ONE-IP: Techniques for Hosting a Service on a Cluster of Machines

Om P. Damani*
Dept. of Computer Science
Univ. of Texas at Austin
damani@cs.utexas.edu

P. Emerald Chung
Bell Laboratories
Lucent Technologies
emerald@bell-labs.com

Yennun Huang
Bell Laboratories
Lucent Technologies
yen@bell-labs.com

Chandra Kintala
Bell Laboratories
Lucent Technologies
cmk@bell-labs.com

Yi-Min Wang
AT&T Labs, Research
ymwang@research.att.com

Abstract

With the explosive growth of the World Wide Web, some popular Web sites are getting thousands of hits per second. As a result, clients (browsers) experience slow response times and sometimes may not be able to access some Web sites at all. Upgrading the server nodes to more powerful machines may not always be cost-effective. A natural solution is to deploy a set of machines, or a cluster, and have them work together to host a single service. Such a server cluster should preferably publicize only one server name for the entire cluster so that any configuration change inside the cluster does not affect client applications.

In this paper, we first discuss existing approaches to distributing client's requests for a single service to different machines in a cluster. We then propose two new techniques, collectively called ***ONE-IP****, based on dispatching packets at the IP level. They have the advantages of fast dispatching and ease of implementation. Ideas presented here are generic and should be applicable to other services as well.*

*This work was done while the author was a summer intern in Bell Laboratories, Lucent Technologies.

Table of Contents

1. Introduction

With the explosive growth of the World Wide Web, many of the popular web sites are heavily loaded with client requests. For instance, the Netscape homepages are getting more than 80 million hits per day [8]. A single server hosting a service is usually not sufficient to handle this aggressive growth. We address this issue by designing a server cluster with the following requirements: only one *cluster address* should be publicized for the entire cluster. All client requests for URLs served by that address are sent to the cluster, and a dispatching mechanism ensures that each request is processed by exactly one server machine in the cluster. The number of connections for each client request should not be increased.

Additional desirable features for the cluster include: (1) server load should be evenly distributed among the machines; (2) if any server fails, the failure should be masked by distributing the requests to the remaining servers without bringing down the service; (3) the design should have the capability to add more machines to the cluster without bringing down the service; (4) it is preferable that the mechanism supports other server applications in addition to web services.

In this paper, we describe the design and implementation of two new techniques for providing single-name image for a server cluster, and compare them with other existing approaches. The first technique is based on a central dispatcher routing IP packets to different machines; the second technique is based on packet broadcasting and local filtering. Prototype implementations on a cluster of Sun SPARC workstations will be discussed.

2. Related Work

The HTTP protocol is an application-level protocol built for the World Wide Web. It is based on the client/server architecture. For each HTTP request, a TCP/IP connection between a client and a server is established. The client sends a request to a server and the server responds by supplying the requested information. The TCP/IP connection is then closed.

Typically, the server location for an URL is identified by the hostname contained in the URL [1]. To establish a TCP connection, the browser asks the DNS (Domain Name Service) to map the server hostname to an IP address. The IP address together with the port number (default is 80) form the transport address for the HTTP server process.

Based on this protocol, there are many ways to implement request distribution to a number of servers. In general, these servers provide the same set of contents. The contents are either replicated on each machine's local disk, shared on a network file system, or served by a distributed file system. Existing request distribution techniques can be classified into the following categories:

- In the *HTTP-redirection approach* [2], a busy server replies with an HTTP-redirection code and provides a new server address to which the client can resubmit its request. This approach is defined in the HTTP protocol [1]. Servers and browsers conforming to HTTP support this mechanism and the redirection is done in a user-transparent way. However, by redirection, a request may require two or more connections to get the service, thus this mechanism increases the response time and network traffic.

