WebSeAl: Web Server Allocation[*]

Mehmet Karaul
Dept. of Computer Science
New York University
karaul@cs.nyu.edu

Yannis A. Korilis
Bell Labs
Lucent Technologies
yannisk@bell-labs.com

Ariel Orda
Dept. of Electrical Engineering
Technion
ariel@ee.technion.ac.il

Abstract

With the rapid growth of the World Wide Web, clients attempting to access some popular web sites are experiencing slow response times due to server load and network congestion. Replacing the single server machine with a set of replicated servers is a cost-effective solution to partition server load which also allows incremental scalability and fault transparency. Distributing these replicated servers geographically can reduce network congestion and increase availability. However, distributed web sites are faced with the issue of allocating servers: how do clients find out about the replicas and how do they decide which one to contact? Popular web sites have well publicized server names and require a transparent mapping of the public server name to replicated servers.

Unlike most traditional approaches, we propose a technique which pushes the server allocation functionality onto the client. We argue that this approach scales well and results in increased performance in many cases. Building on theoretical work based on game theory, we show that the usage of individual replicas can be effectively controlled with cost functions even when the clients are noncooperative. We present the design and implementation of *WebSeAl*, our prototype system realizing these techniques. WebSeAl does not require any changes to existing client and server code, conforms to all standards, and does not generate any control messages. Preliminary experiments utilizing servers on six continents and in controlled settings indicate that WebSeAl improves performance significantly while imposing little overhead.

[*]This research was sponsored by the Defense Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0320; by the National Science Foundation under grant number CCR-94-11590; and by the Intel Corporation. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

1 Introduction

The rapid growth of the World Wide Web has led to a steady increase of client requests to many popular web sites. Both overloaded servers and network congestion contribute to slow response times of such sites. It may not be cost-effective to upgrade the server machine with a more powerful one, especially when incremental scalability is desired. Instead, most sites opt to replace the single server with a cluster of replicated servers [15, 16]. Although this may solve the problem of overloaded servers, it does not address network congestion. In addition, increasing the network capacity may not be cost-effective when incremental scalability is desired. Instead, some sites choose to geographically distribute the replicated servers—this approach has become popular with software archives (e.g. [20]) which have *mirror* sites, typically on several continents. Such a distributed architecture may result in increased availability of the service in times of network congestion and partial unavailability, and it may increase performance by taking advantage of "proximity" between clients and servers.

Currently, distributed web sites require the user to manually select a server out of a list of replicas. For example, there exist over 70 mirror sites distributed all over the world from which users can download Netscape browsers [17]; the decision as to which one to use is left to the user however. Designing a transparent allocation strategy for a distributed web site which does not sacrifice any of its benefits is a challenging task. A successful solution must meet several requirements:

- **Transparent Name Resolving:** Popular web sites have well publicized server names and require a transparent mapping to replicated servers.

- **Scalability:** Server allocation should gracefully scale with the increasing number of clients.
- **Flexibility:** Different users may have different objectives when accessing web sites, requiring support for customized strategies.
- **Load Balancing:** Service providers should be able to effectively control the utilization of individual servers.
- **Dynamic Changes in Server Pool:** Addition, removal, and migration of servers should be supported, and changes should be reflected as quickly as possible.
- **Fault Transparency:** Unresponsive machines should be detected and requests transparently redirected to other replicas. Also, previously unresponsible machines which become available again should be incorporated quickly.
- **Geographic Distribution:** Network delays between a client and individual servers of a distributed service might differ significantly. Server allocation should take advantage of this while still accommodating dynamic changes in network performance and server load.
- **Legacy Code and Standards:** It should not require any changes to existing client or server code and should conform to existing standards.

A comprehensive solution for allocation of distributed web servers must address all these factors. We are not aware of any system which achieves this. In this paper, we present a system called *WebSeAl* which addresses these issues.

