LSMAC: An improved load sharing network service dispatcher

Xuehong Gan[a] and Byrav Ramamurthy[b]

[a] *Microsoft Corporation, One Microsoft Way, Redmond WA 98052, USA*
E-mail: xuehongg@microsoft.com

[b] *Department of Computer Science and Engineering, University of Nebraska – Lincoln, Lincoln, NE 68588, USA*
E-mail: byrav@cse.unl.edu

The rapid growth of the Internet is changing the way we do business. Electronic Commerce (or E-Commerce) is already a reality and will expand rapidly in the near future. However, the success of E-Commerce depends heavily on the scalability and availability of the servers. Cluster-based servers using commodity hardware have been accepted as a good alternative to expensive specialized hardware for building scalable services. In this paper, we summarize the two clustering architectures: IP-based clustering and MAC-based clustering. A new efficient implementation of the MAC-based clustering architecture is presented and its performance in clustering Web servers was measured using the WebStone benchmark and was found to be superior to that of existing MAC-based clustering implementations.

1. Introduction

The Internet is changing every aspect of our lives including transforming the traditional business processes into E-Commerce applications. Today, more and more companies are using the Internet to communicate with their partners, to transact business, and to provide customer services. The function performed by the network servers such as *a World Wide Web (WWW) server* is critical to a company's business. Busy corporate web-sites will need to handle millions of "hits" on their servers daily. Other companies may not need such a powerful server on a day-to-day basis, but they will need to handle potentially a very large peak load. For example, a sports magazine's site may need to handle millions of requests during the Olympic games. Server overload is frustrating to the customers, and harmful to the company's bottomline.

The first option many companies use to scale their network services 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. 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 personal computer systems is a cheap alternative to upgrading a single high-end server with faster hardware. Busy sites such as Excite, Inc. depend heavily on clustering technologies to handle a large number of requests [Bruno 1997].

In the Internet, each server (single-homed) has a hostname and its corresponding Internet Protocol (IP) address, and a Medium Access Control (MAC) address. The clients communicate with the server though the IP address or the hostname. The mapping between IP addresses and hostnames is handled by Domain Name Service (DNS). In the usual case (i.e., a non-clustered server), there is only one server serving the requests addressed to one hostname or IP address. With a cluster-based server, several back-end servers cooperatively serve the requests addressed to the hostname or IP address corresponding to the network service. 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".

Cluster-based servers can provide high availability and good scalability. Regarding availability, cluster-based servers can handle a large number of concurrent requests and reduce the request latency. Also, the dispatchers can detect malfunctioning servers in real-time and remove them from the server pool. Availability is a key factor for customer satisfaction for online businesses. Regarding scalability, the administrators can easily add or remove servers according to business demands.

The rest of this paper is organized as follows. We first discuss related work in section 2, which summarizes the two clustering architectures: IP-based clustering and MAC-based clustering. Then we describe our implementation of the MAC-based clustering architecture in section 3. Section 4 describes how we evaluated our implementation and presents the results. We present our conclusions and describe future work in section 5.

2. Related work

Until recently, clustering has been typically used in proprietary server environments to improve application uptime. For clustering, two or more servers are combined into a configuration where one server takes on the workload of another in case of a hardware or software failure. 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, and little scalability. Some products support rudimentary scalability. But they usually require developers to rewrite applications to be cluster-aware with vendor-specific Application Programming Interfaces (APIs).

In contrast, a new generation of more generic server clustering systems are hitting the market. These systems provide scalability as well as availability without these limitations – the server software does not have to be cluster-aware; i.e., the servers can run any combination of operating systems on any mix of hardware. At the same time, these systems can provide server fail-over and fault tolerance just as the high-end server-clustering does. These new generation clustering technologies can be divided into two categories: Round Robin Domain Name Service (RR-DNS) [Brisco 1995] and Single-IP-Image [Damani *et al.* 1997]. 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 [Dias *et al.* 1996]. Even though these systems are primarily being marketed and used as Web load-sharing systems, they could just as easily be configured and/or modified to provide load sharing to other network services. We discuss each of these techniques below.