- The *Client-side approach* requires each client to have certain knowledge about the server cluster so that requests can be sent to different server addresses in a load-balanced manner. For example, when a Netscape Navigator is used to access Netscape's home page, it randomly picks a number, *N*, between 1 and 32, and goes to the server at www*N*.netscape.com [8]. However, few companies have the privilege of having the browsers to balance the load on their servers.

  Smart Client [9] is another client-side approach. It suggests that the service provide an applet running at the client side. The applet makes requests to several mirror sites to collect information about server loads and other characteristics, and forwards each client request to an appropriate server based on that information. Fault-tolerance and scalability are implemented at the client site. This approach is not client-transparent and it potentially increases the web traffic by sending extra status information from the servers to the client-side applets.

- The *Server-side DNS approach* implements load balancing in the *domain name to IP* translation process [5]. When a DNS server receives a translation request, it selects the IP address of one of the servers in the cluster in a round-robin fashion [7]. A drawback of this approach is that a translation request may go through a chain of DNS servers, and the name-to-IP mapping is cached by the intermediate DNS servers, which prevents future round-robin effect from reaching the clients. Also, in the event of a server failure, some clients may continue trying to access the failed server using the cached address.

- The *Server-side single-IP-image approach* implements a single IP image to the clients. The *TCP router* approach, proposed by IBM [4, 6] achieves the single-address image by publicizing the address of the server-side router. Every client request is sent to the router which then dispatches the request to an appropriate server based on load characteristics. The dispatching is performed by changing the destination IP address of each incoming IP packet to the address of a selected server. In order to create a seamless TCP connection, the selected server puts the router address instead of its own address as the source IP address in the replying packets, as shown in Figure 1. The advantage of this approach is that it does not increase the number of TCP connections, and is totally transparent to the clients. However, since the address change is done at the TCP/IP layer, the kernel code of every server in the cluster has to be modified to implement this mechanism. A hybrid of the DNS approach

and the TCP-routing approach has also been proposed, in which the DNS server selects one of several clusters in a round-robin fashion [6].

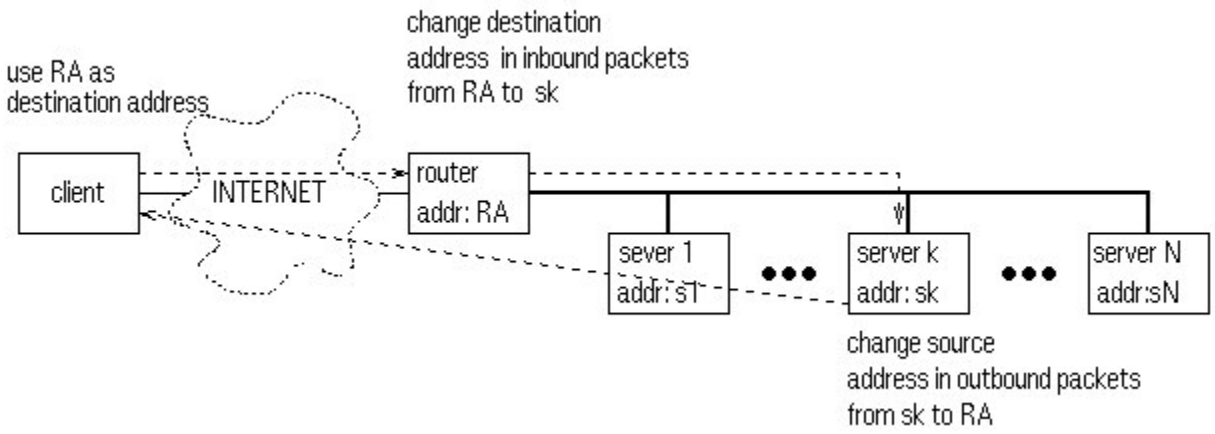

Figure 1. TCP router approach.

The *Network address translation approach*, is another server-side single-IP-image approach. Two examples are Cisco Local Director [11] and the Magic-router [3]. In this approach, all address changes are performed by the server-site router, including changing the source addresses of the responding packets, as shown in Figure 2.

