LSMAC vs. LSNAT: Scalable Cluster-based Web Servers

Xuehong Gan, Trevor Schroeder, Steve Goddard, and Byrav Ramamurthy

Department of Computer Science and Engineering
University of Nebraska–Lincoln
Lincoln NE 68588-0115, U.S.A.

Server scalability is 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 architectures: OSI layer two dispatching (LSMAC) and OSI layer three dispatching (LSNAT). Both cluster-based server systems were implemented as application-space programs running on commodity hardware in contrast to other, similar, solutions which require specialized hardware/software. 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.

Keywords: Internet, HTTP, Web server, Web server clustering, LSNAT.

1. Introduction

More and more companies have turned to the World Wide Web (WWW) 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 customers, and harmful to companies.

The first option many companies use to scale their 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 up with the demand for their services. The increasing use of dynamic content such as CGI only serves to amplify this problem. What companies need for their Web sites is incremental growth and massive scalability–the flexibility to grow with the demands of the business without incurring large expenses. One such solution is a cluster-based Web server. Clustering a few low-cost commodity systems is usually a cheap alternative to upgrading a single high-end Web server with faster hardware.

In a non-clustered server, there is only one Web server serving the requests sent to one hostname or Internet Protocol (IP) address. In contrast, 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. In general, 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 Web server clustering technologies (Figure 1). The first is Load Sharing using Medium Access Control, or LSMAC (based on OSI layer two dispatching), in which the dispatcher forwards packets by controlling MAC addresses. The second is Load Sharing using Network Address Transla-

tion, or LSNAT (based on OSI layer three dispatching), in which the dispatcher distributes packets by modifying Internet Protocol (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. In addition, LSMAC provides auto configuration of the backend server pool. Our results show that under nearly all conditions, a layer two approach offers superior performance to an IP-level approach.

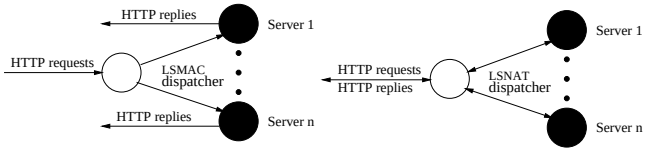

Figure 1. Logical representations of LSMAC and LSNAT cluster-based server.

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. In Section 4 we describe how we evaluated our systems and present the results. We present our conclusions and describe future work in Section 5.

2. Background and Previous Work

Clustering is any of a range of technologies that combines multiple inexpensive computers to work together as a single unit. Until recently, it was typically used in proprietary server environments to improve application uptime. In a cluster, two or more servers were tied into a configuration where one server takes on the workload of another in the event of a hardware or software failure. Availability measures how long a server can operate continuously without, planned or unplanned, outages. Scalability refers to how well a server performs when the number of client requests increase. In fact, high-end high-availability server clustering technologies, developed by such vendors as IBM, HP, and Sun Microsystems, have been introduced into the commercial mainstream. Most of these products only provide high availability, not scalability. Some products support rudimentary scalability. But they usually require developers to rewrite applications to be cluster-aware with vendor-specific APIs.

Historically, when clustering has been used to improve performance, it has required the use of special tools or interfaces such as PVM [2] or MPI [3]. This requirement has made developing and deploying clustered servers significantly more expensive than deploying an equal number of servers acting independently.

In contrast, Web server clustering can provide scalability without these limitations: owing to the simple nature of HTTP, the Web server software does not have to be cluster-aware; the servers can run any combination of operating systems on any mix of hardware with load distribution being done at the level of a single HTTP request. Web server clustering has proved to be effective in improving performance. One particular reason for this is its incremental scalability. Administrators can easily add or remove servers according to business demands without the high cost associated with purchasing and deploying a completely new system. For example, a sports magazine site may add more servers in the cluster during the Olympic games. Web server clustering technologies can be divided into two categories: Round Robin Domain Name Service (RR-DNS) and Single-IP-Image. Neither requires client modification. We discuss each of these techniques below.

2.1. Round Robin DNS

Early server clusters were implemented using the Round Robin Domain Name Service (RR-DNS) [4]. RR-DNS is a hostname-based approach. It works by mapping a single hostname of the server to several different IP addresses though the Domain Name Service (DNS). DNS is a giant hierarchical distributed database for mapping hostnames to their corresponding IP addresses. In RR-DNS, one of a set of server IP addresses will be returned with each request. The return record sequence is circular-shifted by one for each response in a round robin fashion. In this way, requests are, hopefully, distributed more or less evenly among a pool of servers rather than requiring a single server to service all requests.

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. RR-DNS does not automatically handle hosts that go down; so manual modification of DNS zone files and reloading DNS is required. For example,

when you take a misbehaving server off-line, the DNS servers will still pass out its address, being unaware of the server's status. This causes clients to receive error messages. Even if the DNS zone file is immediately modified after a server failure, problems still arise due to DNS caching.