2.1. Round Robin Domain Name Service (RR-DNS)

Early implementations of the cluster-based server concept used RR-DNS. RR-DNS is a hostname-based approach. It works by mapping a single hostname of the server to several different IP addresses though 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. 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. Even if the DNS zone file is immediately modified after a server failure, the problem still arises due to DNS caching. The IP address has been cached by local DNS servers across the Internet; for the next few minutes, many users get error messages instead of connecting to one of the available servers. Secondly, clients themselves may cache DNS replies and bypass DNS for subsequent requests, which defeats the attempts at load sharing using the round robin mechanism.

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 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 n back-end servers. The methods differ in the way they forward packets to a back-end server. Currently there are two major schemes: IP-based dispatching and MAC-based dispatching.

In IP-based approaches, 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 to the IP address of the chosen back-end server and recalculating the IP and Transport Control Protocol (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 RFC2391, Load Sharing Using IP Network Address Translation (LSNAT) [Srisuresh and Gan 1998]. A commercial example of the LSNAT approach is Cisco's Local Director [Cisco 1999]. A slight variation of this approach was proposed for IBM's TCP Router [Attanasio and Smith 1992], 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.

In MAC-based approaches, all servers in the cluster share the cluster address as a secondary IP address while the dispatcher is assigned a different address. 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. The secondary IP address assignment can be accomplished using interface aliases (e.g., the *ifconfig* command) on most Unix systems. The dispatcher controls the MAC addresses of the frames carrying the request packets and then forwards the frames over a local area network (LAN) to an appropriate back-end server. 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. Different MAC-based approaches vary in the mechanism

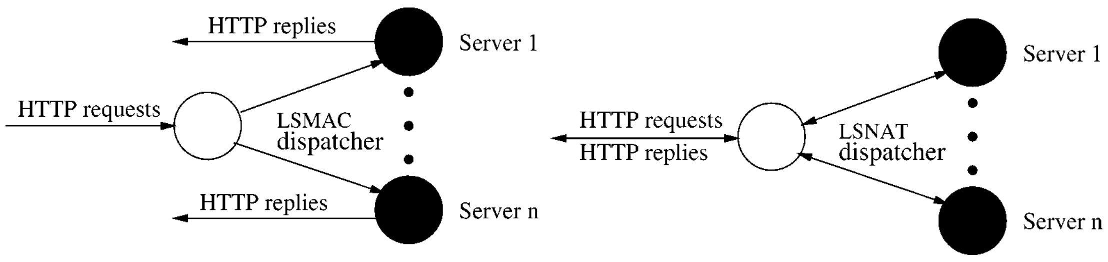

Figure 1. Logical representations of LSNAT and LSMAC dispatchers.

for controlling the MAC addresses. It appears that IBM eNetwork Dispatcher [Hunt *et al.* 1998; IBM 1999] modifies the routing algorithm for the cluster address so that it routes packets based on the primary IP addresses of back-end servers. This method was first proposed for ONE-IP [Damani *et al.* 1997], even though their use of IP aliasing is slightly different. In addition, ONE-IP proposed another MAC-based 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 with this method is that filtering reduces the capacity of each server to serve client requests. Our implementation of the MAC-based clustering differs from both methods in that it directly rewrites the MAC addresses of each frame (see section 3.1). We call our dispatcher LSMAC. LSMAC takes its name because it realizes Load Sharing Using Medium Access Control.

MAC-based approaches (e.g., LSMAC) have several advantages over IP-based approaches (e.g., LSNAT). First, the outgoing packets do not need to traverse the dispatcher (figure 1). Thus, the outgoing packets can be directed to take a different route from the incoming packets, so that the network contention is eased. Secondly, 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 WWW, i.e., requests are small while the server responses are much larger. On the other hand, MAC-based approaches have their own disadvantage: they require the dispatcher and back-end servers to be interconnected in a LAN. In contrast, IP-based approaches allow the dispatcher and back-end servers to be in different internal LANs, since IP packets can be routed even across LANs. However, in the case of different LANs, additional care is needed to set up the routing rules correctly so that all traffic directed to the cluster address and all replies from back-end servers traverse the IP-based dispatcher.

3. LSMAC implementation

LSMAC differs from other MAC-based clustering approaches in that it directly modifies the MAC addresses, while other MAC-based implementations usually customize the dispatcher host's IP routing algorithm. LSMAC has two advantages over other MAC-based approaches: (1) it simplifies the packet routing; (2) it avoids checksum recalculations. These two advantages speed up the packet dispatching process. It operates only on the Data-Link layer and dispatches each incoming frame by directly modifying its MAC addresses (both source and destination).