The research leading to our system is based on theoretical work where provable methods for controlling network load using pricing mechanisms were developed. It was shown that even with noncooperative clients (in a fully distributed, and therefore scalable fashion), the network load can be controlled effectively. The work presented in this paper applies these techniques to provide scalable and controllable load balancing for distributed web servers.

The remainder of this paper is structured as follows. Section 2 gives an overview of related work. Section 3 discusses WebSeAl's architecture and describes how clients strive to minimize delays. Section 4 shows how load balancing can be achieved by introducing cost functions. Experimental results showing WebSeAl's performance are presented in Section 5, and Section 6 provides concluding remarks.

2 Related Work

The HTTP redirect [1] approach uses the HTTP return code *URL Redirection* [2] to perform load balancing. A busy server returns the address of another server instead of the actual response, asking the client to resubmit its request to that server. This creates additional network traffic and increased latency. Every request is initially addressed to the publicly known server which creates a single point of failure and the potential for a bottleneck due to servicing redirects.

Domain Name Server (DNS) based approaches [3, 10, 5] perform load balancing at the name resolution level. The name server at the server side is modified to respond to translation requests with the IP numbers of different hosts in a Round-Robin fashion. This results in partitioning client requests among the replicated hosts. The main disadvantage of this approach is that intermediate name servers and clients cache name-to-IP mappings which can result in significant load imbalance.

Server side approaches [7, 5] use a server side routing module which redirects all incoming requests to a set of clustered hosts based on load characteristics. This is achieved at the IP layer—i.e., the routing module modifies all IP packets before forwarding them to individual hosts. An alternate server side solution which avoids modifying IP packets is presented in [6]. These approaches have the drawback that the routing module represents a single point of failure, and therefore can result in a bottleneck since all requests pass through it. In addition, server side approaches work well only for clustered servers.

Perhaps most closely related to WebSeAl is the work presented in [21]. It uses a modified web browser to perform routing decisions at the client side. The browser downloads an applet which the service provider needs to implement to realize service specific routing. This approach creates increased network traffic due to applet transmission and potential control messages between the applet and the servers.

3 WebSeAl Architecture

A distributed web site (fig. 1) consists of a set of servers $S_1 \ldots S_n$, each with its own IP number $IP_1 \ldots IP_n$. One of these servers is known to the standard DNS system by the logical address of the original single server. We assume that the service content is replicated and that each server knows about the IP numbers of all individual servers comprising the distributed service. This might be achieved through mirroring or with a distributed file system [8, 18]. Except for the bootstrapping phase, all replicas are treated equally, and as long as any one replica is responsive, clients will be able to access the service.

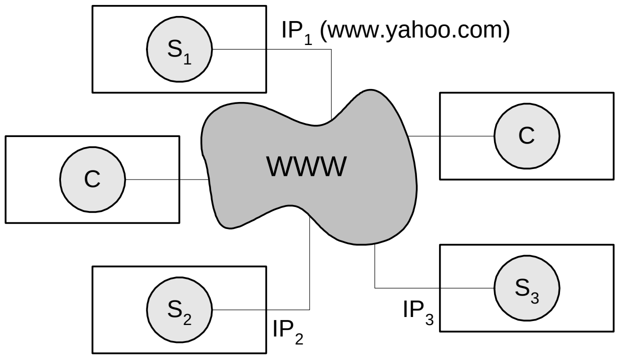

Figure 1: A distributed web site.

In WebSeAl, clients are responsible for routing individual requests to different servers comprising a distributed web site. This functionality is provided by a *client agent* module. In the basic architecture of the system, one agent is associated with each client. The client agent:

- intercepts the requests generated by the local client;
- has address information about the individual servers;
- collects dynamic performance data (e.g. network conditions, server load, and other site specific information);
- makes routing decisions based on this information;
- forwards the request to the selected server, receives the response, and delivers it to the client;
- transparently redirects the request to an alternate server if the selected server is not responsive.

A *server agent* module located on each server host provides address information to the client agent. It also communicates other site specific information which might be used to control access to the server pool, to support charging for services, and so forth, as will be discussed in Section 4.