If a name to IP address mapping has been cached by local DNS servers across the Internet; for the next few minutes, many users will get error messages instead of connecting to one of the available servers. Additionally, clients themselves may cache DNS replies and bypass DNS for subsequent requests, which similarly defeats the load sharing mechanism. Furthermore, with RR-DNS, each server must be configured identically because RR-DNS provides no control over the distribution of connections. In the same sense, low-end personal computers and powerful servers both have an equal chance of serving a HTTP request. This may result in the personal computers becoming overloaded with requests while the more powerful servers remain underutilized. It is worth noting that RR-DNS could be modified to weight each DNS entry differently according to the capacity of each server, although this is currently not the case.

2.2. Single-IP-Image

In contrast to the multiple IP addresses in RR-DNS, methods for presenting a single IP image to clients have been developed over the years. These methods publish one IP address (the 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 n back-end servers (Figure 1). Single-IP-Image approaches are advantageous because they require no modification to client software. While RR-DNS provides this same seamless transition from a single server to multiple servers, the undesired interaction between local name servers, local DNS resolvers, and the RR-DNS server make it less optimal than a Single-IP-Image.

The methods to achieve a Single-IP-Image differ in the way they forward packets to a back-end server. Currently there are two main schemes: OSI layer two dispatching and OSI layer three dispatching.

In the *layer two approach*, the dispatcher controls the MAC addresses of the frames carrying the request packets. All servers in the cluster share the cluster address as a secondary IP address while the dispatcher is assigned a different address. In our implementation, a routing rule is inserted into the routing table in the immediate router so that those packets destined for the cluster address are always routed to the dispatcher. Other options include a static ARP entry on the gateway to the LAN hosting the dispatcher so that all packets addressed to the cluster address are sent to the dispatcher, or disabling ARP responses on the back-end servers. The secondary IP address assignment can be accomplished using interface aliases on most Unix systems. 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 in-bound or out-bound packets are modified, and the in-bound packets and the out-bound packets may go by different routes. The fact that out-bound 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 WWW, i.e., requests are small while server responses are much larger.

The mechanism for controlling the MAC addresses varies in different implementations. ONE-IP [5] proposed a layer two method using broadcasting. Each packet is broadcast by the dispatcher to every back-end server. Each server implements a local filter so that every packet is processed by exactly one server. The disadvantage of this is that filtering reduces the capacity of each server to process client requests. Our implementation of the layer two approach, i.e., LSMAC, differs in that it directly rewrites the MAC addresses of each frame.

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 IP and TCP header checksums. The rewritten packet is then sent to the appropriate back-end server. Packets flowing from a back-end 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 clients. The dispatcher changes the source IP address in the response packet to the cluster address, recalculates the IP and TCP checksums, and sends it to the clients.

This method is detailed in RFC 2391, Load Sharing Using IP Network Address Translation (LSNAT) [6]. A commercial example of the LSNAT approach is Cisco's Local Director [7]. A slight variation of this approach was proposed for IBM's TCP Router [8], in which the selected back-end server places 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. Our implementation of the layer three approach, i.e., LSNAT, follows RFC 2391. Unlike TCP router, LSNAT is totally transparent to the clients and servers. We believe our LSNAT implementation is the first user-level implementation of RFC 2391.

2.3. Other Approaches

A hybrid of the RR-DNS approach and Single-IP-Image approach has also been studied, in which the DNS server selects one of several dispatchers in a round robin fashion [9]. Other innovative approaches, such as locality-aware distribution and dispatcher-side Web caching, are being investigated by the research community [10,11].

3. 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. Our layer two dispatcher, LSMAC (Load Sharing Using Medium Access Control), dispatches each incoming packet by directly modifying its MAC addresses (Figure 2). Our layer three dispatcher, LSNAT (Load Sharing Using Network Address Translation) follows RFC 2391 (Figure 3). 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. While LSNAT offers reasonable performance when serving dynamic content, LSMAC offers performance rivaling specialized solutions costing much more.

3.1. LSMAC

In this section, we look at the design and implementation of the LSMAC (OSI layer two) application. This includes the software tools used, the sequence of events in establishing connections using LSMAC, and a brief comparison of features with LSNAT (discussed in Section 3.2).

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` packet capture library [12] 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, using `libnet` [13], 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. The LSMAC dispatcher needs the MAC address of each back-end server for forwarding packets. The Address Resolution Protocol (ARP) is used to automatically discover back-end servers. This novel approach enables administrators to add (remove) servers to (from) the cluster dynamically. The user specifies the cluster address and load-sharing algorithm

when invoking the LSMAC dispatcher. The LSMAC dispatcher then broadcasts an ARP request for the cluster address and retrieves the back-end servers' MAC addresses from their ARP replies.

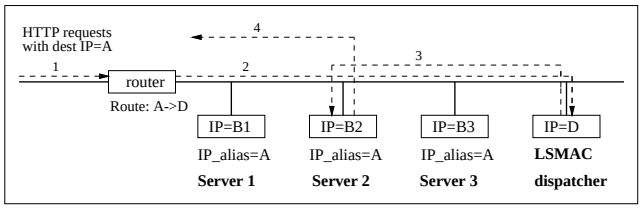

Figure 2. LSMAC implementation in a LAN environment.

Figure 2 illustrates the packet flow in a LSMAC cluster-based server.

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

The operation of LSMAC offers two distinct advantages over LSNAT, discussed below. As all operations are performed at OSI layer two, it is not necessary to modify layer three data. This allows us to avoid recalculating TCP/IP checksums, an otherwise 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 well since the introduction of additional clients has relatively little impact in terms of the amount of data processed.

