6-2000

Scalable Web Server Clustering Technologies

Trevor Schroeder
University of Nebraska-Lincoln

Steve Goddard
University of Nebraska-Lincoln, goddard@cse.unl.edu

Byrav Ramamurthy
University of Nebraska-Lincoln, bramamurthy2@unl.edu

Digital Object Identifier: 10.1109/65.844499

Scalable Web Server Clustering Technologies

Trevor Schroeder, Steve Goddard, and Byrav Ramamurthy
University of Nebraska-Lincoln

Abstract

The exponential growth of the Internet, coupled with the increasing popularity of dynamically generated content on the World Wide Web, has created the need for more and faster Web servers capable of serving the over 100 million Internet users. Server clustering has emerged as a promising technique to build scalable Web servers. In this article we examine the seminal work, early products, and a sample of contemporary commercial offerings in the field of transparent Web server clustering. We broadly classify transparent server clustering into three categories.

The exponential growth of the Internet, coupled with the increasing popularity of dynamically generated content on the World Wide Web, has created the need for more and faster Web servers capable of serving the over 100 million Internet users.

The only solution for scaling server capacity in the past has been to completely replace the old server with a new one. Organizations must discard their investment in the old server and purchase a new one — an expensive, short-term solution. A long-term solution requires incremental scalability, which provides the ability to grow gradually with demand.

A pool of servers tied together to act as a single unit, or *server clustering*, provides such incremental scalability. Service providers may gradually add additional low-cost computers to augment the performance of existing servers. As Internet usage has grown, so has investigation into Web server clustering. The past four years have seen the emergence of several promising experimental server clustering approaches as well as a number of commercial solutions.

All Web server clustering technologies are transparent to client browsers (i.e., the client browsers are unaware of the existence of the server cluster). However, not all clustering technologies are transparent to the Web server software. Early commercial cluster-based Web servers such as Zeus and Inktomi [1] are, in many respects, continuations of the traditional approach to cluster-based computing: treat the cluster as an indissoluble whole rather than the layered architecture assumed by (fully) transparent clustering. Thus, while transparent to clients, these systems are not transparent to the server nodes and require specialized software throughout the system.

For example, Inktomi has a central point of entry and exit for requests, but nodes in the cluster are specialized to perform certain operations such as image manipulation and document caching. There is a coordinator that coordinates all the nodes to service client requests. In a similar vein, the Zeus Web server provides server clustering for scalability and availability, but each server node in the cluster must be running the Zeus Web server, a specialized server software developed for this environment.

The cost and complexity of developing such proprietary systems is such that while they provide improved performance over a single-server solution, they cannot provide the flexibility and low cost service providers have come to expect with the wide array of Web servers and server extensions available. For this reason, our emphasis is on solutions that allow service providers to utilize commodity hardware and software. This implies that the clustering technique must be transparent to both the Web client and the Web server since the overwhelming majority of Web servers do not have any built-in clustering capabilities.

While the emphasis of this article is on clustering in a Web server context, the technology is more generally applicable. Any server application may be clustered as long as it fulfills the following two properties:

- The application must maintain no state on the server. Any state information that is maintained must be maintained by the client. This prevents the cluster from having to deal with distributed state consistency issues. Note that some clustering agents do provide the capacity for some stateful services, but this is done on a service-by-service basis and is very protocol-specific.
- Client/server transactions should be relatively short and high in frequency. As we are interested in commodity systems (hardware and software), we cannot decompose transactions into any smaller operations. Therefore, it is required that the transactions themselves be relatively small so that we can employ stochastic distribution policies to share the load more or less equally among all servers.

