved by the cluster DNS to below 5 percent.

To evaluate the dispatcher-based approach, we simulated a distributed Web cluster in which the scheduler controls incoming requests. We assumed the dispatcher to have sufficient processing capacity to implement any algorithm. However, in reality, to avoid bottlenecks, system administrators would minimize the dispatcher's request processing using simple algorithms such as round-robin, hash functions, or least-loaded node. Our simulation used a round-robin algorithm, which demonstrated better performance than even the least-loaded node approach.

To evaluate the server-based approach, we implemented a simplified version of the HTTP

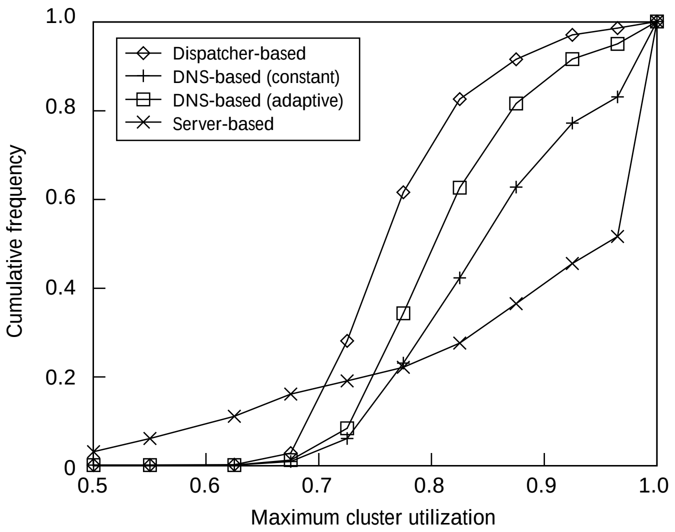


Figure 7. Heavy-tailed distribution model showing performance of distributed Web-server architectures.

and packet redirection policies. A server node replies to a client request unless its load exceeds a predetermined limit and sets off an alarm. If so, the server 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. The information exchange frequency must be fairly high for this to succeed.

For each approach under the exponential distribution model, Figure 6 shows the maximum cluster utilization and its cumulative distribution on the *x*-axis and *y*-axis, respectively. The idealized, dispatcher-based architecture clearly outperforms all other approaches because it keeps the Web-server node utilization below 0.8.

Both the DNS-based approach with adaptive TTL and the server-based policies also work well, because the maximum cluster utilization is below 0.90 with a probability close to 1—no server is overloaded. On the other hand, the DNS-based architecture with constant TTL has at least one overloaded Web node (utilized above 0.90) for almost 20% of the time. However, both constant and adaptive TTL DNS-based architectures perform much better than the basic RR-DNS solution that overloads at least one server node more than 70 percent of the time.

Figure 7 shows the architectures' performance under the more realistic heavy-tailed distribution model. As expected, the results degrade for all approaches. Although the average offered load is still about two-thirds of the cluster capacity, it can afford more frequent server overloading because most of the distributions we used have infinite variance. Although both approaches perform well, the DNS-based adaptive TTL approach is a realistic alternative to the dispatcher-based solution; its slightly worse results carry no bottleneck risks.

The server-based approach performs poorly, with at least one server overloaded more than half the time, where our simulation implementation redirected requests to the least-loaded server. This policy works fine when the client load has a limited variability, such as in the exponential distribution model, but is unacceptable when the past and future node load are poorly correlated as in the heavy-tailed distribution model. To be useful, this architecture needs more sophisticated scheduling policies.[16]

CONCLUSIONS

An in-depth understanding of the trade-offs posed by the approaches we have examined requires a more detailed quantitative comparison of the architectures. Furthermore, network bandwidth—more so than server node capacity—can constrain load-balancing performance. LAN-distributed Web-server clusters are thus a limited solution to increased client requests. A more effective approach requires geographically distributed Web-server nodes residing on separate networks. In these instances, the dispatching algorithm must take network load and client proximity into account when distributing requests. One difficulty that these approaches must address is in dynamically evaluating such information, as it varies frequently in the Internet environment. ■

ACKNOWLEDGMENTS

We thank the referees for their constructive comments and suggestions. The first two authors are supported by the Italian Ministry of University and Scientific Research in the project framework of "Design Methodologies and Tools of High Performance Systems for Distributed Applications."

REFERENCES

1. D. Mosedale, W. Foss, and R. McCool, "Lessons Learned Administering Netscape's Internet Site," *IEEE Internet Computing*, Vol. 1, No. 2, Mar.-Apr. 1997, pp. 28–35.
2. C. Yoshikawa et al., "Using Smart Clients to Build Scalable Services," *Proc. Usenix 1997*, Usenix Assoc., Berkeley, Calif., Jan. 1997.
3. M. Baentsch, L. Baum, and G. Molter, "Enhancing the Web's Infrastructure: From Caching to Replication," *IEEE Internet Computing*, Vol. 1, No. 2, Mar.-Apr. 1997, pp. 18–27.