3.2. LSNAT

In our LSNAT (OSI layer three) 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` 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.

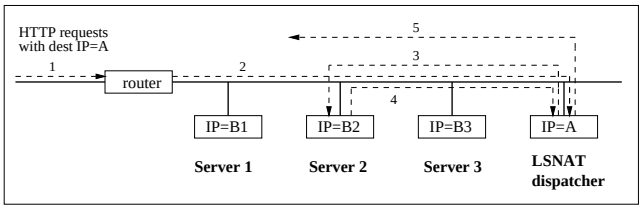

Figure 3. LSNAT implementation in a LAN environment.

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 the 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 between the back-end servers and the clients is always routed through the dispatcher.

Figure 3 illustrates the packet flow in a LSNAT cluster-based server.

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

IP and TCP checksums, and sends the packet to the client.

As we see here, the operation of LSNAT may be reduced to two simplex flows: one from clients to servers, the other the return path from servers to clients. In addition, the `libpcap` library encourages this approach. Therefore, LSNAT contains two threads of execution. The first processes all data traveling from the clients to the back-end servers; the other processes all data traveling in the opposite direction, from the back-end servers to the clients. Thread-safe execution is achieved by using mutex locks for all shared objects. In the case of the connection map, these locks are record-level.

LSNAT suffers owing to its position in the connection between client and server. Unlike LSMAC, LSNAT changes the layer three (network) payload so that data destined for the cluster address appears 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 both IP and TCP 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. In Figure 4 we see an illustration of the traffic flow between a client and a server. In the case of LSMAC, we see that the incoming packet requires no modification and that the response travels directly back to the client whereas with LSNAT, both the incoming packet and the outgoing packet must be modified by the dispatcher. LSNAT could realize a significant performance boost through the addition of hardware for checksum recalculation such as the use of gigabit Ethernet cards which do TCP/IP checksum calculation or through the use of a field programmable gate array (FPGA).

The one advantage offered by LSNAT is the ability to have servers in multiple networks without requiring multiple interfaces. As LSNAT operates at layer three, all that is required is that it be positioned on the network between clients and the back-end server. This would even allow for sparse geographic distribution for fault tolerance. LSMAC, on the other hand, must reside on the same physical network as the servers in its cluster. Therefore, if we wished to split the server pool into two or more networks to offer higher aggregate bandwidth, it would be necessary to have a physical interface on the LSMAC dispatcher to each of those networks.

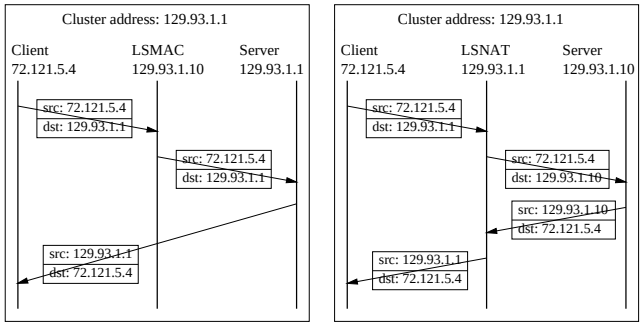

Figure 4. IP header modifications during dispatching operation in LSMAC and LSNAT approaches.

3.3. 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. Once this is done every server in the group must start its server software to handle the incoming requests. 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 dispatchers are transparent to the clients and the servers. We currently 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. Our modular design allows new load-sharing algorithms to be easily added to the systems.

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. Finally, even though we focus on Web server clustering in this paper, LSMAC and LSNAT can be used to distribute any simple TCP/IP-based network

Table 1
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)	Bidirectional
Incoming packet modification	No	Dest. IP address and checksums
Outgoing packet modification	Not applicable	Src. IP address and checksums
Servers in different LANs	Phys. Interface on Each LAN	Allowed

services with no or little modification. Table 1 provides a comparison of the LSMAC and LSNAT dispatchers.

4. Evaluation

WebStone [14] was used to benchmark the performance of our cluster-based server systems. WebStone is a popular configurable load generator for Web servers. As most of the published benchmarks were done using WebStone, we felt it was the best choice. WebStone consists of one "Webmaster" and a variable number of clients. The Webmaster is the controlling process, which spawns a variable number of client processes on each client workstation. These clients then communicate in a synchronous manner with the Web server, requesting documents for a given period of time and reporting their findings to the Webmaster. This freely available benchmark simulates varying client loads to measure a server's capability to accommodate client connections, to deliver data, and to answer requests.

4.1. Experiment Design

We measured the performance of the LSMAC and LSNAT dispatchers in both shared and switched 100 Mbps Ethernet environments. In our experiments, the dispatcher (LSMAC/LSNAT) and the back-end servers were executing on 266 MHz Pentium II machines with 64 MB memory. For tests in the shared Ethernet environment these machines were connected to a Linksys SPHUB08S hub. In the case of the switched Ethernet environment, every Webstone client workstation or back-end server was connected to one port of a Cisco Catalyst 2900 XL switch. Red Hat Linux 5.2 (kernel 2.2.6) and Apache Web Server 1.3 were installed on every machine. The Web server was configured to allow a maximum of 256 simultaneous server processes. 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 five configurations: single server (no cluster and hence no dispatcher), one-server cluster, two-server cluster, three-server cluster, as well as a special configuration with three dedicated servers in addition to the dispatcher itself acting as a back-end server. The results from the single server and one-server cluster tests measure the overhead due to the dispatcher. The server performance usually depends on the types 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 page of a Web server; a file mix with file sizes and access frequencies derived from a Web server access log (available from [15]); 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. 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.