In the remainder of this section, we will first discuss WebSeAl's routing strategies, then present how logical names of distributed web sites are resolved, and conclude with implementation specific issues.

3.1 Routing Strategies

The combination of the stateless nature of HTTP and the fact that many web pages contain several images and frames result in the generation of several requests to retrieve a single web page. WebSeAl client agents measure the total response time for each such request. The total response time measured is the complete end-to-end delay which includes connection establishment, network delay, and server time. WebSeAl client agents strive to minimize this total delay.

Each client agent makes routing decisions based on the average response time of each server. These averages are estimated using the measured response times for the N most recent requests. The updated routing strategy is used to direct the next N requests to the appropriate servers in the pool. Alternatively, the client agent could estimate the average response times by sending occasional *probes* at the cost of increased network traffic. One of the main design goals of WebSeAl is to avoid control traffic, so we decided against this approach.

One possible routing strategy client agents could employ is to always contact the most responsive server. Routing all requests to a single server, however, will fail to collect new performance data for the slower servers. Instead, we use probabilistic routing to ensure that client agents collect new performance data for all servers. More specifically, if T_{mi} denotes the average response time for requests routed from client m to server i, then routing of the next N requests is based on the probability distribution:

$$p_{mi} = \frac{1/T_{mi}^k}{\sum_j 1/T_{mj}^k}, \qquad (1)$$

where $k \geq 0$ is a constant. With $k = 0$, requests are routed to the servers randomly, without taking into account their performance. With $k = 1$, we can achieve linear distribution. This will favor fast machines while still using slower ones. However, the overall performance might suffer due to possibly long delays from slow servers. By raising k, more requests will be routed to the most responsive servers.[1] Very high routing probabilities for the fastest servers will cause very infrequent usage of slower ones, which in turn will decrease the potential to quickly detect improved servers. WebSeAl imposes a minimum threshold to circumvent this.

In our current implementation, we base routing decisions only on the most recent N measurements. We are considering several alternate strategies two of which are:

- **Weighted Average:** When calculating the performance estimate of a replica, more recent data should impact the overall performance more than older data, and the estimates should be updated more frequently.
- **Time-of-Day:** Network conditions and server usage vary with the time-of-day or the day of the week [4, 9], and this information should be considered in the routing strategy.

WebSeAl allows different clients to use different routing strategies. We plan to experiment with various strategies and to investigate how each one and various combinations perform in different settings. Our goal is to realize a set of routing strategies and to adapt dynamically to changing conditions.

3.2 Name Resolution

A server is identified by a logical address in the form of a hostname. When a client attempts to contact a server, the DNS system transparently resolves the hostname to an IP number, which is successively used to establish the connection. To contact a distributed server in a transparent fashion, a one-to-many mapping from the hostname to one of the IP numbers of the replicated machines is needed. WebSeAl pushes this name resolving functionality onto the client agents.

Client agents maintain a cache of logical hostnames and corresponding IP numbers to perform the mapping using address information provided by server agents. When a client agent attempts to access a distributed server for which it does not have a mapping cached, it uses standard DNS name resolving and contacts the server agent at that logical address. The server agent uses the local web server to generate the response and includes the addresses of the individual server agents in the response. The client agent extracts the addresses from the response and creates an entry in its cache. Future requests to this distributed server use this information to perform a one-to-many mapping from the logical address to the individual hosts. The standard DNS system is used only for bootstrapping—once a mapping for a logical address is cached, the DNS system is not needed to access any of the replicas.

Client agents need to retrieve the addresses of the server agents only to create an initial entry or to refresh their cache if the address list has changed in any way. To avoid unnecessary transmission of address information, client agents include a timestamp in their requests which indicates the state of the currently cached mapping for the given distributed server. Upon receipt of a request, each server agent inspects this timestamp and includes the addresses in the response only if more up-to-date address information is available. This is very similar in nature to the `If-modified-since` header [2], which is used to avoid retrieving cached files which have not been modified since a certain date.