0890-8044/00/$10.00 © 2000 IEEE

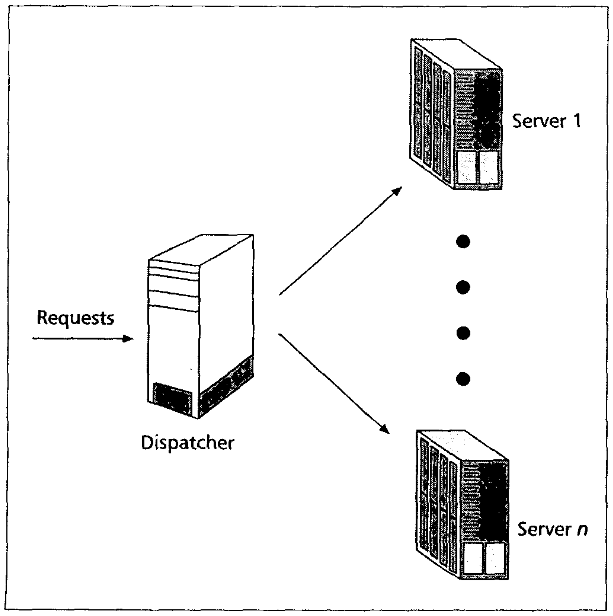
Terminology

The terminology used to describe server clustering mechanisms varies widely. Some refer to it as *application-layer switching* or *layer 4–7 switching* (e.g., Alteon Web Systems product literature); others refer to it as *server load balancing* [2]; still others refer to it as, simply, *clustering*. The term application-layer switching is inadequate in that it is not clear exactly how application-layer switching actually takes place. The term server load balancing encompasses a wide range of technologies (not necessarily related to networks). Moreover, the solutions we consider provide load sharing rather than load balancing. That is, they attempt to ensure that the load is more evenly distributed, but do not attempt a completely even distribution. This arises due to the fact that the load sharing is on the granularity of a single client request. Finally, the term clustering by itself is too general.

Instead of these terms, we use the terms *layer four switching with layer two packet forwarding* (L4/2), *layer four switching with layer three packet forwarding* (L4/3), *and layer seven* (L7) *switching with either layer two packet forwarding* (L7/2) *or layer three packet forwarding* (L7/3) clustering. These terms refer to the techniques by which the systems in the cluster are tied together. In an L4/2 cluster, the systems are identical above open systems integration (OSI) layer two (data link). That is, each system has a unique layer two (i.e., medium access control, MAC) address, but identical layer three (network) addresses, and identical services are provided. In an L4/3 cluster, each system has a unique network address but still offers the same services. L7 clusters may — but do not have to — employ L4/2 or L4/3 clustering *in addition* to potentially different offerings among the back-end servers. The clustering agent uses information contained in the client/server transaction to perform load sharing.

An Overview of Transparent Clustering

In each of the network clustering technologies discussed in this article, one entity sits on the network and acts as a proxy for incoming connections, as shown in Fig. 1. We call this entity the *dispatcher*. The dispatcher is configured with a particular network address, called the *cluster address*. The servers appear as a single host to clients because of the dispatcher (client-side transparency). The dispatcher receives service requests from clients and selects a server from the server pool to process the request. Depending on the clustering technology, the dispatcher appears as either a switch (processing incoming data only) or a network gateway (processing incoming and outgoing data) to the servers in the pool. In either case, we assume each server is executing standard Web server software designed for a standalone server (server-side transparency). Incoming client requests are distributed more or less evenly to the pool of servers. This is made possible by protocols such as HTTP which typically have small requests (thus allowing load sharing at the request level to achieve relatively even loading of the servers) and save no state information (thus allowing a client to utilize multiple servers to service a set of requests without having to manage consistency between the servers).



■ Figure 1. *A high-level view of a basic server cluster showing the dispatcher and* n *servers in the server pool.*

Vendor	URL
Alteon Web Systems	http://www.alteonwebsystems.com
ArrowPoint Communications	http://www.arrowpoint.com
Cabletron Systems	http://www.ctron.com
Cisco Systems	http://www.cisco.com
Intel	http://www.intel.com/network
Zeus Technology	http://www.zeus.co.uk