It should be noted that the static file set configuration is designed to optimize the performance of the back-end web servers. The files being served are small enough that they are cached by the host operating system on the back-end servers. Thus the performance of the server does not rely on the disk subsystems, but upon other factors such as CPU speed, network bandwidth, etc. This allows us to benchmark our clustered Web servers versus the best performance that may be expected from a single server.

4.2. Performance Measurement

This section examines various metrics useful in evaluating the performance of a Web server, including a clustered server. We examine shared media networks versus switched environments, static versus dynamic content,

dispatcher CPU utilization for LSNAT and LSMAC, and dedicated dispatchers versus dispatchers coexisting with Web servers on the same machine.

Server connection rate, throughput, and request latency are the three most important performance metrics for Web systems. The server connection rate is expressed as connections per second. This 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. The request latency is the time elapsed between when the client sends its request to when it receives the entire response.

4.2.1. Shared vs. Switched Environment

In a cluster-based server there is one or more backend servers simultaneously handling requests. In general, it should have a higher connection rate than a single server, unless the network bandwidth or the dispatcher becomes a bottleneck. Our tests with small files (0-2 KB) in the shared Ethernet environment show that LSMAC with three servers can handle over 1600 connections per second (Figure 5), and LSNAT can handle about 800 connections per second (Figure 6). Tests in the switched Ethernet environment show that the LSMAC dispatcher can reach 1800 connections per second (Figure 7) while LSNAT achieves approximately 900 connections per second (Figure 8). The connection rate for a single server with the same file size is around 550 connections per second in shared environment and 700 connections per second in switched environment.

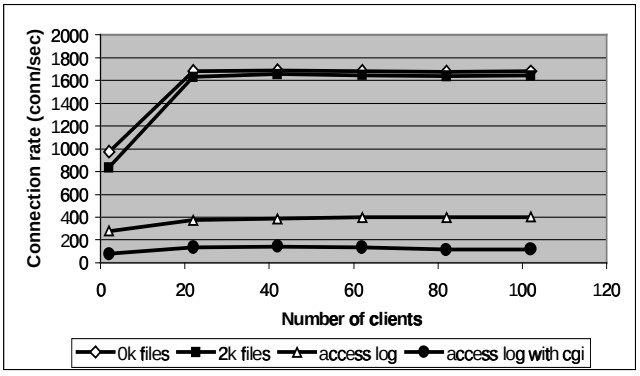

Figure 5. LSMAC connection rates in shared Ethernet environment with Linksys SPHUB08S hub.

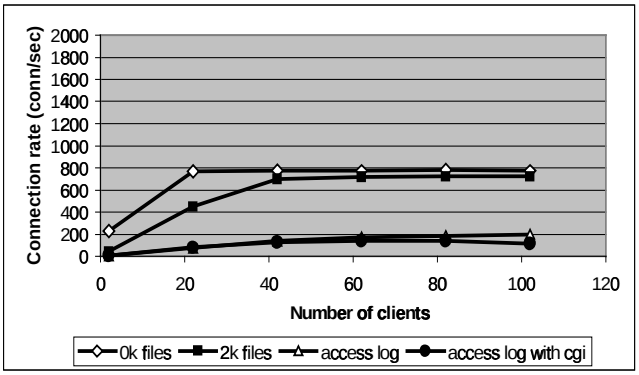

Figure 6. LSNAT connection rates in shared Ethernet environment with Linksys SPHUB08S hub.

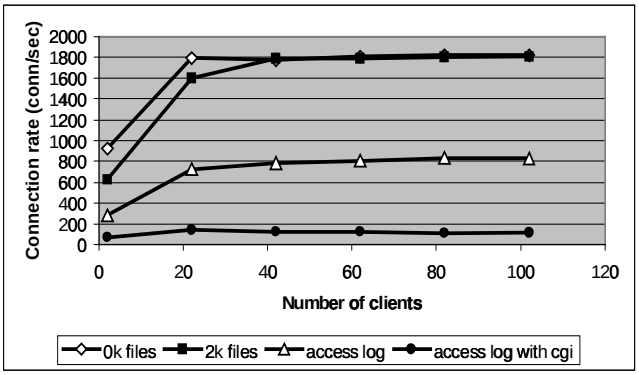

Figure 7. LSMAC connection rates in switched Ethernet environment with Cisco Catalyst 2900 XL switch.

