LSMAC and LSNAT: Two Approaches for Cluster-based Scalable Web Servers

Xuehong Gan[a,1], Trevor Schroeder[b], Steve Goddard[b], and Byrav Ramamurthy[b]
[a]Microsoft Corporation
One Microsoft Way
Redmond, WA 98052, U.S.A.
xuehongg@microsoft.com
[b]Department of Computer Science and Engineering
University of Nebraska–Lincoln
Lincoln, NE 68588, U.S.A.
{tschroed,goddard,byrav}@cse.unl.edu

Abstract— **Server responsiveness and scalability are more important than ever in today's client/server dominated network environments. Recently, researchers have begun to consider cluster-based computers using commodity hardware as an alternative to expensive specialized hardware for building scalable Web servers. In this paper, we present performance results comparing two cluster-based Web servers based on different server infrastructures: MAC-based dispatching (LSMAC) and IP-based dispatching (LSNAT). Both cluster-based server systems were implemented as application-space programs running on commodity hardware. We point out the advantages and disadvantages of both systems. We also identify when servers should be clustered and when clustering will not improve performance.**

I. Introduction

More and more companies have turned to the World Wide Web as an alternative way to provide channels for software distribution, online customer service, and business transactions. The function performed by the Web server is critical to a company's business. Successful companies will need to handle millions of "hits" on their server as well as handle millions of dollars in transactions per day. Server overload is frustrating to the customers, and harmful to the companies.

For many companies, the first choice to improve Web service is simply to upgrade the server to a larger, faster machine. While this strategy relieves short-term pressures, many companies find that they are repeatedly increasing the size and power of the server to cope with the demand for their services. What those companies need for their Web sites is incremental growth and massive scalability–the flexibility to grow with the demands of the business without incurring a large expense. One such solution is using a cluster-based server. Clustering low-cost computer systems is a cheap alternative to upgrading a single high-end Web server with faster hardware.

In the usual case (i.e., a non-clustered server), there is only one Web server serving the requests addressed to one hostname or Internet Protocol (IP) address. With a cluster-based server, several back-end Web servers cooperatively serve the requests addressed to the hostname or IP address corresponding to the company's Web site. All of these servers provide the same content. The content is either replicated on each machine's local disk or shared on a network file system. Each request destined for that hostname or IP address will be distributed, based on load-sharing algorithms, to one back-end server within the cluster and served by that server. The distribution is realized by either a software module running on a common operating system or by a special-purpose hardware device plugged into the network. In either case, we refer to this entity as the 'dispatcher'. Busy sites such as Excite, Inc. depend heavily on clustering technologies to handle a large number of requests [1].

We implemented and compared two different cluster-based Web servers using two different clustering technologies. The first is LSMAC , in which the dispatcher forwards packets by controlling Medium Access Control (MAC) addresses. The second is LSNAT , in which the dispatcher distributes packets by modifying IP addresses. We have implemented, for the first time, both methods in application space and they achieve comparable performance at a fraction of the cost of existing products.

The rest of this paper is organized as follows. We first discuss related work in Section 2, and then describe our implementations in Section 3. Section 4 describes how we evaluated our systems and presents the results. We present our conclusions and describe future work in Section 5.

II. Previous Work

Thanks to the widespread use of the World Wide Web, improving Web performance has been an important issue among researchers, Web server vendors, Web site administrators, and Web-related software developers. Web server clustering has proved to be effective in improving performance. One particular reason for this is its scalability. The administrators can easily add or remove servers according to business demands. Web server clustering technologies, such as Round Robin Domain Name Service (RR-DNS) and Single-IP-Image, require no changes on the client side. We discuss each of these techniques below.

A. Round Robin DNS

Early implementations of the cluster-based server concept used the Round Robin Domain Name Service. In RR-DNS, one of a set of server IP addresses will be returned with each DNS request. The return record sequence is circular-shifted by one for each response in a round robin fashion. RR-DNS is the most commonly used method mainly due to its simplicity and low cost. No additional software or hardware is needed. However, there are many drawbacks in using the RR-DNS technique for clustering servers. If a back-end server is taken off-line and the DNS record modified to reflect this, clients may still make requests for the old back-end server's IP address for several minutes because that name to IP mapping is cached by a local DNS

[1]The work was done when the author was at the University of Nebraska.

server or the client itself.