■ Table 1. *Vendors providing clustering devices.*

Performance Comparisons

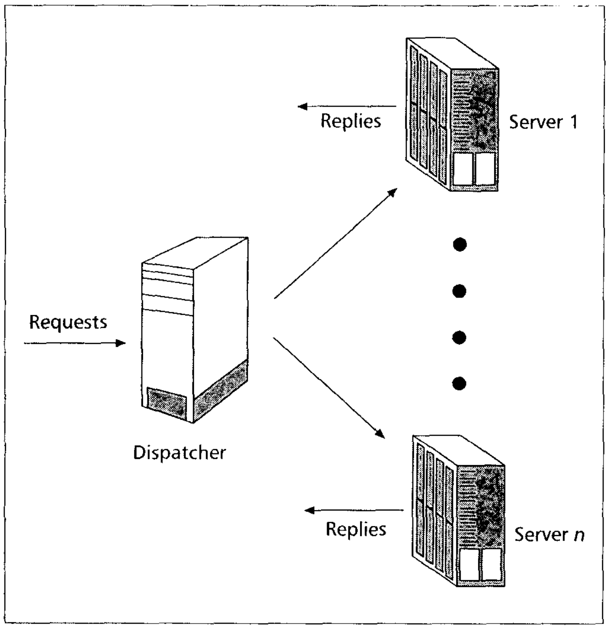
Server clustering is an area of technology that is expanding rapidly, with new commercial products appearing on the scene with regular frequency. It is our intent to present the seminal work in the field, early products, and a sampling of contemporary commercial offerings. Table 1 lists the vendors offering server clustering products discussed in this article. We report the performance of various clustering technologies, when possible, using the broad classification metrics of connections per second or throughput in bits per second. In some of the seminal research projects, performance numbers are either not available or outdated. We have not personally evaluated each product, let alone performed head-to-head comparisons of similar products. Rather, this article summarizes performance results reported by the product's developers, which (in most cases) have not been independently verified.

It should be noted that performance of clustering technologies can vary significantly depending on server configurations, client mix, test duration, content, and so on. Thus, performance metrics such as connections per second or throughput represent a broad classification of performance and provide only a relative measure of performance.

The rest of our presentation is organized as follows. We present clustering techniques which operate at OSI layer two (data link layer). We also present OSI layer three (network layer) approaches. We present solutions operating at OSI layer seven (application layer), and then present our conclusions.

L4/2 Clustering

Some of the earliest research in transparent clustering was based on L4/2, which provides excellent performance and a high degree of scalability [3]. In L4/2 clustering, the cluster



■ Figure 2. *A high-level view of traffic flow in an L4/2 cluster.*

network-layer address is actually shared by the dispatcher and all of the servers in the pool through the use of primary and secondary Internet Protocol (IP) addresses. That is, while the primary address of the dispatcher is the same as the cluster address, each server is configured with the cluster address as a secondary address. This may be done through the use of *interface aliasing* or by changing the address of the loopback device on the servers in the server pool. The nearest gateway is then configured such that all packets arriving for the cluster address are addressed to the dispatcher at layer two. This is typically done with a static Address Resolution Protocol (ARP) cache entry.

If the packet received corresponds to a TCP/IP connection initiation, the dispatcher selects one of the servers in the server pool to service the request (Fig. 2). Server selection is based on some load sharing algorithm, which may be as simple as round-robin. The dispatcher then makes an entry in a connection map, noting the origin of the connection, the chosen server, and other information (e.g., time) that may be relevant. The layer two destination address is then rewritten to the hardware address of the chosen server, and the frame is placed back on the network.

If the incoming packet is not for connection initiation, the dispatcher examines its connection map to determine if it belongs to a currently established connection. If it does, it rewrites the layer two destination address to be the address of the server previously selected and forwards the packet as before. In the event that the packet does not correspond to an established connection but is not a connection initiation packet itself, it is dropped.

Note that these are general guidelines, and actual operation may vary. For example, the dispatcher may simply establish a new entry in the map for all packets that do not map to established connections, regardless of whether or not they are connection initiations.

The traffic flow in an L4/2 clustered environment is illustrated in Fig. 3 and summarized as follows:

- A client sends an HTTP packet with A as the destination IP address.
- The immediate router sends the packet to the dispatcher at A.
- Based on the load sharing algorithm and the session table, the dispatcher 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 address of the packet to server 2's MAC address and forwarding it.
- Server 2 accepts the packet and replies directly to the client.