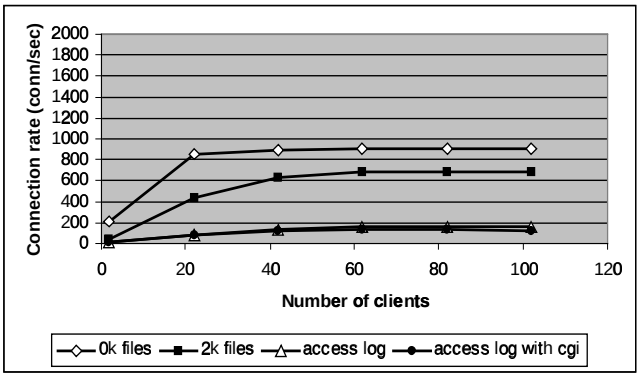

Figure 8. LSNAT connection rates in switched Ethernet environment with Cisco Catalyst 2900 XL switch.

With the access log file mix, whose average file size is 108.5 KB, cluster-based servers do not improve the connection rate in shared Ethernet environment due to network collision. LSMAC with three servers maintains about 400 connections per second, which is very close to the connection rate of a single server (Figure 9). LSNAT

supports only 200 connections per second (Figure 9). The processing capacity of the LSNAT dispatcher becomes the bottleneck before the network bandwidth. 400 connections per second are about the maximum rate for that file size in our 100 Mbps shared Ethernet environment, because at that rate the server throughput reaches 60 Mbps (Figure 11). However, with the same log file mix in a switched environment, the LSMAC dispatcher reaches near 800 connections per section, while the LSNAT dispatcher still maintains about 200 connections per second. This test points out that a higher capacity network is needed in order to fully utilize the functionality provided by LSMAC.

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 2200 connections per second when it runs on an SP-2 node with 6 back-end servers [16]. Each SP-2 node has a POWER2 67 MHz CPU and 256 MB memory. The maximum connection rate observed in the 1996 Summer Olympics was only about 600 connections per second [16]. IBM eNetwork Dispatcher and our dispatchers can easily handle that amount of traffic. While the IBM eNetwork Dispatcher requires expensive IBM hardware and is not generally portable to other environments, our dispatchers run on commodity hardware and commodity operating systems. Our solution is cost-effective as we can expect that the price/performance ratio of commodity hardware and commodity operating systems will remain superior to that of proprietary systems.

It is interesting to note that LSMAC consistently shows more than twice the connection rate of LSNAT in both shared and switched environment for all cases but the CGI case. The reason is that LSNAT spends more time in processing each packet than LSMAC and therefore slows down the connections. The LSNAT dispatcher must modify IP addresses in both in-bound and out-bound packets and recalculate their checksums, while LSMAC only changes the MAC addresses of inbound packets and no checksum recomputation is required. We will discuss the CGI case in the next section.

Some vendors also publish the number of simultaneous connections supported. However, different connections transfer different amounts of data. Some may cause a lot of traffic because of downloading multimedia data, while others may transfer only a few packets. Thus the number of simultaneous connections supported has more to do with memory in the dispatcher machine than its ability to quickly forward packets. A dispatcher usually uses less than 32 bytes of memory to keep track of each TCP session. Thus, a dispatcher running on a machine with 32 MB memory could theoretically support 1 million simultaneous connections. Yet in order to support a mere 1 Kbps sustained transfer rate (less than 2% of the capacity of a 56K modem) for each connection, 1 Gbps of network bandwidth would be required, not to mention the level of processing power required.

4.2.2. Static vs. Dynamic Content

Figures 9–14 show the relative performance of the three-server LSMAC cluster and the three-server LSNAT cluster with respect to static and dynamic content. WebStone was used to generate requests for 42 Web client processes distributed evenly among two client computers. 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 in a shared environment are nearly triple those of LSNAT (Figure 9 and 11). The difference in a switched environment is even larger (Figures 10 and 12). The LSNAT dispatcher is the obvious bottleneck in this case. However, in the CGI case they achieve a similar connection rate and server throughput (Figures 10 and 12). This is because, in the CGI case, the back-end servers are the bottleneck. 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 (i.e., ease the bottlenecks) in the CGI case (Figure 10).

In the access log case, LSMAC adds minimal overhead to the response time of a single server in shared environment and halves the response time of a single server in switched environment (Figure 13). LSNAT triples the response time of a single server in both shared and switched environments (Figure 13). However, both LSMAC and LSNAT show response time improvement in the CGI case (Figure 14). Even though

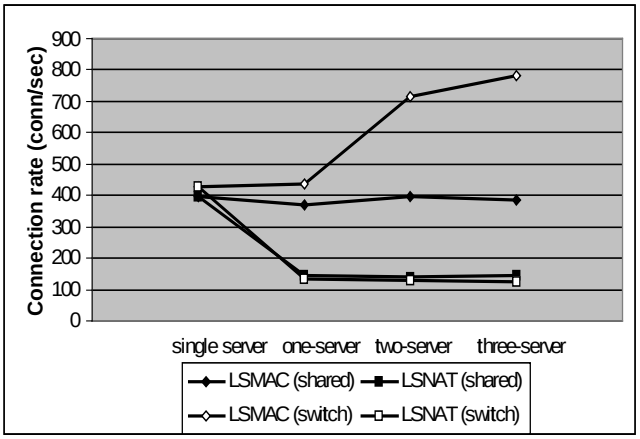

Figure 9. Connection rate for access log trace.