B. Single-IP-Image

In contrast to the multiple IP addresses in RR-DNS, methods for presenting a single IP image to clients have been sought and developed over the years. These methods work by publishing one IP address (cluster address) in DNS for clients to use to access the cluster. Each request reaching the cluster using the cluster address is distributed by the dispatcher to one of back-end servers. The methods differ in the way they forward packets to a back-end server. Currently there are two major schemes: layer two dispatching and layer three dispatching.

In the layer two approach, the dispatcher directly controls the MAC addresses of the frames carrying the request packets and then forwards the frames over a local area network (LAN). All servers in the cluster share the cluster address as a secondary IP address. The TCP/IP stack of the back-end server, which receives the forwarded packets, will handle the packets just as a normal network operation since its secondary IP address is the same as the destination IP address in the packets. No IP addresses in either inbound or outbound packets are modified, and the inbound packets and the outbound packets may go by different routes. The fact that outbound packets need not pass through the dispatcher reduces the amount of processing the dispatcher must do and speeds up the entire operation. This feature is especially important considering the extreme downstream bias on the World Wide Web, i.e., requests are small while the server responses are much larger. The mechanism for controlling the MAC addresses varies in different implementations [2].

In the layer three approach, each server in the cluster has its own unique IP address. The dispatcher is assigned the cluster address so that all client requests will first arrive at the dispatcher. After receiving a packet, the dispatcher rewrites the IP header to enable delivery to the selected back-end server, based on the load-sharing algorithm. This involves changing the destination IP address and recalculating the header checksums. The rewritten packet is then sent to the appropriate back-end server. Packets flowing from a server to a client go through a very similar process. All of the back-end server responses flow through the dispatcher on their way back to the client. The dispatcher changes the source IP address in the response packet to the cluster address, recalculates the checksums, and sends it to the clients. This method is detailed in RFC2391, Load Sharing Using Network Address Translation (LSNAT) [3]. A commercial example of the LSNAT approach is Cisco's Local Director [4]. A slight variation of this approach was proposed for IBM's TCP Router [5], in which the selected back-end server puts the cluster address instead of its own address as the source IP address in the reply packets. Even though the TCP Router mechanism has the advantage of not requiring the reply packets go through the TCP Router (dispatcher), the TCP/IP stack of every server in the cluster has to be modified.

III. Implementation

We are most interested in the Single-IP-Image approach, which is at the core of most commercial products. We implemented both layer two and layer three approaches as application-space programs. We call our implementations LSMAC and LSNAT, respectively. LSMAC dispatches each incoming packet by directly modifying its MAC addresses (Fig. 1). LSNAT follows RFC2391 (Fig. 2). Our solutions are much simpler and more portable than existing products, which involve modifying the TCP/IP stacks of the dispatcher and/or server machines.

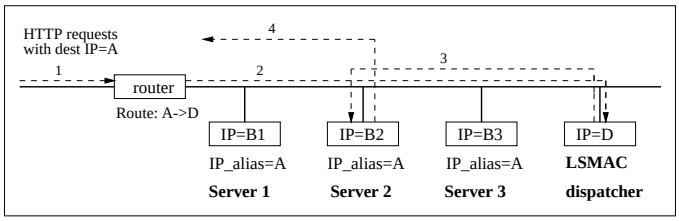

Fig. 1. LSMAC implementation in a LAN environment.

A. LSMAC

In LSMAC, the back-end servers are aliased to the cluster address and the dispatcher is assigned a different IP address. In order to make the dispatcher the only entry point for each packet addressed to the cluster-based server, we add one route in the immediate router to route every incoming packet to the LSMAC dispatcher. The LSMAC dispatcher uses the `libpcap` [6] packet capture library to capture each packet. The dispatcher maintains a table containing information about all existing sessions. Upon receipt of the packet, the dispatcher will determine whether it belongs to an existing session or is a new request. The IP addresses and port numbers of the two endpoints uniquely define every TCP connection (session) on the Internet. We use these to map incoming packets to corresponding connections already established with the back-end servers. If the session does not already exist, it is simply a matter of creating a new entry in our table. TCP flags on the incoming packets are used to identify the establishment and termination of each connection. The first packet of a TCP session is recognized by the presence of SYN bit and absence of ACK bit in the TCP flags. The end of a TCP session is detected when a packet with both FIN and ACK bits set is received or when a packet with RST bit set is received. Upon the termination of a TCP session, the corresponding mapping in the table is removed.

Once a mapping has been established, the LSMAC dispatcher rewrites the source and destination MAC addresses of each frame and sends them to a chosen back-end server. Since the MAC addresses have significance only in a LAN environment, LSMAC requires that the dispatcher and back-end servers be connected in a LAN.