L4/2 clustering realizes a tremendous performance advantage over L4/3 clustering (to be discussed later) because of the downstream bias of Web transactions. Since the network address of the server to which the packet is delivered is identical to the one the client used originally in the request packet, the server handling that connection may respond directly to the client rather than through the dispatcher. Thus, the dispatcher processes only the incoming data stream, a small fraction of the entire transaction. Moreover, the dispatcher does not need to recompute expensive integrity codes (e.g., IP checksums) in software since only layer two parameters are modified. Thus, the scalability of the server is primarily limited by network bandwidth and the dispatcher's sustainable request rate, which is the only portion of the transaction actually processed by the dispatcher.

A restriction on L4/2 clustering is that the dispatcher must have a direct physical connection to all network segments which house servers (due to layer two frame addressing). This contrasts with L4/3 clustering (as we show in the next section), where the server may be anywhere on any network with the sole constraint that all client-to-server and server-to-client traffic must pass through the dispatcher. In practice, this restriction on L4/2 clustering has little appreciable impact since servers in a cluster are likely to be connected via a single high-speed LAN anyway.

Among research and commercial products implementing layer two clustering are ONE-IP developed at Bell Laboratories, IBM's eNetwork Dispatcher, LSMAC from the University of Nebraska-Lincoln, and ACEdirector from Alteon. We describe these in detail below.

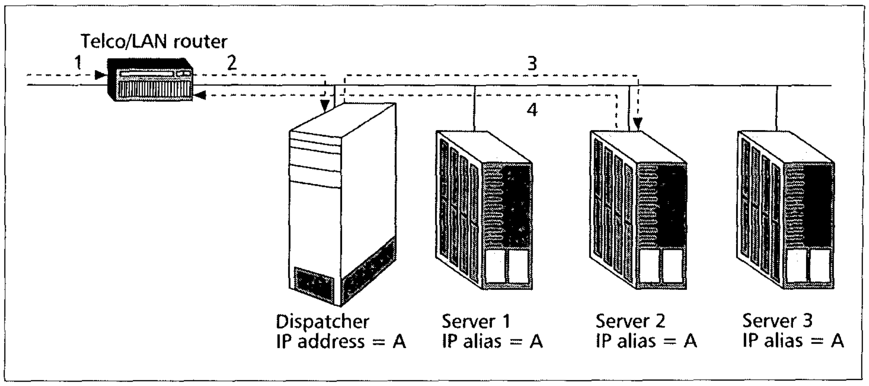
ONE-IP

One of the first implementations of layer two clustering was ONE-IP developed at Bell Laboratories (circa 1996) [4]. ONE-IP uses a modified NetBSD kernel to support two different dispatching methods. With the first method, when a packet is received by the dispatcher, the client's address is hashed to obtain a value indicating which server in the server pool will service the request. The second dispatching method broadcasts packets destined for the cluster on the LAN that connects the dispatcher with the pool of servers. Each server in the pool implements a filter on the client address such that a server only responds to a fixed and disjoint portion of the address space. Neither of these algorithms is able to adapt to conditions when clients disproportionately load the server.

ONE-IP supports fault tolerance for both the dispatcher and the servers through the use of a watchdog daemon. When a server fails, the dispatcher does one of two things. If it is using the first dispatching method, it modifies the hash table to take into account the reduced server pool. If the dispatcher is using the second (broadcast-based) method, it informs the entire server pool of the failed server. Each server then changes its filter accordingly. In the event of a dispatcher failure, a backup dispatcher will notice the missing dispatcher heartbeat messages and take over. Since there is no state information, none needs be replicated or rebuilt, and the failover is simple and fast.

eNetwork Dispatcher

A commercial product based on L4/2 clustering is IBM's eNetwork Dispatcher, unveiled in 1996. The eNetwork Dispatcher successfully powered the 1998 Olympic Games Web site. It serviced up to 2000 requests/s in this role, although experimental results show it capable of serving up to 2200 requests/s [5]. The eNetwork Dispatcher runs on a single node of an IBM SP-2 parallel computer. It uses a weighted round-robin algorithm to distribute connections to the servers in the server pool. Periodically, it recomputes the weights based on load metrics collected from the servers.