4. T.T. Kwan, R.E. McGrath, and D.A. Reed, “NCSA’s World Wide Web server: Design and Performance,” *Computer*, Vol. 28, No. 11, Nov. 1995, pp. 68–74.
5. 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*, Vol. 9, No. 6, June 1998, pp. 585–600.
6. D.M. Dias et al., “A Scalable and Highly Available Web-Server,” *Proc. 41st IEEE Computer Soc. Int’l Conf.*, IEEE Computer Soc. Press, Los Alamitos, Calif., Feb. 1996, pp. 85–92.
7. A. Singhai, S.-B. Lim, and S.R. Radia, “The SunSCALR Framework for Internet Servers,” *Proc. IEEE Fault-Tolerant Computing Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., June 1998.
8. R.J. Schemers, “lbmnamed: A Load Balancing Name Server in Perl,” *Proc. 9th Systems Administration Conf.*, Usenix Assoc., Berkeley, Calif., Sept. 1995.
9. M. Beck and T. Moore, “The Internet2 Distributed Storage Infrastructure Project: An Architecture for Internet Content Channels,” *Proc. 3rd Workshop WWW Caching*, Manchester, England, 1998. Available online at http://www.cache.ja.net/events/workshop/18/mbeck2.html.
10. V. Cardellini, M. Colajanni, and P.S. Yu, “DNS Dispatching Algorithms with State Estimators for Scalable Web-Server Clusters,” *World Wide Web J.*, Baltzer Science, Bussum, Netherlands, Vol. 2, No. 2, July 1999.
11. E. Anderson, D. Patterson, and E. Brewer, “The Magicrouter, An Application of Fast Packet Interposing,” Univ. of California, Berkeley, May 1996. Available online at http://www.cs.berkeley.edu/~eanders/projects/magicrouter/osdi96-mr-submission.ps.
12. G.D.H. Hunt et al., “Network Dispatcher: A Connection Router for Scalable Internet Services,” *J. Computer Networks and ISDN Systems*, Vol. 30, Elsevier Science, Amsterdam, Netherlands, 1998.
13. O.P. Damani et al., “ONE-IP: Techniques for Hosting a Service on a Cluster of Machines,” *J. Computer Networks and ISDN Systems*, Vol. 29, Elsevier Science, Amsterdam, Netherlands, Sept. 1997, pp. 1,019–1,027.
14. M. Garland et al., “Implementing Distributed Server Groups for the World Wide Web,” Tech. Report CMU-CS-95-114, School of Computer Science, Carnegie Mellon Univ., Pittsburgh, Pa., Jan. 1995.
15. D. Andresen et al., “SWEB: Toward a Scalable World Wide Web-Server on Multicomputers,” *Proc. 10th IEEE Int’l Symp. Parallel Processing*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 850–856.
16. V. Cardellini, M. Colajanni, and P.S. Yu, “Redirection Algorithms for Load Sharing in Distributed Web-Server Systems,” *Proc. 19th IEEE Int’l Conf. Distributed Computing Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., May 1999.
17. A. Bestavros et al., “Distributed Packet Rewriting and its Application to Scalable Web Server Architectures,” *Proc. 6th IEEE Int’l Conf. Network Protocols*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1998.
18. P. Barford and M. Crovella, “Generating Representative Web Workloads for Network and Server Performance Evaluation,” *Proc. ACM Sigmetrics 98*, ACM Press, New York, 1998, pp. 151–160.

Valeria Cardellini is a doctoral candidate in computer engineering at the University of Rome Tor Vergata. Her research interest is in distributed computer systems, with an emphasis on Internet and Web-based applications. Cardellini received the Laurea degree in computer engineering from the University of Rome Tor Vergata. She is a student member of the IEEE Computer Society.

Michele Colajanni is an associate professor in the Department of Information Engineering at the University of Modena, Italy. His research interests include parallel and distributed systems, Web infrastructure, parallel computing, load balancing, and performance analysis. Colajanni received a Laurea degree in computer science from the University of Pisa and a PhD in computer engineering from the University of Roma Tor Vergata. He is a member of the IEEE Computer Society and the ACM.

Philip S. Yu is manager of the Software Tools and Techniques group at the IBM T.J. Watson Research Center. He holds or has applied for 74 US patents, and is recognized as a Master Inventor in the IBM Research Division. Yu received a BS from National Taiwan University, Taipei, an MS and a PhD from Stanford University—all in electrical engineering—and an MBA from New York University. He is a Fellow of the IEEE and a Fellow of the ACM.

Contact the authors concerning this article at cardellini@uniroma2.it; colajanni@unimo.it; and psyu@us.ibm.com.