3.1. Dispatching process

Each TCP session consists of multiple packets. Once a server is selected for the first packet of one request, future incoming packets for the same request must be directed to the same server. The dispatcher maintains a table containing information about all existing sessions. Upon receipt of a packet, the dispatcher will determine whether a corresponding connection has already been established with a back-end server. If such a session does not already exist, it is simply a matter of creating a new entry in the dispatcher's session 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 headers. 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 it to a chosen back-end server. The choice of a particular back-end server can be made in a round robin fashion or based on the actual server loads. The back-end server handles the packet in a normal fashion and replies directly to the client. Figure 2 illustrates an example of the packet flow in a LSMAC cluster-based server. In this example, the back-end server is aliased to the cluster address 129.93.1.2. The dispatcher is assigned a different IP address (129.93.1.10). We can see that the IP header is not changed before entering and after leaving the dispatcher. However, the dispatcher changes the MAC addresses of the packet; it places its own MAC address (e3:67:89:a1:2b:04) as the source MAC address of the packet and the back-end server's MAC address (a2:c2:04:05:56:13) as the destination MAC address. This modification enables the back-end server to receive the packet.

The back-end servers behave as if they were communicating directly with the clients and do not need to know

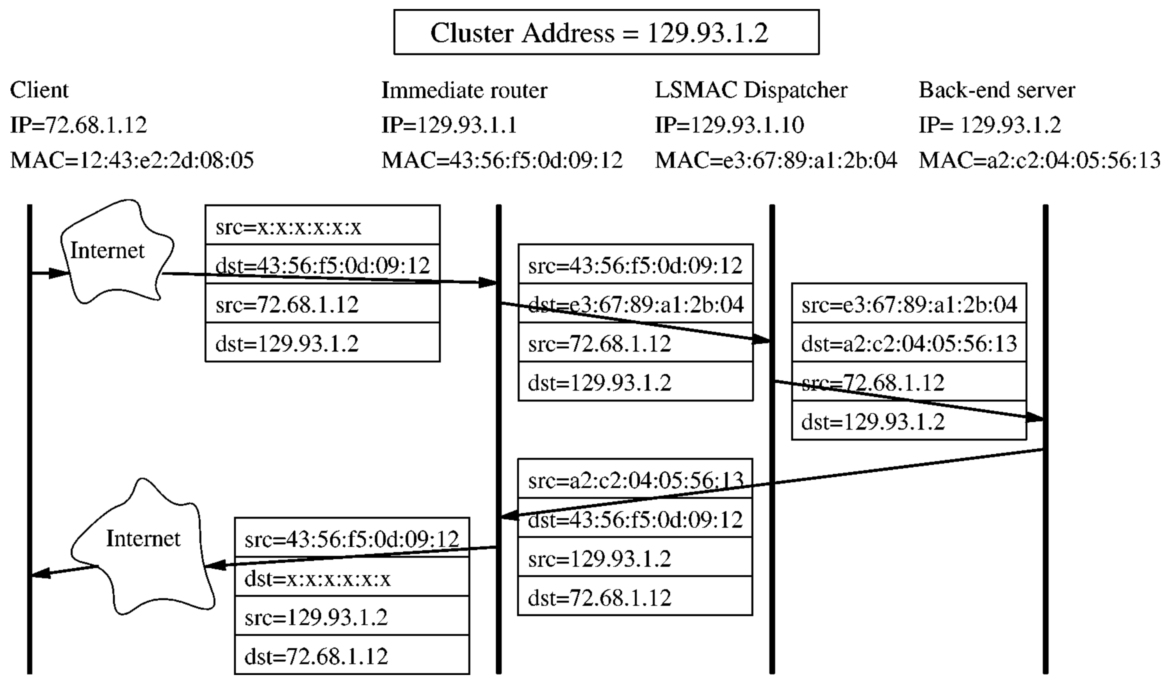

Figure 2. Packet modifications in a LSMAC cluster-based server.

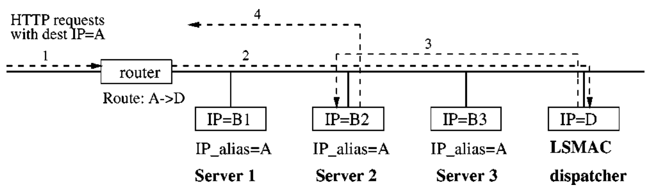

Figure 3. LSMAC dispatcher in a LAN environment.

anything about the clustered nature of the system. This means that no special software needs to be installed on the back-end servers. The LSMAC dispatcher is thus transparent to the clients and the servers.