Fig. 1 illustrates the packet flow in a LSMAC cluster.

1. A client sends a packet with a destination IP address *A*.
2. The immediate router sends the packet to LSMAC on *D*, due to the added route: *A*→*D*.
3. Based on the load sharing algorithm and the session table, LSMAC decides that this packet should be handled by the back-end server *B2*, and sends it to *B2* by changing the MAC addresses of the packet to *B2*'s MAC address.
4. The back-end server *B2* accepts the packet and replies directly to the client.

The operation of LSMAC offers two distinct advantages over

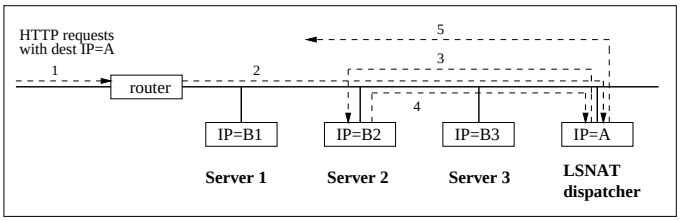

Fig. 2. LSNAT implementation in a LAN environment.

LSNAT, discussed below. As all operations are performed at OSI layer two, it is unnecessary to modify layer three data. This allows us to avoid recalculating TCP/IP checksums, an expensive operation. Secondly, LSMAC only processes half of the TCP stream: the portion flowing from client to server. This is only a small fraction of the total traffic flowing between the client and server as most of the data is contained in the server's response. This allows LSMAC to scale quite easily as the introduction of additional clients has relatively little impact in terms of the amount of data processed.

B. LSNAT

In our LSNAT implementation, only the dispatcher is configured to the cluster address. Normal routing rules ensure that it receives in-bound requests. We then use IP filters to keep the host operating system from responding to the requests itself, allowing the LSNAT application to process them manually using the `libpcap` [6] packet capture library. Conceptually, LSNAT appears as a single host to clients, but–as we will see–as a gateway to the back-end servers.

After receiving a client request, the LSNAT dispatcher sets up the connection mapping just as the LSMAC dispatcher does. Once a mapping has been established, it is necessary to rewrite the packet headers since it is addressed to the cluster address and not to an individual back-end server. The LSNAT dispatcher changes the destination IP address of each in-bound packet to the IP address of a selected server. For each out-bound packet, the LSNAT dispatcher changes source IP address to the cluster address, which is expected by the client. LSNAT allows the dispatcher and back-end servers to be in different LANs provided that traffic from the back-end servers to the clients is always routed through the LSNAT.

Fig. 2 illustrates the packet flow in a LSNAT cluster.

1. A client sends a packet with a destination IP address *A*.
2. The immediate router sends the packet to LSNAT on *A*, since the LSNAT machine is assigned the IP address *A*.
3. Based on the load sharing algorithm and the session table, LSNAT decides that this packet should be handled by the back-end server *B2*. Then it rewrites the destination IP address as *B2*, recalculates the IP and TCP checksums, and send the packet to *B2*.
4. The back-end server *B2* accepts the packet and replies to the client via the LSNAT dispatcher, which the back-end server sees as a gateway.
5. LSNAT rewrites the source IP address of the replying packet as *A*, recalculates the IP and TCP checksums, and send the packet to the client.

LSNAT suffers owing to its position in the connection between client and server. Unlike LSMAC, LSNAT changes the layer three payload so that data destined for the cluster address appears to, to a back-end server, to be bound for that back-end server. The reverse operation is applied to packets originating from the back-end server so that they appear to be from the cluster address. This requires the recalculation of packet checksums. Additionally, we must process both sides of the connection, not just the relatively small amount of data traveling upstream from the client. These two factors combine to make LSNAT extremely CPU intensive.

C. Discussion

To ensure that each back-end server contains the same set of files, some sort of file replication must be done or a common network file system must be used. The back-end servers behave as if they were communicating directly with the clients and do not need to know anything about the clustered nature of the system. This means that no special software needs to be installed on the back-end servers. Both the LSMAC and LSNAT approaches are transparent to the clients and servers. We use the round robin algorithm to distribute the load amongst the entire set of back-end servers for load sharing. This works well since all our servers are configured in a similar fashion and the requests from clients are comparable in size and duration. However, because our solution does not restrict the user to a certain server configuration, load-sharing algorithms based on individual server usage could yield better results in a heterogeneous environment.