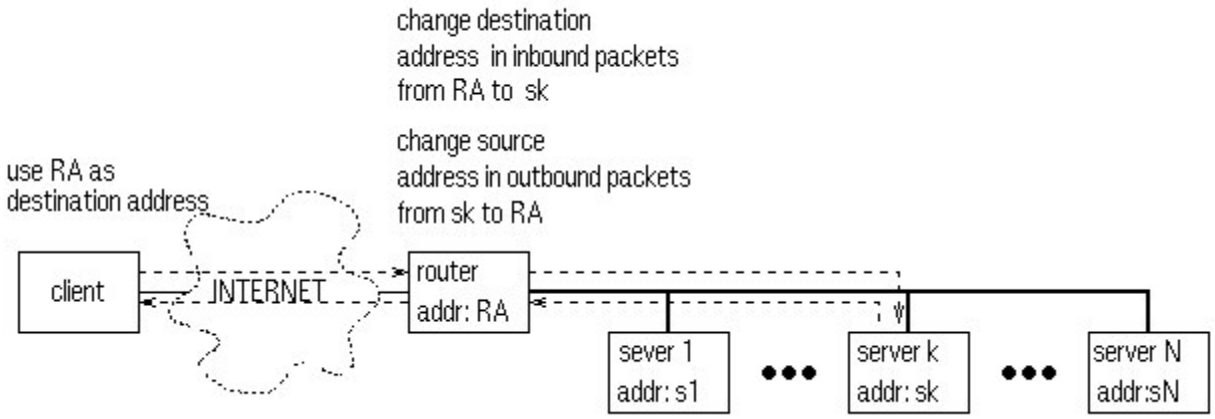

Figure 2. Network address translation approach Compared to the TCP router approach, network address translation approach has the advantage of server machine transparency, i.e., no specific changes from the server machine are required.

3. Design Strategies

3.1. Single Name Image By Single IP Address

Different from the above schemes, our approach lets all the machines in the cluster share one IP address as their secondary address by using the `ifconfig alias` option, which is available on most UNIX platforms. This shared address, called the *cluster address*, is publicized in the Domain Name Service. All client requests are sent to that address, and processed by one of the servers through its secondary address. All other communications inside the cluster are through the servers' primary IP addresses.

Note that our goal is to use multiple machines to serve one domain name, while the `ifconfig alias` option is typically used to solve a different problem: *how to use a single machine to serve more than one domain name?* In the latter case, `ifconfig alias` allows a machine to attach multiple IP addresses, and hence multiple domain names, to a single network interface [12]. Client requests destined for any of those domain names can then be picked up by the same machine. By examining the destination address in the packet, the server can distinguish which domain the request is for.

Our use of the `ifconfig alias` option has a different issue. Normally, two machines cannot share the same IP address because that would cause any packet destined for that address to be accepted and replied by both machines. That usually confuses the sender and leads to a connection reset. Therefore, before a machine attaches a new IP address to its network interface, it is checked that no other machine on the same LAN is using that address. If a duplicate is found, both machines are informed and warnings are issued. By carefully designing the dispatching and local filtering functions, our approach ensures that every packet will be processed by exactly one server, and so the above warnings do not cause a problem.

Using `ifconfig alias` to provide single IP image has two advantages. First, in both the TCP router and Network Address Translation approaches, it is necessary to change the destination address in the request packet header to a server address so that the server can accept the request; it is also necessary to change the source address in the response packet header to the cluster address so that the client can accept the reply. With `ifconfig alias` , all servers accept and respond packets through the cluster address, and so the packet headers need not be modified. Second, it has been noted that, because of the address change, the Network Address Translation approach may not work with those protocols that carry and use the IP addresses inside the application [15]. Since our approach does not modify the IP addresses in the packets, it can be used with those protocols as well.