HTTP allows application specific header fields and requires that all intermediaries such as proxies or gateways conforming to HTTP ignore these and forward them unchanged. We utilize this to "piggyback" timestamps and addresses in HTTP messages. WebSeAl introduces two new message headers: `Replica-Date` and `Replica-Addresses`. Client agents use the first header to tell server agents the status of their cached addresses for the distributed server at hand. Servers use both headers to return a list of addresses and the timestamp at which this information was updated.

Mapping a logical hostname to a set of IP numbers shares many similarities with DNS based and server side approaches described in the previous section. Notice that these approaches require that the servers on all replicated hosts accept connections at the same port. Also, the directory structure must be identical on each host. WebSeAl's architecture relaxes these restrictions. The mapping from hostname to IP numbers can be easily extended to a mapping from hostname and port to IP number and port to accommodate usage of different port numbers. This requires that the address information included in re-

[1] As k approaches infinity, all requests will be routed to the most responsive machines.

sponses be extended to contain port numbers as well as hostnames. Path offsets can be accommodated similarly. For example, `www.yahoo.com:80/` can be mapped to `www.cs.nyu.edu:8888/yahoo/`. On the first host, the server is accepting connections at port 80 and the directory structure is rooted at /. On the second host, the server accepts connections at port 8888 and the root directory is at `/yahoo/`. Many mirror sites use different root directories and require a relative path offset.

3.3 Implementation Issues

WebSeAl requires a server agent module at the server side (fig. 2). This functionality could be added to existing web servers quite easily and should impose only little computational overhead. However, to create a usable system without having to modify existing servers, WebSeAl provides a stand-alone Java application which implements the server agent functionality. It intercepts every incoming request, forwards it to the local web server, accepts the response, adds the address information to the response as needed, and forwards it to the client agent.

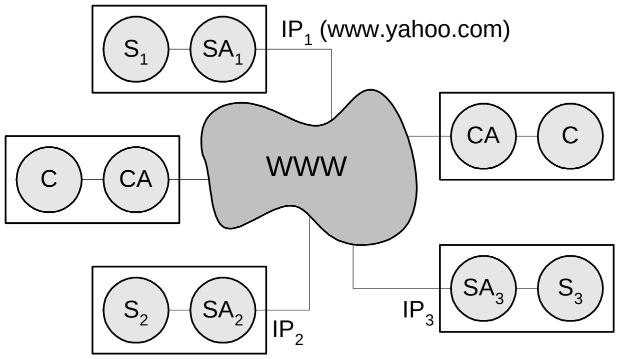

Figure 2: A distributed web site with WebSeAl client agents and server agents.

The client agent module is somewhat more complex, but it should be fairly straightforward to extend existing web browsers to support this functionality. Similar to the server agent, WebSeAl provides a stand-alone Java application which realizes the client side functionality in order to provide a usable system without having to modify existing clients. We utilize the fact that virtually all browsers support proxies to intercept requests. When the client agent is started up, it creates a server socket which accepts HTTP requests, very much like a proxy does. By configuring the browser to use the "proxy" (i.e., WebSeAl client agent), the client agent effectively intercepts each request.

Proxies are generally used to allow Internet access through firewalls and perform caching of web documents. WebSeAl's client agent can accommodate proxies in two ways. First, a client agent can be located between one or more clients and a proxy. Since name resolution is performed at the client agent, the proxy will treat identical documents from different replicas of the same distributed server as different documents and create redundant copies in its cache. Alternatively, the client agent can be located "behind" the proxy. This configuration avoids the problem of redundant copies in the proxy cache. Also, only one address cache and a single set of statistical data is maintained for a number of users, resulting in more up-to-date address caches and more accurate estimates.