In summary, the following steps are involved in LSMAC dispatching (figure 3).

1. A client sends a HTTP packet with the well-known cluster IP address *A* as the destination IP address.
2. The immediate router sends the packet to the LSMAC dispatcher at IP address *D*, using the added route rule: $A \rightarrow D$.
3. Based on the load sharing algorithm and the session table, the LSMAC dispatcher decides that this packet should be handled by the server (say *B2*), and sends the packet to *B2* by modifying the MAC addresses of the frame.
4. The server *B2* accepts the packet and replies directly to the client without going through the LSMAC dispatcher again.

Even though we focus on clustering Web servers in benchmarking LSMAC in section 4, the LSMAC dispatcher can be used to balance any TCP/IP-based network service. In general, those protocols which do not open additional ports during a TCP session, can be handled by LSMAC as described above. For example, we employ the LSMAC dispatcher to “load share” TELNET sessions to our research workstation-cluster. For others, which do open additional ports, more processing is needed for the session lookup. For example, in FTP, the client and the server send commands on a control connection with port 21 (at the server), but they transfer data on a separate data connection with port 20 (at the server). The incoming packets on the data connection from a client must go to the same back-end server, to which the client is connected to on its control connection.

In comparison to other MAC-based approaches, our approach simplifies the packet routing and avoids the checksum recalculations. These two advantages can largely improve the dispatcher performance, considering that these two operations need to be performed on each and every packet.

3.2. Implementation considerations

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 particular connections already established with the back-end servers. A hash table (figure 4) is used to store the association between client requests and back-end servers.

For studying load sharing, the prototype utilized two algorithms: round robin and least-connections. The round robin algorithm selects the back-end server as the target in a circular fashion. The least-connections algorithm counts the number of live TCP sessions at each back-end server. When a new TCP session request comes in, the dispatcher selects the back-end server with the least number of live

```
struct   hentry {
         u_int           saddr;          /* source IP address */
         u_int           daddr;          /* destination IP address */
         u_short         sport;          /*source port addesss */
         u_short         dport;          /*destination port address */
         u_int           serverid;       /* id of the selected server */
         u_int           timestamp;      /* starting time of the session */
         struct   hentry *next;          /* pointer to the next entry */
}
```

Figure 4. Data structure for associations between sessions and servers.

TCP sessions as the target. For measuring the performance, we use the round robin algorithm to distribute the load amongst the entire set of back-end servers. This works well since all of our servers are configured in a similar fashion and the requests from the 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.

The LSMAC dispatcher requires the MAC address of each back-end server for forwarding packets. For easy management, the Address Resolution Protocol (ARP) is used in our prototype to automatically discover back-end servers. The administrator specifies the cluster address and load-sharing algorithm when invoking the LSMAC dispatcher. The LSMAC dispatcher will broadcast an ARP request for the cluster address and retrieve the back-end servers' MAC addresses from their ARP responses. In addition, the dispatcher periodically broadcasts ARP requests and processes ARP replies to check the server availability. It also captures Internet Control Message Protocol (ICMP) messages, which will detect the case where the server could be down while its ARP still works. This approach enables administrators to add (remove) servers to (from) the cluster dynamically, and enable the dispatcher to detect dead servers and remove them from the server pool at running time. Additional details of the LSMAC prototype implementation can be found in Gan [1999].

4. Evaluation

We evaluated the performance of the prototype in clustering Web servers. Web servers play a vital role in enabling E-Commerce. All Web servers (clustered and non-clustered) communicate with clients (browsers) using the Hypertext Transfer Protocol (HTTP) [Berners-Lee *et al.* 1996]. The client establishes a connection to a TCP port of the server (by default, port 80), and no other ports are opened during the session.

WebStone [Mindcraft 1999] was used to benchmark the performance of our cluster-based server systems. WebStone is a configurable load generator for Web servers. This freely available benchmark simulates varying client loads to measure a server's connection rate, throughput, and request latency.