3.2. Dispatching by Client IP address

When a client request reaches the cluster, some mechanism is required to select one machine to process that request. Once a server is selected, future incoming packets for the same request must be directed to the same server. In both TCP router and network address translation approaches, the router has to remember the IP mapping for every TCP connection. Upon receiving an incoming packet belonging to an existing TCP connection, the router has to search through all the mappings to decide which server it should forward this packet to. Under heavy load, the router itself may become a bottleneck unless special hardware design is used. * Since our goal is to allow low-cost implementations with fast dispatching, we propose the use of a static dispatching algorithm that selects servers solely based on a hash value of the client IP addresses. This will guarantee that all packets of the same connection always reach the same server. The hash function can be determined by analyzing the distribution of client IP addresses in the access log so that client requests are approximately evenly distributed to all servers. Although this type of static load balancing may not perform as well as the dynamic load balancing used in the other schemes, it should be sufficient for many applications.

When a server in the cluster fails, the subset of clients assigned to it will not be able to connect to the server. We handle this problem by dynamically modifying the dispatching function upon detecting a server failure. If the hash value of a client's IP address maps to the failed machine, the IP address will be rehashed to map to a running server. Note that the connections of the remaining clients are not affected.

Based on the above design strategies, we describe two architectures in the next two sections, which provide a trade-off between network bandwidth and CPU cycles.

4. Routing-based Dispatching

In this scheme, we publicize a *ghost* IP address as the cluster address. The name *ghost* is used to emphasize that none of the physical machines has that IP address as its primary address. Therefore, packets destined for the *ghost* address must belong to client requests that demand the service of the single-IP-image cluster. This clear distinction separates the cluster service from other normal activities so that they do not interfere with each other. Figure 3 shows the architecture of our routing-based dispatching scheme, where solid lines show LAN connections and dashed arrows show the path taken by a request.

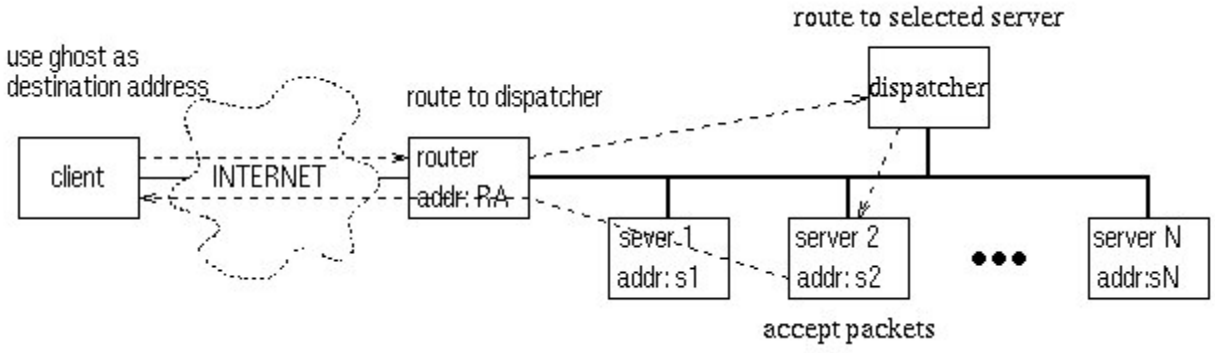

Figure 3. Architecture of routing-based dispatching.

To set up the configuration, we first insert a record in the routing table in the router so that it always routes those packets destined for the *ghost* IP address to the dispatcher. The operating system of the dispatcher is configured to run in the router mode, and the routing algorithm for *ghost* IP is modified to perform dispatching. Note that if we have access to the router's source code, this modification can be implemented at the router itself for better efficiency. Finally, all servers use `ifconfig alias` to set the *ghost* IP address as their secondary address. The kernel code running on the servers need not be modified.