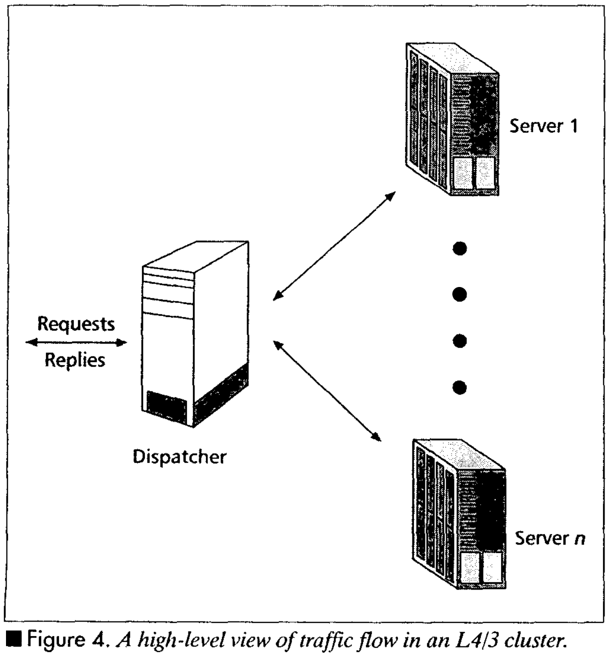
■ Figure 3. *Traffic flow in an L4/2 cluster.*

The eNetwork Dispatcher supports fault detection and masking for both the dispatcher and the servers in the pool. Server fault tolerance is achieved through IBM's High Availability Cluster Multi-Processing for AIX (HACMP) on an SP-2 server. Additionally, the dispatcher may have a hot spare that functions as a backup dispatcher. The primary dispatcher runs a cache consistency protocol with the backup. In the event that the backup no longer receives heartbeat messages from the primary, it takes over.

Through a mechanism IBM calls *client affinity*, the eNetwork Dispatcher is able to support services such as FTP and SSL. With client affinity, multiple connections from the same client within a given period are directed to the same server. This allows servers and clients to share state information such as SSL session keys during the timeout period.

LSMAC

LSMAC, from the University of Nebraska-Lincoln, implements L4/2 clustering as a portable user-space application running on commodity systems [6]. Utilizing `libpcap` [7] and `libnet` [8], LSMAC achieves performance comparable to the eNetwork Dispatcher. Experimental results demonstrate that three server nodes plus LSMAC, all running on Pentium II-266s in a switched Fast Ethernet environment, achieve about 1800 connections/s [6].



■ Figure 4. *A high-level view of traffic flow in an L4/3 cluster.*

Like other dispatchers, LSMAC provides fault detection and masking for the server pool. Periodically, the dispatcher sends ARP queries to determine which servers are currently active, thus allowing for automatic detection of dynamically added or removed systems. In addition, it watches for TCP reset messages corresponding to the service being clustered and removes the nonperforming system from the pool.

Alteon ACEdirector

ACEdirector from Alteon is another L4/2 clustering product. ACEdirector is implemented as an Ethernet switch (both layers two and three) based on a 2.5 Gb/s switch fabric. However, it has the added ability to operate as an L4/2 cluster dispatcher. While Alteon was the first to offer in-switch clustering, others have followed suit (e.g., Arrow Point, Cabletron, and Intel, which also support L7 dispatching, discussed later).

ACEdirector provides round-robin and least-connections load sharing policies, and allows for some stateful services such as SSL. Moreover, it provides fault detection and masking for the server pool and hot-standby with another ACEdirector switch. According to Alteon's product literature, the ACEdirector is capable of 25,000 connections/s at full "wire speed." This high connection rate is due to their extensive use of specialized application-specific integrated circuits (ASICs) that do most of the session processing.