Additionally, while different approaches were taken with regards to delivering data to the dispatcher (special routing rules in the case of LSMAC versus normal delivery and IP filtering in the case of LSNAT), neither approach must necessarily use the delivery mechanism we chose for it. Table 1 provides a comparison of the LSMAC and LSNAT approaches.

IV. Evaluation

WebStone [7] was used to benchmark the performance of our cluster-based server systems. WebStone is a configurable load generator for Web servers, which launches a number of Web clients to generate GET requests to the server, and measures the replies from the server.

A. Experimental Design

In our experiments, the dispatcher (LSMAC/LSNAT) and the back-end servers were executing on 266 MHz Pentium II machines with 64 MB memory. These machines were connected in a shared 100 Mbps Ethernet environment. Red Hat Linux 5.2 (kernel 2.2.6) and Apache Web Server 1.3 were installed on every machine. WebStone 2.0 was run on two 266 MHz Pentium II machines with 128 MB memory each on the same network. For the scalability studies, we ran experiments on four configurations: single server (no cluster and hence no dispatcher), one-server cluster, two-server cluster, and three-server cluster. We used the results from the single server and one-server cluster tests to measure the overhead due to the dispatchers. The server performance usually depends on the type of files that are being served. For this reason, we chose four file types in measuring each configuration: 0 KB files that have no payload but still require HTTP headers, 2 KB files which are typical of the first

TABLE I
COMPARISON OF KEY FEATURES OF THE LSMAC AND LSNAT IMPLEMENTATIONS.

Feature	LSMAC	LSNAT
OSI Layer of operation	Layer 2 (Data-link)	Layer 3 (Network)
Traffic Flow through dispatcher	Unidirectional (Incoming only)	Bidirectional
Incoming Packet Modification	No	Dest. IP address and checksum
Outgoing Packet Modification	Not applicable	Source IP address and checksum
Routing table change in immediate router	Yes	No
Servers in different LANs	Requires interface on each LAN	Allowed

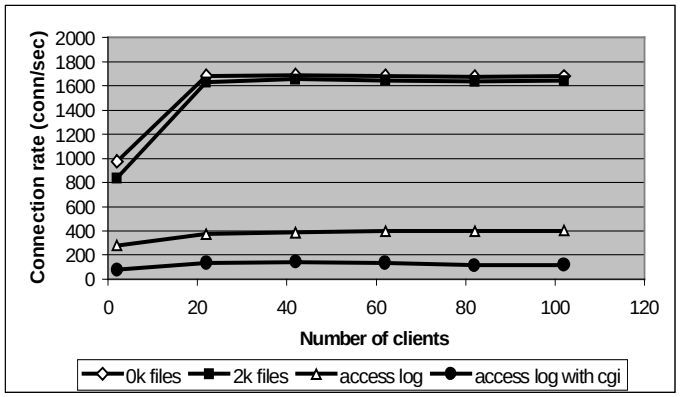

Fig. 3. LSMAC connection rates with 3 servers.

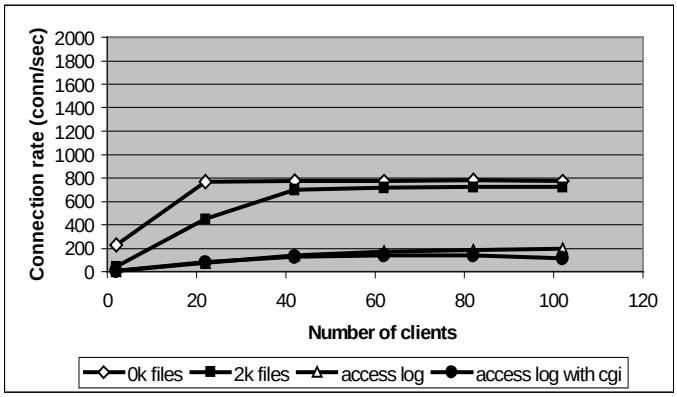

Fig. 4. LSNAT connection rates with 3 servers.

page of a Web server, a file mix with file sizes and access frequencies derived from a Web server access log (available from [8]), and fully dynamic files. The dynamic files were generated by a Common Gateway Interface (CGI) program based on file sizes and access frequencies derived from the same Web server access log. The testing with dynamic files is necessary since more and more dynamic content is appearing on the Web. Dynamic content plays an important role in nearly all high-volume Web sites.

B. Performance Measurement

Server connection rate and throughput are the two most important performance metrics for Web systems. The server connection rate is an indication of how fast the server can establish a connection and start communicating with the clients. The calculation of server throughput is simple: total bytes (body + header) transferred throughout the test divided by the total test duration. The server throughput depends on the transferred file size, server capability, and the network bandwidth.