When a client issues a request to the *ghost* IP address, the IP packets of this request arrive at the router. The router then routes the packets to the dispatcher according to the routing table entry. By applying a hash function to the client's IP address, the dispatcher selects one of the servers, *S2* for example, and routes the packets to *S2*. This routing should be based on *the primary address of S2* , instead of using the shared secondary address which would have caused confusion. After the network interface of *S2* accepts the packets, all higher level processing is based on the aliased *ghost* IP address because that is the destination IP address in the IP headers and possibly in the application packet contents. After processing the request, server *S2* replies directly to the client, again using the *ghost* IP address, without passing through the dispatcher.

4.1. Implementation Issues

When a packet destined for the *ghost* IP is received and put back on the same network interface by the dispatcher, it causes an ICMP host redirect message to be sent to the router. The purpose of this message is to ask the router to update its routing cache, so that any future packets for the *ghost* IP can bypass the dispatcher and go directly to their final destinations [13]. However, this effect is undesirable in our scheme

because we need the packets to be always dispatched by the dispatcher. Therefore, our implementation needs to suppress the ICMP redirect message from the dispatcher for the *ghost* IP. Note that if we implement the dispatching function at the router, the ICMP redirect message will not be generated.

Another potential problem is the ARP cache. When a reply packet is sent back to the client from the selected server with the *ghost* IP address, it causes the router to associate the *ghost* IP with the server's Ethernet address in its ARP cache [13]. However, since our scheme always first routes the packet to the dispatcher and the dispatching is based on servers' primary IP addresses, this ARP cache is never used and so the update does not interfere with the dispatching.

5. Broadcast-based Dispatching

As in the previous scheme, we publicize a *ghost* IP address and all the servers are aliased to this address. A broadcast mechanism is used to send client packets to every server. Each machine implements a local filter so that every packet is processed by exactly one server. The architecture for this scheme is shown in Figure 4. When a request packet from a client arrives at the router, the router puts the packet on the server network as an Ethernet broadcast packet, and so the packet is picked up by all the server machines. A small filtering routine is added to each server's device driver to ensure that only one machine accepts the packet. Each machine is assigned a unique ID number, and the filtering routine computes a hash value of the client IP address and compares it with the ID. If they do not match, the packet is discarded; otherwise, the server accepts the packet, and processes it as if it has received the packet through a normal IP routing mechanism. Since all the processing is based on the *ghost* IP, the reply packets are sent directly back to the client.

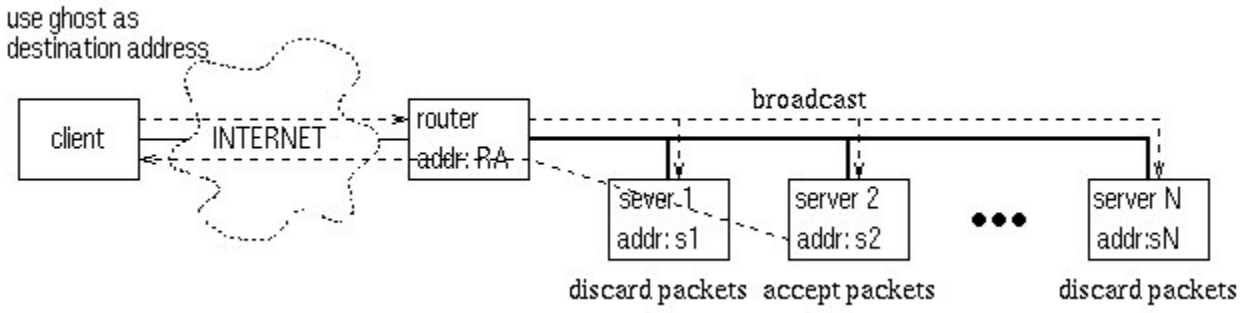

Figure 4. Architecture of broadcast-based dispatching.