Any one of the dispatcher, the back-end servers, or the network can "bottleneck" the operation of a cluster-based server system. In order to evaluate the capability of a dispatcher correctly, we need to make sure that neither the back-end servers nor the network bandwidth "bottlenecks" the system and thus cuts the performance of the dispatcher. Dynamic content such as Common Gateway Interface (CGI) could easily "bottleneck" the system, since a CGI program runs as a separate process in the server machine every time a CGI document is requested and therefore is very computation-intensive. Similarly, requests for large files could cause network saturation. The server throughput is more related to the network bandwidth capability than to the dispatcher capability. Request latency does not play a significant role in the performance improvement achieved using the dispatcher, considering the large latency encountered in a Wide Area Network (WAN). Thus, the connection rate with small static files would be the best indicator of the performance of the dispatcher. The server connection rate is expressed as connections per second. This is an indication of how fast the server system can establish a connection and start communicating with the clients.

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 only transfer a few packets. Thus the number of simultaneous connections supported has more to do with memory available at the dispatcher than its ability to forward packets quickly. A dispatcher usually uses less than 32 bytes of memory to keep track of each TCP session (24 bytes in our implementation, see figure 4). 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, a network bandwidth of 1 Gbps would be required!

4.1. Experiment design

In our experiments, the dispatcher and three standalone 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 or a switched 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. We chose four types of server files: 0 KB files that have no payload but still require HTTP headers, 2 KB files which are typical of the first page at a Web server, a file mix with file sizes and access frequencies derived from a Web server access log (available from

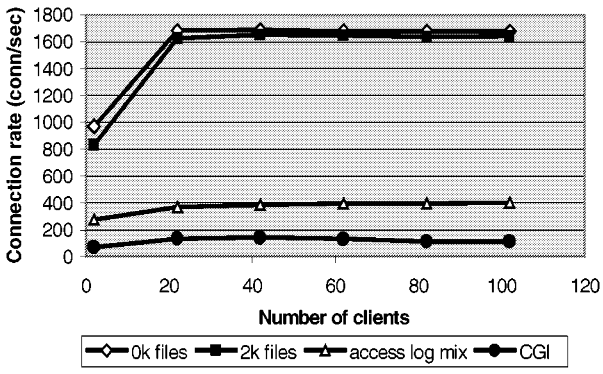

Figure 5. Connection rates in shared Ethernet environment.

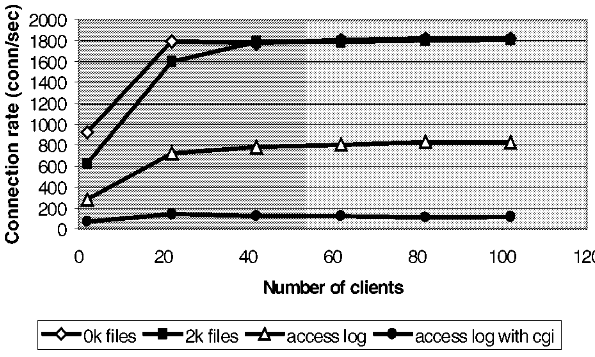

Figure 6. Connection rates in switched Ethernet environment.

Mindcraft [Mindcraft 1999]), and purely dynamic files. The dynamic files were generated by a CGI program based on file sizes and access frequencies derived from the same Web server access log. A performance comparison of the LSMAC dispatcher and a LSNAT dispatcher can be found in [Gan *et al.* 2000].

4.2. Server connection rate

In a cluster-based server there is one or more back-end 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) show that the dispatcher with three servers can handle over 1600 connections per second in shared environment (figure 5) and reach 1800 connections per second in switched environment (figure 6) at close to 100 percent CPU utilization. The connection rate in a single server configuration with the same file size is around 550 connections per second. With a 2-byte page size, IBM Network Dispatcher can handle 815 connections per second when it runs on a SP-2 machine in a shared Token Ring environment [Hunt *et al.* 1998]. Each SP-2 node has a POWER2 67 MHz CPU and 256 MB memory. Our implementation achieved superior performance because it avoids computation-intensive checksum recalculations.