B.1 Server Connection Rate

In general, a cluster-based server should have a higher connection rate than a single server, unless the network bandwidth or the clustering agent (dispatcher) becomes a bottleneck. Our tests with small files (0-2 KB) show that LSMAC with three servers can handle over 1600 connections per second (Fig. 3), and LSNAT can handle about 800 connections per second (Fig. 4). The connection rate in a single server configuration with the same file size is around 550 connections per second.

However, with the access log file mix, whose average file size is 108.5 KB, cluster-based servers do not improve the connection rate due to network congestion (Fig. 5). LSMAC with three servers maintains about 400 connections per second, which is very close to the connection rate of a single server (Fig. 5). LSNAT supports only 150 connections per second (Fig. 5). In LSNAT, the processing capacity of the dispatcher becomes the bottleneck before the network bandwidth. In practice, no actual connection would result in a zero-byte transaction. Nevertheless, the number of connections per second with a small file size is an important indicator of the dispatcher's capability. With a 2-byte page size, IBM Network Dispatcher can handle 850 connections per second when it runs in a Token Ring network [9].

It is interesting to note that LSMAC consistently shows more than twice the connection rate of LSNAT for all cases but the CGI case. This is because LSNAT spends more time in processing each packet than LSMAC–including the server to client flow which LSMAC does not process at all. We will discuss the CGI case in the next section.

B.2 Static vs. Dynamic Content

Fig. 5-8 show the relative performance of the LSMAC cluster and LSNAT cluster with respect to static and dynamic content. WebStone was used to generate requests for 42 Web clients. For easy comparison, the performance measurements of a single server (without a dispatcher) are also plotted in the figures. In the access log case, LSMAC significantly outperforms LSNAT. Both connection rate and server throughput of LSMAC are nearly triple those of LSNAT (Fig. 5 and 7). The LSNAT dispatcher is the obvious bottleneck in this case. However, in the CGI case they achieve similar connection rate and server

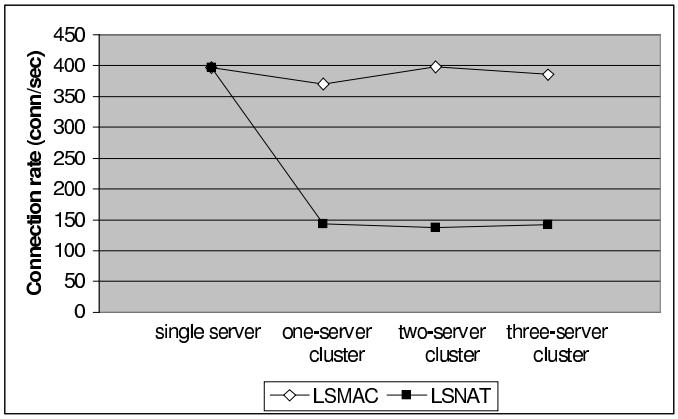

Fig. 5. Comparison of connection rates for access log file mix.

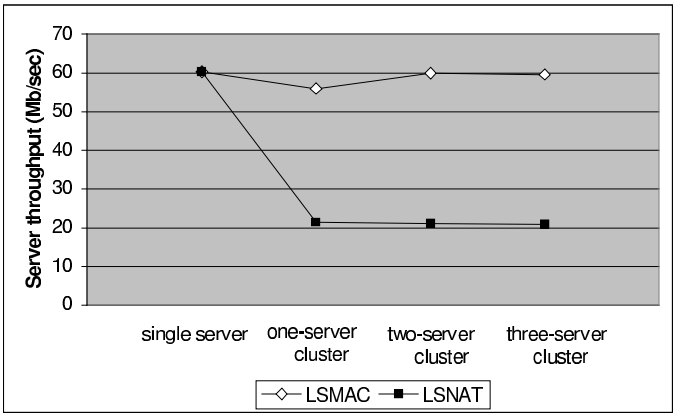

Fig. 7. Comparison of server throughput for access log file mix.