5.1. Implementation Issues

The above scheme requires a permanent ARP entry at the router, which associates the *ghost* IP address to the Ethernet broadcast address. One problem is that any reply packet from a selected server appears to be coming from the *ghost* IP address, and so it would cause the router to overwrite the entry in its ARP cache to associate the *ghost* IP with that server's Ethernet address. To overcome this problem, we set up a routing entry that asks all packets destined for the *ghost* IP to be routed to another *ghost1* IP which is a legal subnet address in the cluster LAN and is not used by any machine. In addition, we insert an ARP entry at the router to associate the *ghost1* IP with the Ethernet broadcast address. When the router routes a packet to *ghost1*, it actually broadcasts the packet and causes every machine to accept that packet. Since no packet ever comes from the *ghost1* IP, its ARP entry is never changed and so the broadcast-based dispatching works correctly.

Another potential problem is that some operating systems, such as NetBSD, do not allow a TCP packet to be processed if it is received from an Ethernet broadcast address. We solved this problem by modifying the Ethernet address in the Ethernet packet header attached to the packet.

6. Prototype Implementations

We have implemented both schemes on a cluster of SUN SPARC workstations. NetBSD is used whenever kernel modification is necessary.

6.1. Overhead

The overhead due to dispatching in both schemes is minimal because packet dispatching is based on simple IP address hashing without storing or querying any mapping information. In the routing-based dispatching scheme, we add one hop on the server side for each incoming request packet, which typically adds a delay of 1 to 2 msecs to the TCP round-trip time. According to a study in [14], the median for TCP round-trip time is 187 msecs, and so the effect of this small delay is negligible. One disadvantage is that every packet sent to the *ghost* IP is routed one more time, which increases the traffic in the cluster LAN. However, in the Web service application, the size of a request is usually much smaller than the size of the response, which goes directly to the client without the additional routing. In the broadcast-based dispatching scheme, broadcasting each incoming packet in the server LAN does not increase network traffic. However, a hash value needs to be computed for every *ghost* IP packet, which increases the CPU load of each server. Compared to the communication delay, this computation overhead is negligible.

6.2. Scalability

In the routing-based dispatching scheme, the dispatcher is a potential bottleneck. The encouraging results from a study in the TCP router paper [6] have shown that a single dispatcher can support up to 75 nodes, which is sufficient for many practical systems. Since our dispatching function is simpler, the maximum number of supported nodes should be even higher. Our approach can also be combined with the DNS Round-Robin technique to further scale up. This can be achieved by asking the DNS to map the domain name to a number of different *ghost* IP addresses belonging to different server clusters in a round-robin fashion. The broadcast-based dispatching scheme has the advantage that there is no dispatching bottleneck. But we do need to modify the device driver of each server machine in order to filter the packets properly.

6.3. Load Balancing and Failure Handling

Given n server machines, $S0, S1, ..., S(n-1)$, and a packet from client IP address CA, a simple dispatching function is to compute $k = CA \bmod n$ and select server Sk to process the packet. More sophisticated dispatching functions can be designed by analyzing the actual service access log to provide more effective load balancing.

In order to detect failures, each server is monitored by a watchdog daemon *watchd* [16]. When a server fails, the detecting *watchd* initiates a change of the dispatching function to mask the failure and rebalance the load. A system call interface has been implemented to allow changing the dispatching function on-line. In the routing-based dispatching architecture, watchd notifies the dispatcher to change the dispatching function; while in the broadcast-based dispatching architecture, all the servers are notified to modify their filtering functions. If server Sk fails, for example, the new dispatching function checks to see if $(CA \bmod n)$

equals *k*. If that is true, a new hash value *j = (CA mod (n-1))* is computed. If *j* is less than *k*, the packet goes to server *Sj*; otherwise, it goes to server *S(j+1)*. Note that this simple scheme leaves all clients of non-failed servers unaffected, and reassigns the clients of the failed server evenly to the remaining servers. It can also be extended to handle more than one server failure.

Conversely, when a new machine is added to an *n*-server cluster, the dispatching function should be changed from *(CA mod n)* to *(CA mod (n+1))*. It is more difficult in this case not to affect existing connections while changing the dispatching function on-line. Since adding machines is usually not as emergent as handling failures, doing it during off-peak hours can minimize service disruption. Our current research focus is on developing algorithms to change the dispatching function on-line in a graceful way.