Both WebSeAl's server and client agent functionality should ideally be included in web servers and browser or proxies. We provide client and server agents to enable service providers and users to take advantage of this technology without the need to modify existing systems. Independent of whether agents are used or existing systems are modified, for a system like WebSeAl to gain wide acceptance it needs to be backward compatible with regard to clients and servers lacking this functionality. WebSeAl is backward compatible and supports gradual infiltration:

- **WebSeAl Client and Standard Server:** A standard HTTP server is required to ignore the timestamp header in a request from a WebSeAl client agent and will service the request as usual. The lack of address information in the response indicates to the client agent that it is dealing with a standard server. It can react to this, for example, by infrequently including the timestamp in its future requests in order to update its cache in case this site is upgraded.

- **Standard Client and WebSeAl Server:** A request received by a server agent will not contain a timestamp header if the client lacks WebSeAl functionality. The server can react to this in several ways; two possibilities are: (1) it services the request in a standard manner without including any address information in its response; (2) it routes the request on behalf of the the client to individual servers.

4 Management of the Server Pool

WebSeAl client agents are *noncooperative* in the sense that they make their routing decisions independently from each other, striving to optimize their individual performance. While each client can implement a routing strategy of its choice, in the current design client agents route requests to servers with minimal average response time. The *operating point* of the system—i.e., the load distribution over the server pool—is therefore solely the result of the interaction among the various distributed client agents and cannot be controlled by the service provider. In this section, we will discuss strategies that can be used at the server side to control the operating point of the system while the client agents make their routing decisions in a noncooperative manner.

The service provider aims at distributing the load currently offered to the server pool in a way that is deemed efficient from the *system's* point of view. The provider, for instance, might desire an operating point that minimizes the *overall* average response time of the server pool. In other cases, the provider might want to discourage usage of certain machines—even if they are the most responsive ones—in order to perform other site specific tasks. Therefore, a mechanism is needed to make the distributed client agents implement routing strategies which lead to an operating point that coincides with the desired one.

The problem of managing the behavior of systems where control is distributed and noncooperative is a fundamental one. The interaction among the various distributed controllers (client agents in WebSeAl) can be modeled as a *game*, and Game Theory provides the systematic framework to study and analyze the behavior of such systems—for an overview of game theoretic aspects in computer networking see [11] and references therein. The operating points of the system are the *Nash equilibria* of the underlying control game. Noncooperative equilibria are inherently inefficient: while each controller strives to optimize its individual performance, the overall behavior of the system is, generically, suboptimal.

WebSeAl uses a *pricing mechanism* to provide incentives to the noncooperative client agents to implement routing strategies that lead to the desired load distribution over the server pool. The methodology is motivated by recent analytical studies in the area of networking which have shown that a network/service provider can enforce any desired operating point by means of appropriate pricing strategies [13, 14]. The key idea in WebSeAl's pricing mechanism is that there is a *service cost* associated with obtaining service from each server in the pool. Client agents are now making their routing decisions based not only on performance statistics, but also on service cost information for each server. The main assumption behind this mechanism is that the client agents are indeed "sensitive" to service costs. This behavior is expected in private Intranets where client agents and the pricing mechanism are part of the same management system. For external client agents accessing the web site, this behavior can be enforced by actual usage-based service charges (for commercial web sites), or by means of limited electronic budget allocated to each client—an architecture developed according to these ideas is proposed in [12]. When client agents are sensitive to service costs, the service provider can control not only the load distribution over the available servers, but also the total offered load itself.

To support pricing functionalities in WebSeAl, each distributed web site is equipped with a *pricing manager* module. Based on the targeted operating point, the price manager determines the service costs to access each server and communicates it to the corresponding agent. The server agent provides pricing information about the server to the client agents that receive service from it. In the remainder of this section, we will first discuss the pricing strategies in the current design of WebSeAl and then address some implementation issues.

4.1 Pricing Strategies

The goal of the pricing mechanism in WebSeAl is twofold:

- Avoidance of congestion (overload conditions) at various servers.
- Load balancing—that is, distribution of the total load offered to the web site among the available servers in a way that is deemed efficient by the provider.