However, with the access log file mix, whose average file size is 108.5 KB, the cluster-based server does not improve the connection rate in a shared environment due to

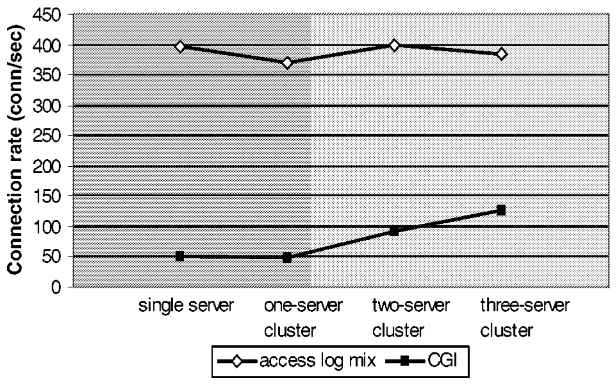

Figure 7. Connection rates with varying number of back-end servers.

network collisions. The dispatcher with three servers maintains about 400 connections per second (figure 5), which is very close to the connection rate of a single server. 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. But, with the same log file mix in a switched environment, the LSMAC dispatcher reaches near 800 connections per second (figure 6). This test points out that a higher capacity network is needed in order to fully utilize the functionality provided by the LSMAC dispatcher.

It is interesting to notice that the cluster-based server scales better with dynamic content than static content. The connection rate increases when more back-end servers are added to the cluster in the CGI case. Figure 7 shows the connection rate changes with varying number of back-end servers for both static and dynamic content in a shared environment under 42 web clients. Apparently adding more servers eases the bottleneck of the back-end servers in the CGI case. Thus, clustering is very useful in scaling Web sites with large amounts of dynamic content, which plays an important role in nearly all E-Commerce Web sites.

5. Conclusions

The scalability and availability of network services are becoming increasingly important as more people get connected to the Internet. Cluster-based servers can achieve good scalability and high availability at a low cost. Currently most cluster-based servers use either an IP-based or a MAC-based approach. In this paper we presented LSMAC, a new variation of the MAC-based clustering architecture. Our approach using direct MAC address modification simplifies the packet routing and avoids checksum recalculations, thus speeding up the dispatching process. Based on Webstone measurements, the LSMAC prototype was found to achieve superior performance to existing MAC-based clustering products. The experiments also show that a high capacity network is necessary for implementing load sharing network services, and a cluster-based web server scales better for dynamic content than static content. We predict that there will be a large deployment of cluster-based servers in the near future due to the rapid expansion of E-Commerce. Our future work will include incorporating fault tolerance and handling non-TCP traffic.

Acknowledgement

The authors thank Trevor Schroeder and Professor Steve Goddard (of UNL) for their valuable comments and help with performance measurements of the prototype.

References

Attanasio, C.R. and S.E. Smith (1992), "A Virtual Multiprocessor Implemented by an Encapsulated Cluster of Loosely Coupled Computers," IBM Research Report RC18442, IBM Research, Yorktown Heights, NY.

Berners-Lee, T., R. Fielding, and H. Nielson (1996), "Hypertext Transfer Protocol – HTTP/1.0," RFC 1945, Internet Engineering Task Force.

Brisco, T. (1995), "DNS Support for Load Balancing," RFC 1794, Internet Engineering Task Force.

Bruno, L. (1997), "Balancing the Load on Web Servers," *Data Communications*, September 21, `http://www.data.com`

Cisco Systems (1999), "Local Director," `http://www.cisco.com/warp/public/751/lodir/`

Damani, O.P., P.E. Chung, Y. Huang, C. Kitala, and Y. Wang (1997), "ONE-IP: Techniques for Hosting a Service on a Cluster of Machines," *Computer Networks and ISDN Systems 29*, 1019–1027.

Dias, D., W. Kish, R. Mukherjee, and R. Tewari (1996), "A Scalable and Highly Available Web Server," In *Proceedings of the IEEE Computer Conference (COMPCON)*, IEEE Computer Society Press, Los Alamitos, CA, pp. 85–92.

Gan, X. (1999), "A Prototype of a Web Server Clustering System," MS Project Report, Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE.

Gan, X., T. Schroeder, S. Goddard, and B. Ramamurthy (2000), "LSMAC vs. LSNAT: Scalable Cluster-based Web Servers," *Cluster Computing*, to appear.

Hunt, G., G. Goldszmidt, R. King, and R. Mukherjee (1998), "Network Dispatcher: A Connection Router for Scalable Internet Service," *Computer Networks and ISDN Systems 30*, 347–357.

IBM (1999), "eNetwork Dispatcher," `http://www.software.ibm.com/network/dispatcher/`

Mindcraft (1999), "WebStone," `http://www.mindcraft.com/webstone/`

Srisuresh, P. and D. Gan (1998), "Load Sharing Using IP Network Address Translation (LSNAT)," RFC 2391, Internet Engineering Task Force.