L4/3 Clustering

L4/3 clustering technologies slightly predate L4/2 methods. L4/3 cluster-based servers provide reasonable performance while simultaneously providing the flexibility service providers expect by leveraging commodity products. Unlike L4/2 clusters, each constituent server is configured with a unique IP address in L4/3 clusters. The IP address may be globally unique or merely locally unique.

An L4/3 dispatcher appears as a single host to a client. To the machines in the server pool, however, an L4/3 dispatcher appears as a gateway. When traffic is sent from the client to the clustered Web server, it is addressed to the cluster address. Utilizing normal network routing rules, this traffic is delivered to the cluster dispatcher.

If a packet received corresponds to a TCP/IP connection initiation, the dispatcher selects one of the servers in the server pool to service the request (Fig. 4). Similar to that in L4/2 clustering, server selection is based on some load sharing algorithm, which may be as simple as round-robin. The dispatcher also then makes an entry in a connection map, noting the origin of the connection, the chosen server, and other information (e.g., time) that may be relevant. However, unlike in the earlier approach, the destination (IP) address of the packet is then rewritten as the address of the server selected to service this request. Moreover, any integrity codes affected — such as packet checksums, cyclic redundancy checks (CRCs), or error correction checks (ECCs) — are recomputed. The modified packet is then sent to the server corresponding to the new destination address of the packet.

If incoming client traffic is not a connection initiation, the dispatcher examines its connection map to determine if it belongs to a currently established connection. If it does, the dispatcher rewrites the destination address as the server previously selected, recomputes the checksums, and forwards as before. In the event that the packet does not correspond to an established connection but is not a connection initiation packet itself, the packet is dropped. Of course, as with L4/2 dispatching, approaches may vary slightly.

Traffic sent from the servers in the server pool to clients must also travel through the dispatcher since the source address on the response packets is the address of the particular server that serviced the request, not the cluster address. The dispatcher rewrites the source address to the cluster address, recomputes the integrity codes, and forwards the packet to the client.

The traffic flow in an L4/3 clustered environment is illustrated in Fig. 5 and summarized as follows:

- A client sends an HTTP packet with A as the destination IP address.
- The immediate router sends the packet to the dispatcher on A, since the dispatcher machine is assigned the IP address A.
- Based on the load sharing algorithm and session table, the 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.
- Server 2 accepts the packet and replies to the client via the dispatcher, which the back-end server sees as a gateway.
- The 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.

The basic L4/3 clustering approach is detailed in RFC 2391, "Load Sharing Using Network Address Translation (LSNAT)" [9]. Magicrouter from Berkeley was an early implementation of this concept based on kernel modifications [10]. Cisco's LocalDirector product is a proprietary commercial implementation, while LSNAT from the University of Nebraska-Lincoln provides an example of a nonkernel space implementation [6].

In hindsight (recall that L4/3 clustering predates L4/2 clustering), it is obvious that L4/2 clustering will always outperform L4/3 clustering due to the overhead imposed by L4/3 clustering (the necessary integrity code recalculation coupled with the fact that all traffic must flow through the dispatcher). Even if hardware support is provided for integrity code recalculation (as with Gigabit Ethernet), an L4/3 dispatcher must process much more traffic than an L4/2 dispatcher. Thus, total data throughput of the dispatcher limits the scalability of the system more than the sustainable request rate.

We describe the above-mentioned implementations of L4/3 clustering in detail below.

Magicrouter

Magicrouter, developed at the University of California at Berkeley, provided an early implementation of L4/3 clustering [10]. Using a kernel modification called "fast packet interposing," Magicrouter provided load sharing and fault tolerance. Magicrouter offered three load sharing algorithms: round-robin, random, and incremental load. As the names suggest, round-robin and random used round-robin and random connection dispatching policies. Incremental load used a per-server load estimate plus an additional adjustment based on the number of connections active at the server in question. During connection initiation, Magicrouter selects the least loaded server.