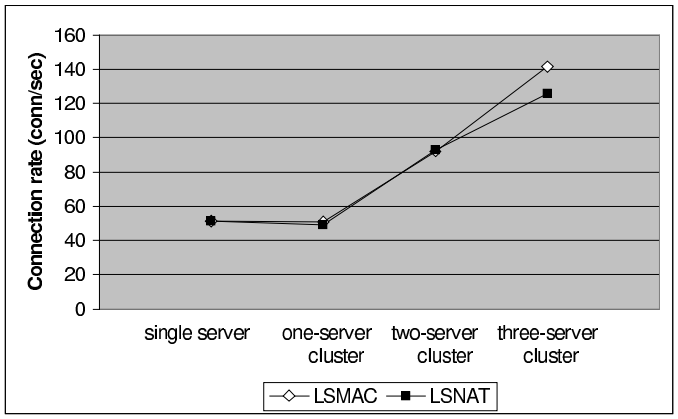

Fig. 6. Comparison of connection rates for CGI content.

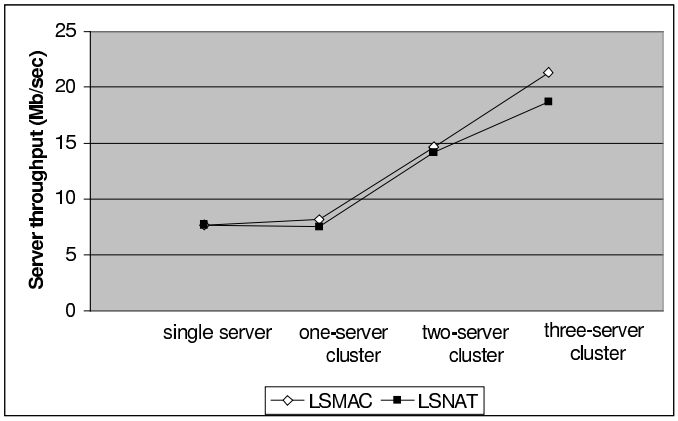

Fig. 8. Comparison of server throughput for CGI content.

throughput (Fig. 6 and 8). This is because in the CGI case the back-end servers are the bottlenecks. A CGI program runs as a separate process in the server machine every time a CGI document is requested and therefore is very costly. The connection rate is expected to increase if we add more back-end servers to the cluster in the CGI case (Fig. 6).

In summary, any one of the dispatcher, the back-end servers, or the network can "bottleneck" the operation of a cluster-based Web server system. Our tests show that LSMAC and LSNAT perform similarly with fully dynamic content, which is computationally intensive at the back-end servers. LSMAC outperforms LSNAT with a static access log mix, though it does not show any performance improvement over the single server due to our limited network bandwidth. Hence, for cluster planning, one needs to take into account the amount and types of information maintained on the Web site.

V. Conclusions

We implemented two cluster-based Web server systems in a simple and portable way: LSMAC and LSNAT. They represent the first application-space implementations of the two clustering technologies, and achieve performance comparable to existing non-application space products. Tests show that LSMAC significantly outperforms LSNAT for static files. But the two systems achieve similar performance for fully dynamic content. The choice of the LSMAC or LSNAT approach depends on the network environment, Web content, and service requirements. If the servers are connected in a LAN and there are a large number of requests, the LSMAC approach is ideal. If the servers are at different sites and there is a significant amount of dynamic content, you may want to choose the LSNAT approach. Our future work will focus on fault tolerance and developing adaptive optimized load-sharing algorithms.

References

[1] L. Bruno, "Balancing the load on Web servers," *Data Communications*, September 21, 1997, http://www.data.com.

[2] O. P. Damani, P. E. Chung, Y. Huang, C. Kitala, and Y. Wang, "ONE-IP: techniques for hosting a service on a cluster of machines," *Proc. 6th International WWW Conference*, Santa Clara, California, April 1997.

[3] P. Srisuresh and D. Gan, "Load Sharing Using Network Address Translation (LSNAT)", *RFC2391*, August 1998.

[4] Cisco Systems: Local Director, http://www.cisco.com/warp/public/751/lodir/.

[5] C. R. Attanasio and S. E. Smith, "A virtual Multiprocessor implemented by an encapsulated cluster of loosely coupled computers," *IBM Research Report RC18442*, 1992.

[6] University of California/LBL: Packet Capture Library, ftp://ftp.ee.lbl.gov/libcap.tar.Z.

[7] G. Trent and M. Sake, "WebStone: the first generation in HTTP benchmarking," *MTS Silicon Graphics*, February 1995.

[8] Mindcraft: WebStone, http://www.mindcraft.com/webstone/.

[9] G. Hunt, G. Goldszmidt, R. King, and R. Mukherjee, "Network Dispatcher: a connection router for scalable Internet service," *Computer Networks and ISDN Systems*, vol. 30, pp. 347-357, 1998.