The pricing strategies in the current version of WebSeAl are based on analytical results in [14]. That study considers a system of general network resources accessed by a number of noncooperative clients. Each resource is characterized by its "capacity," that is, the maximum load that can be accommodated by the resource. *Congestion pricing* is proposed as a means for avoiding overload conditions: the service cost per size unit (i.e., the price) of each resource is proportional to the congestion level at the resource that depends on the total load offered to it by the clients. More specifically, the price of each resource is given by the congestion function associated with the resource multiplied by a weight factor. These weights determine the relative sensitivity of the clients to the congestion level at the various resources and will be referred to as the *discount factors*. Load balancing can be achieved by appropriate choice of these discount factors. This pricing strategy is shown to allow the provider to enforce any desired operating point while the clients make their routing decisions noncooperatively.

Along the lines of these analytical results, the pricing strategy in the current design of WebSeAl is based on determining a discount factor for each server in the pool, which determines the relative sensitivity of the client agents to the responsiveness of the server.[2] In particular, the performance metric considered by each client agent in making its routing decisions is the average response time of each server multiplied by the corresponding discount factor. Therefore, if w_i is the discount factor of server i, and T_{mi} the average response time from the server to client agent m, then the routing strategy of the client agent described by eq. 1 becomes:

$$p_{mi} = \frac{1/(w_i T_{mi})^k}{\sum_j 1/(w_j T_{mj})^k}. \qquad (2)$$

4.2 Implementation Issues

The server discount factors are determined by the pricing manager based on the operating point that the provider wants to enforce. One way to determine these factors is to map the parameters of the model considered in [14] to the characteristics of WebSeAl and apply the corresponding analytical results, expecting to achieve a good approximation of the desired operating point. Instead, we chose to use an *adaptive algorithm*, also proposed in [14], which does not depend on the details of the underlying analytical model. The algorithm updates the discount factors iteratively, based on the “distance” of the current operating point from the desired one.

If f_i^* denotes the desired load at server i and $f_i(n)$ the actual load offered to the server during the n-th iteration, then its discount factor w_i is updated using the following:

$$w_i(n+1) = w_i(n) e^{\theta_i (f_i^* - f_i(n))}, \qquad (3)$$

where $\theta_i > 0$ is a constant that determines the rate of change in the discount factor of server i. The idea behind this iterative scheme is that, if the server is currently receiving less load than the desired one, its discount factor should be decreased. This decreases the clients’ sensitivity to the congestion level at the server, thus encouraging them to direct more of their requests to it. Similarly, if the server receives more load than the desired one, its discount factor is increased. Under a set of general assumptions guaranteeing that the client population as a total reacts “rationally” to price changes, this iterative scheme was shown in [14] to drive the system to the desired operating point.

In the current implementation of WebSeAl, server load is expressed in requests per unit of time. Considering HTTP requests, we expect that each client generates a large number of requests, each of small to moderate size. Therefore, this is a satisfactory approximation. A more precise load metric would consider the actual size of each request and will be incorporated in future implementations.

The pricing manager periodically collects information about the load offered to each server by contacting the corresponding server agent, updates the discount factors according to iteration 3 and communicates them to the server agents. Each agent receives only the update of its associated server and is responsible for advertising it to the client agents. This is achieved by piggybacking the discount factor of the server to HTTP messages that contain the responses to the clients’ requests.

Iteration 3 indicates that the discount factor of each server is determined using only *local* information, namely, the difference between the load currently offered to it and the targeted one. Therefore, the adaptive algorithm is well suited for distributed implementation: if the server agent is cognizant of the desired load at the server (f_i^* in eq. 3), then it can update the discount factor of the server without contacting the pricing manager. Note, however, that the target load (in requests per time unit) typically depends on the total load offered to the web site, information that is only available to the pricing manager. If the total offered load is not expected to change dramatically, the pricing manager can inform the server agents about their target load less frequently. Then, each server agent can use itera-

[2]Note that the discount factor of each server is the same for all clients.

tion 3 to update the server's discount factor on a faster time scale.