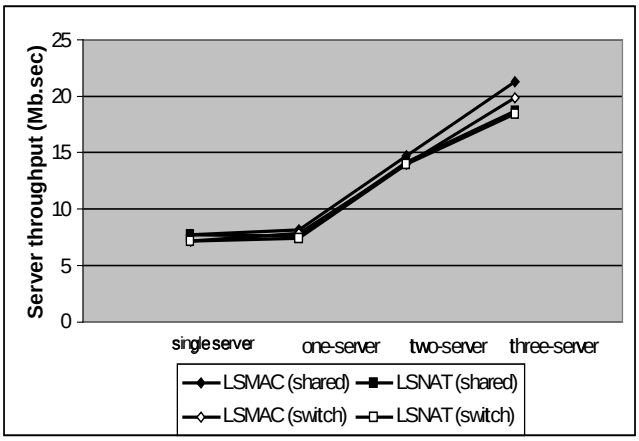

Figure 12. Server throughput for CGI trace.

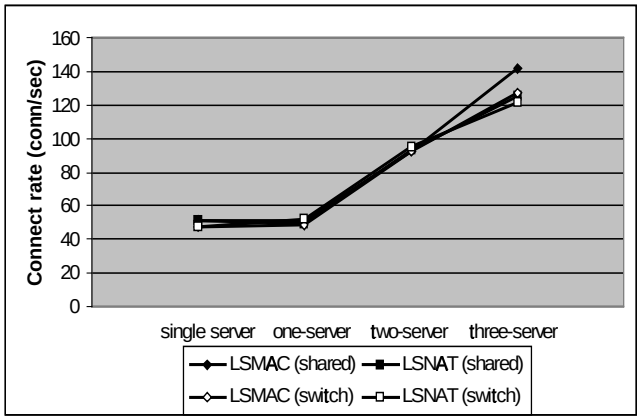

Figure 10. Connection rate for CGI trace.

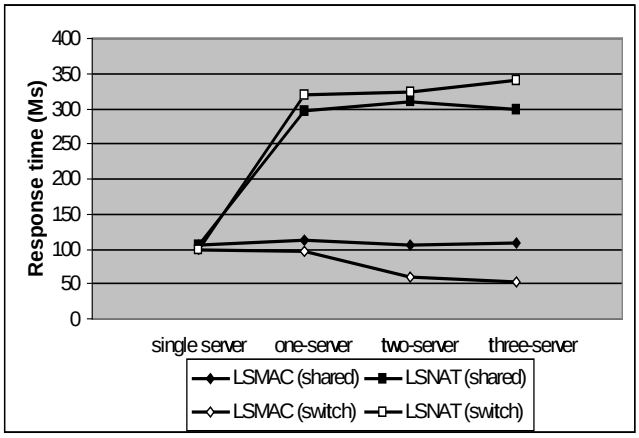

Figure 13. Response time for access log trace.

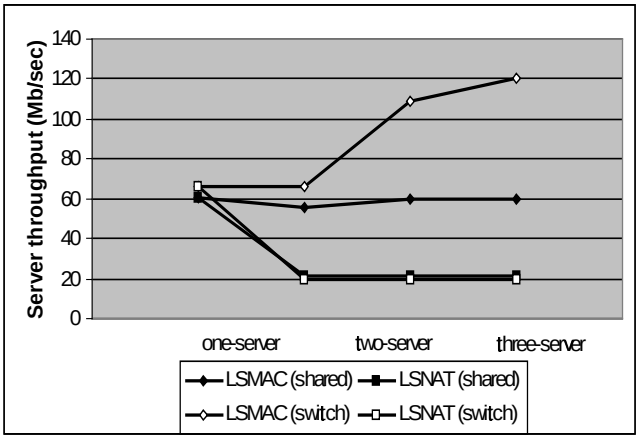

Figure 11. Server throughput for access log trace.

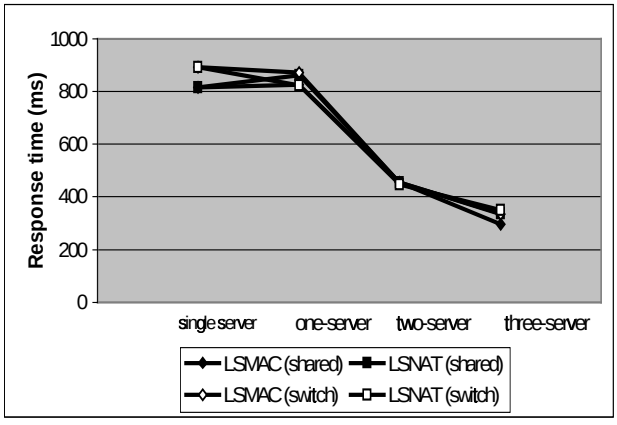

Figure 14. Response time for CGI trace.

the improvement may be small relative to the large latency involved in a wide area network, it is significant in an intranet environment. Intranet servers usually generate a lot of dynamic content such as directory query and database search results. Web servers now support several means to create dynamic content [17] (e.g., CGI, FastCGI, vendor-dependent Web server APIs, and Java servlets). These methods involve complex computation on the Web server. Since Web server processing per request has become more time-consuming, clustering is mandatory to handle even a small number of concurrent requests to a Web site [17].