In the routing-based dispatching scheme, the dispatcher can become a single point of failure. Therefore, it also needs to be monitored by a *watchd*. Upon detecting a failure, the *watchd* triggers a fail-over of the dispatching function to a backup dispatcher, and asks the router to change the routing entry so that future *ghost* IP packets are routed to the backup. Since our scheme does not maintain any mapping table, the dispatcher is basically stateless and so the fail-over is straightforward.

7. Summary

We evaluated existing technologies and proposed two new techniques for providing single-name image for server clusters. A prototype on NetBSD kernel has been implemented and shows the advantages of fast dispatching and ease of implementation. Issues on overhead, scalability, and failure handling are discussed. Ideas presented here are generic and should be applicable to other servers as well.

Acknowledgment

The authors would like to express thanks to Gaurav Banga and Navjot Singh for their valuable discussions and their assistance in prototyping, and to Sampath Rangarajan for his helpful comments.

Footnotes

* The Cisco Local Director is a special-design hardware. It runs secure real time kernel and it claims that it is able to handle up to 1,000,000 TCP connections at the same time.

References

1.

T. Berners-Lee, R. Fielding and H. Frystyk, ``Hypertext Transfer Protocol - HTTP/1.0'', http://www.ics.uci.edu/pub/ietf/http/rfc1945.

2.

D. Anderson, T. Yang, V. Holmedahl and O. H. Ibarra, ``SWEB: Towards a Scalable World Wide Web Server on Multicomputers'', http://www.cs.ucsb.edu/Research/rapid_sweb/SWEB.html, *IPPS'96*, April, 1996.

3.

E. Anderson, D. Patterson, and E. Brewer, ``The Magicrouter, an Application of Fast Packet Interposing", http://www.cs.berkeley.edu/~eanders/magicrouter/osdi96-mr-submission.ps, OSDI, 1996.

4.

C. R. Attanasio, and S. E. Smith, ``A Virtual Multiprocessor Implemented by an Encapsulated Cluster of Loosely Coupled Computers", *IBM Research Report RC18442*, 1992

5.

T. Brisco, ``DNS Support for Load Balancing", *Network Working Group, RFC 1794*, http://andrew2.andrew.cmu.edu/rfc/rfc1794.html.

6.

D. Dias, W. Kish, R. Mukherjee, and R. Tewari, ``A Scalable and Highly Available Server", *COMPCON 1996*, pp. 85-92, 1996.

7.

T. T. Kwan, R. E. McGrath, and D. A. Reed, ``NCSA's World Wide Web Server: Design and Performance", *IEEE Computer*, pp. 68-74, Nov. 1995.

8.

S. L. Garfinkel, ``The Wizard of Netscape", *WebServer Magazine, July/August 1996*, pp. 58-64.

9.

C. Yoshikawam, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler, ``Using Smart Clients to Build Scalable Services", USENIX'97, Jan. 1997.

10.

``NetBSD Project", http://www.NetBSD.org/.

11.

``Cisco Local Director", http://www.cisco.com/warp/public/751/lodir/index.html.

12.

``Two Servers", http://www.thesphere.com/~dlp/TwoServers/.

13.

W. R. Stevens, ``TCP/IP Illustrated", Volume 1, pp. 69-85.

14.

W. R. Stevens, ``TCP/IP Illustrated", Volume 3, pp. 185-p.186.

15.

K. Egevang and P. Francis, ``The IP Network Address Translator (NAT)", http://www.safety.net/rfc1631.txt

16.

Y. Huang and C. Kintala, ``Software implemented fault tolerance: technologies and experience'', IEEE Fault-Tolerant Computing Symposium, pp. 2-9, June 1993.

Return to Top of Page
Return to Technical Papers Index