5 Experiments

In this section, we will present initial performance results. For the first three tests, we used ten mirror sites of a popular software archive which repeatedly appears in [19] as one of the most accessed web sites. These tests were conducted under real world conditions, using standard machines, networks, and software. The ten servers were located on six continents: two each in North America, South America, Europe, and Asia, and one each in Africa and Australia. The client was running at New York University. Geographically, the closest server to the client was located in Massachusetts, the second closest in California.

The client running five threads generated 1000 requests for a file of length 4253 bytes. All requests were addressed to a single logical address. A local client agent intercepted each request and provided transparent access to a distributed web site. Since we experimented with existing web sites not running WebSeAl's server agent, we added the server addresses manually into the cache of the client.

For the first experiment, we ran two tests: one using WebSeAl's client agent and one contacting the closest server directly. Using the client agent, the total response time for 1000 requests was 291.6 s. The response time we measured is the end-to-end delay which includes connection establishment, network delay, and server time. 95.4% of the requests were serviced by the closest server. The total response time for contacting the closest server directly was 266.9 s. This translates to an overhead of 9.2%. The fact that the WebSeAl client agent sent the vast majority of the requests to the closest server indicates that this server was delivering the best performance. Besides the computational and communication overhead of the client agent, an important factor contributing to this overhead is that 4.6% of the requests were routed to slower servers to update performance data for these machines. As mentioned before, this could be avoided by occasionally sending probes at the cost of generating additional traffic.

In our second experiment, we used the same setup as before, but ran the experiment at a different time of the day. This time, only 3.9% of the requests were serviced by the closest site. The total response time was 761.4 s as opposed to 1295.3 s when contacting the closest host directly—an improvement of 41.2%. These two experiments indicate that WebSeAl can deliver significant performance gains while imposing only little overhead, compared to the scenario when the user is able to always pick the fastest machine.

The third experiment investigates how WebSeAl client agents adapt to the dynamic performance changes of individual servers. As with the previous two experiments, the client, using a local client agent, generated 1000 requests to a logical address. After 300 requests, we started downloading several large files from the fastest site, which happened to be the geographically closest one, thus generating additional load at that server. This traffic was discontinued after another 300 requests. Of the first 300 requests, 93.3% were serviced by the closest server. This percentage sank to 11.6% for the next 300 requests, and went up again to 93.2% for the last 400 requests. The second closest server received initially 2.0% of the requests, which increased to 69.3% when the performance of the closest server started to degrade. This indicates that WebSeAl adapts well to performance changes in the server pool (fig. 3).

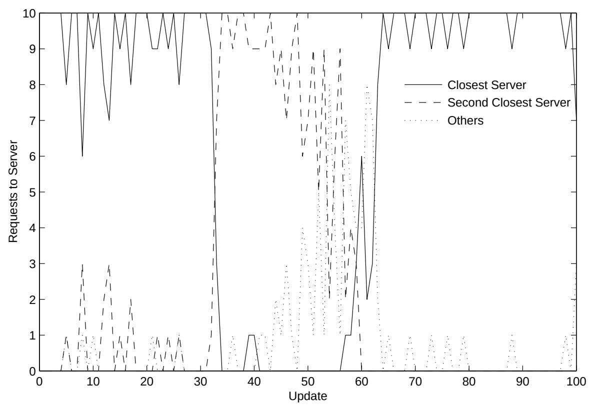

Figure 3: Request distribution in a dynamically changing environment.

For our last experiment, we used several identical machines in a controlled environment to show how WebSeAl reacts to changes in the server pool. On each of four machines, we started a WebSeAl server agent and a standard web server. We first used three servers, added another one after about 300 requests, and removed one of the original three servers after another 400 requests[3]. Since we used identical machines, it can be expected that the two fully available machines would each get 300 requests, the other two each 200 requests. The actual distribution was 295 and 286 requests for the first two machines, and 225 and 194 requests for the other two. This illustrates that WebSeAl quickly and effectively accommodates changes in the server pool.