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 in shared Ethernet environment over the single server due to limited network bandwidth. Hence, for cluster planning, one needs to take into account the amount and types of information maintained on the Web site.

4.2.3. Dispatcher and Server CPU Utilization

When considering performance bottlenecks, one of the simplest, and often most useful, aspects to consider is the processor utilization. In our case, processing requirements proved to be the decisive difference, in terms of performance, between LSNAT and LSMAC. For this reason, Table 2 provides a brief summary of CPU utilization on both the dispatcher and the back-end Web servers for both LSMAC and LSNAT for a peak loads of 42 clients in shared environment. This clearly illustrates the processor-intensive nature of LSNAT vis-a-vis LSMAC.

We conjecture that the low utilization of the dispatcher and back-end servers on the access log trace-driven tests for LSNAT were due to limitations in Linux's packet capture facilities which causes it to drop packets [18]. Further, we believe that the cause of the less-than-full utilization in the case of LSMAC is due to network saturation as the throughput on the shared Ethernet was in excess of 60 Mbps. Finally, the low utilization on the CGI-driven tests for both systems may be due to memory-thrashing on the back-end servers. The way in which a Web server invokes a CGI program is by forking a separate process that executes the program. Thus, a great deal of time was spent paging in and out of physical memory rather than performing useful work.

4.2.4. Dispatcher and Server on Same Host

Our solutions allow the dispatcher and the Web server to run on the same host. In LSMAC, we configure a second network interface on the dispatcher machine to the cluster IP address. The LSMAC dispatcher captures the packets from the first interface and dispatches them to the second interface or other back-end servers. One difficulty encountered during the test was the tendency of the dispatcher to reply directly to the clients, even though the packets arrived at an interface other than the one with the cluster address. Braden refers to this as the *weak end-system* model, as opposed to the *strong end-system* model where datagrams are only processed if they are received on the interface with the address specified in the datagram's destination address field [19]. We believe that this case illustrates a flaw in the weak end-system model and thus in Linux's networking which is based on this model. We used IP filtering to prevent the host's network stack from receiving these packets on the primary interface.

Tests show that combining the dispatcher and the Web server on a same host is not a good idea (Figure 15). Three stand-alone back-end servers plus one additional server on the dispatcher host (3.5 server cluster) performs worse than three stand-alone back-end servers (3 server cluster) in all the cases except for the fully dynamic content, which shows a nominal improvement. This is because the Web server on the dispatcher host takes away crucial CPU time from the dispatcher. Tests with the LSNAT dispatcher show a similar pattern (Figure 16). Thus we do not recommend running the dispatcher and Web server on a same host. Indeed, the point of clustering is to offer the ability to incrementally increase performance for very little cost. With the cost of a PC in the cluster being only $1,000 to $2,000, it's relatively cheap to add another machine to the server pool rather than trying to make the dispatcher act as a server.

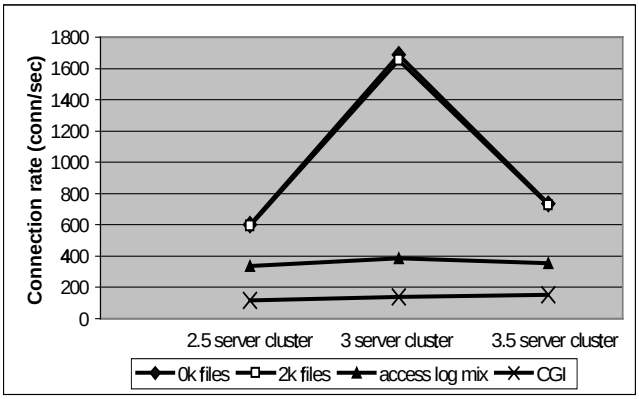

Figure 15. Connection rate with non-dedicated LSMAC.

Table 2
Processor utilization under peak loads.

File type	LSMAC (%) Dispatcher/Server	LSNAT(%) Dispatcher/Server
0 KB files	100/80	100/42
2 KB files	98/80	100/40
Access log trace	45/20	65/10
CGI	20/40	67/35

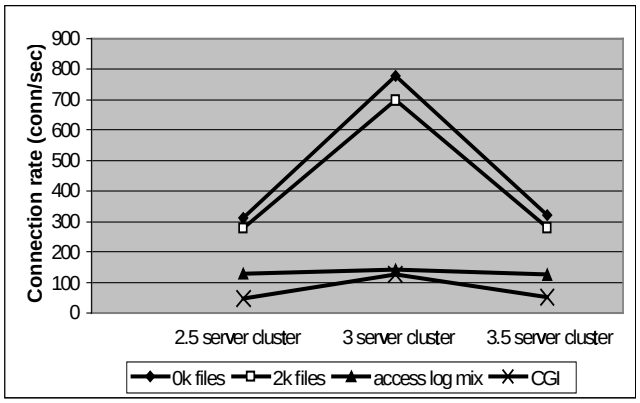

Figure 16. Connection rate with non-dedicated LSNAT.

5. 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 Web server 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. It is conceivable that dynamic content could swamp LSNAT just as static content does if we had a sufficient number of servers. 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, the LSNAT approach is more appropriate. To fully utilize the functionality provided by the dispatchers (especially LSMAC), a switched environment is needed, even though the dispatchers in shared Ethernet environment provide adequate performance. In the future we will examine fault tolerance, develop adaptive optimized load-sharing algorithms, and investigate performance improvements in LSNAT through the use of checksumming hardware.