6 Conclusions

WebSeAl is a novel architecture for managing resources of web sites consisting of a pool of replicated servers. Unlike most existing proposals, in WebSeAl it is the responsibility of the clients to route their requests to individual servers. This architecture scales well with the number of users, delivers flexible quality of service, and provides fault masking.

We proposed routing strategies for directing client requests to the most responsive servers. Unlike server side approaches, routing decisions are based not only on server load, but also on network traffic conditions. We also discussed strategies that can be used at the server side to induce efficient allocation of resources (load balancing) while clients make their routing decisions in a noncooperative manner. Motivated by recent studies on game-theoretic aspects of networking, we proposed a pricing mechanism that provides incentives to the clients to route their requests in a way that is deemed efficient by the service provider.

A prototype system based on this architecture has been implemented and its functionality has been validated through a series of experiments. These results indicate that WebSeAl can deliver significant performance gains while imposing minimal overhead.

References

[1] D. Andresen, T. Yang, V. Holmedahl, and O.H. Ibarra. Sweb: Towards a scalable world wide web server on multicomputers. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)*. IEEE Computer Society Press.

[2] T. Berners-Lee, R. Fielding, and H. Nielsen. RFC 1945: Hypertext transfer protocol — HTTP/1.0, May 1996.

[3] T. Brisco. RFC 1794: DNS support for load balancing, April 1995.

[4] K.C. Claffy, H.W. Braun, and G.C. Polyzos. Tracking long-term growth of the NSFNET. *Communications of the ACM*, 37(8):34–45, August 1994.

[5] IBM Corporation. *Interactive Network Dispatcher User's Guide*, 1997. Available at http://www.ics.raleigh.ibm.com/netdispatch/ND2MST.HTM.

[6] O.P. Damani, P.E. Chung, Y. Huang, C. Kintala, and Y.M. Wang. One-IP: Techniques for hosting a service on a cluster of machines. In *Proceedings of the Sixth International World Wide Web Conference*, Santa Clara, CA, April 1997.

[7] D. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available server. In *Digest of Papers. COMPCON '96. Technologies for the Information Superhighway*, pages 68–74, Santa Clara, CA, February 1996. IEEE Computer Society Press.

[8] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[9] Matrix Information and Directory Services Inc. *MIDS Internet Weather Report*. Available at http://www3.mids.org/weather/.

[10] E.D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN Systems*, 27(2):155–164, 1994.

[11] Y.A. Korilis, A.A. Lazar, and A.Orda. Architecting Noncooperative Networks. *IEEE Journal on Selected Areas in Communications*, 13(7):1241–1251, September 1995.

[12] Y.A. Korilis, T.A. Varvarigou, and S.R. Ahuja. Pricing Mechanisms for Distributed Resource Management. Technical Memorandum BL0112570-120396-TM3, Bell Laboratories, Lucent Technologies, December 1996.

[3]We removed the server by sending the server agent process a *kill*-signal.

[13] Y.A. Korilis, T.A. Varvarigou, and S.R. Ahuja. Optimal Pricing Strategies in Noncooperative Networks. In *Proceedings of the 5th International Conference on Telecommunication Systems: Modeling and Analysis*, pages 110–123, Nashville, TN, March 1997.

[14] Y.A. Korilis, T.A. Varvarigou, and S.R. Ahuja. Pricing Noncooperative Networks, June 1997. Submitted to the *IEEE/ACM Transactions on Networking*. Available at http://www.multimedia.bell-labs.com/people/yannisk/price.html.

[15] http://www.ncsa.uiuc.edu/.

[16] http://www.netscape.com/.

[17] http://www.netscape.com/download/client_download .html.

[18] A. Siegal, K. Birman, and K. Marzullo. Deceit: A flexible distributed file system. In *Proceedings of the 1990 Summer USENIX Conference*, Anaheim, CA, June 1990.

[19] http://www.top100.com/.

[20] http://www.tucows.com/.

[21] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using smart clients to build scalable services. In *Proceedings of the 1997 USENIX Annual Technical Conference*, pages 105–117, Anaheim, CA, January 1997. USENIX.