References

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

[2] V. Sunderam, "PVM: A Framework for Parallel Distributed Computing," *Concurrency: Practice and Experience*, December 1990, pp. 315-339.

[3] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, The MIT Press, Cambridge, Massachusetts, 1996.

[4] T. Brisco, "DNS Support for Load Balancing," *RFC 1794*, April 1995.

[5] 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.

[6] P. Srisuresh and D. Gan, "Load Sharing Using IP Network Address Translation (LSNAT)", *RFC 2391*, August 1998.

[7] Cisco Systems: Local Director, http://www.cisco.com/warp/public/751/lodir/, 1999.

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

[9] D. Dias, W. Kish, R. Mukherjee, and R. Tewari, "A scalable and highly available Web server," *IEEE International Conference on Data Engineering*, New Orleans, February 1996.

[10] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-aware request distribution in cluster-based network servers", *ASPLOS-VIII*, San Jose, CA, October 1998.

[11] E. Levy-Abegnoli, A. Iyengar, J. Song, and D. Dias, "Design and performance of a Web server accelerator," *INFOCOM*, New York, NY, March 1999.

[12] Lawrence Berkeley Laboratory: Packet Capture Library, ftp://ftp.ee.lbl.gov/libcap.tar.Z, 1999.

[13] Daemon9, Libnet: Network Routine Library, http://www.packetfactory.net/libnet/, August 1999

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

[15] Mindcraft: WebStone, http://www.mindcraft.com/webstone/, 1999.

[16] 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.

[17] R. Mukherjee, "A scalable and highly available clustered Web server," In: *High performance cluster computing – vol. 1, Architectures and systems* (Ed. R. Buyya), New Jersey: Prentice Hall PTR, pp. 811-840, 1999.

[18] D. Song and M. Undy, "NFR Performance Testing," http://www.anzen.com/products/nfr/testing/, February 1999.

[19] R. Braden, "Requirements for Internet Hosts – Communication Layers," *RFC 1122*, October 1989.

Xuehong Gan received his B.S. and M.S. in Geology from Peking University and University of Alabama in 1991 and 1996, respectively. He worked for Global Internet Software Group and Cisco Systems as a Software Engineer developing Internet firewall and network management software. Later he received his M.S. in Computer Science from the University of Nebraska-Lincoln in 1999. Since August 1999, he has been working for Microsoft Corporation as an Escalation Engineer in the Internet Critical Problem Resolution Team. In his job he provides reliable resolution of the most critical and highest impact problems for Microsoft's strategic corporate customers. His research interests include network security and distributed computing. He is a member of the ACM(S).
E-mail: xuehongg@microsoft.com

Trevor Schroeder received his B.S. in Computer Science from Wayne State College in 1998. He is currently a M.S. student in Computer Science at the University of Nebraska-Lincoln while working at the MIT Media Lab with the Network+Computing Systems (NeCSys) staff where he is responsible for network security and the maintenance of UNIX machines. His research interests include distributed systems and especially security in such environments. He is a member of the ACM(S).
E-mail: tschroed@media.mit.edu.

Steve Goddard received the B.A. degree in computer science and mathematics from the University of Minnesota (1985). He received the M.S. and Ph.D. degrees in computer science from the University of North Carolina at Chapel Hill (1995, 1998). He worked as a Systems Engineer with Unisys Corporation for four years and as a real-time, distributed, systems consultant with S.M. Goddard & Co. Inc. for nine years before joining the Computer Science and Engineering faculty at the University of Nebraska-Lincoln (UNL) in 1998. He is the founding co-director of the Advanced Networking and Distributed Experimental Systems (ANDES) Laboratory at UNL. His research and teaching interests are in real-time systems, distributed systems, operating systems, software engineering and computer networks.
E-mail: goddard@cse.unl.edu.

Byrav Ramamurthy received his B.Tech. degree in Computer Science and Engineering from Indian Institute of Technology, Madras (India) in 1993. He received his M.S. and Ph.D. degrees in Computer Science from University of California (UC), Davis in 1995 and 1998, respectively. Since August 1998, he has been an assistant professor in the Department of Computer Science and Engineering at the University of Nebraska-Lincoln (UNL). He is the founding co-director of the Advanced Networking and Distributed Experimental Systems (ANDES) Laboratory at UNL. He is the Feature Editor on Theses for the Optical Network Magazine. He serves as a guest co-editor of an upcoming special issue of IEEE Network magazine. He served as a member of the technical program committees for the IEEE GLOBECOM'99 Symposium on High Speed Networks, the IEEE International Conference On Networks (ICON'99) Conference, and the Eighth IEEE International Conference on Computer Communication and Network (ICCCN'99). Prof. Ramamurthy was a recipient of the Indian National Talent Search scholarship and was a fellow of the Professors for the Future program at UC Davis. He is a recipient of the UNL Research Council Grant-in-Aid award for 1999. His research areas include optical networks, distributed systems, and telecommunications.
E-mail: byrav@cse.unl.edu.