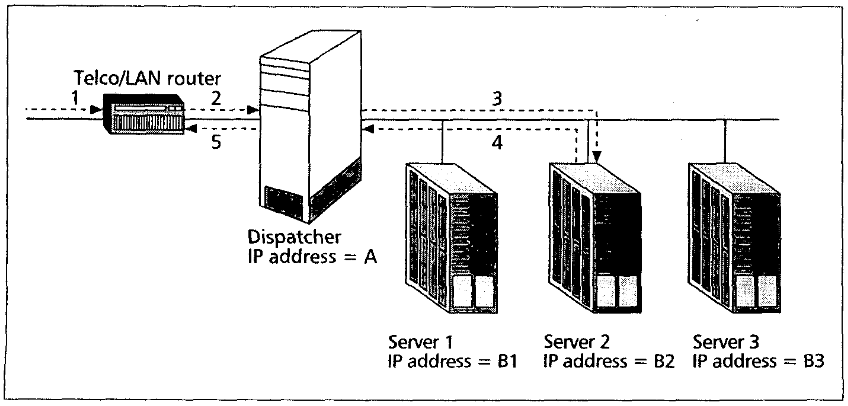
To provide fault detection, Magicrouter utilizes ARP as well as TCP reset detection. Periodically, ARP requests are sent out to map server IP addresses to MAC addresses. In the event that a server does not respond, it is declared dead. Additionally, if any server responds to a packet with a TCP reset message, it is declared dead. For fault tolerance on the part of the dispatcher, Magicrouter employs a primary/backup model. A primary Magicrouter replicates state information to one or more backup units over the network. In the event that a backup Magicrouter does not receive a heartbeat message from the primary for three time units, it declares itself the primary. Numerical order on Ethernet addresses is utilized to resolve conflicts in the event that two Magicrouters declare themselves as the primary unit.

LocalDirector

The LocalDirector product from Cisco Systems was an early commercial implementation of L4/3 clustering. According to Cisco documentation, LocalDirector provides over 45 Mb/s throughput and supports a combined total of 8000 cluster addresses and actual servers. LocalDirector provides the ability to support up to 1 million simultaneous connections, and offers the following load sharing policies:

- Least Connections: This policy directs incoming connects to the server with the fewest currently established connections.
- Weighted Percentage: This policy is similar to the least connection policy but with the addition that weights may be assigned to each of the servers in the server pool. This allows the user to manually tune the dispatching policy to take into account varying server capacities.
- Fastest Response: This policy attempts to dispatch the connection to the server that responds to the connection request first.
- Round-Robin: This is a strictly round-robin policy.

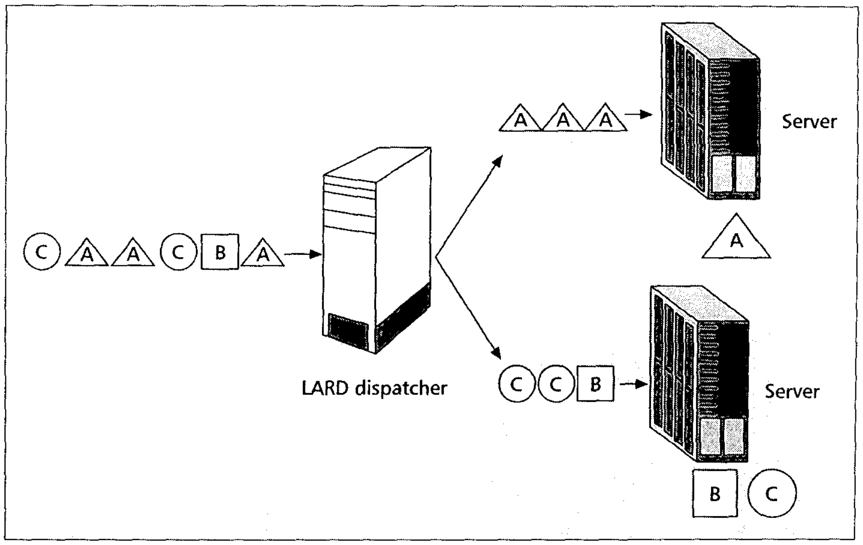
An additional LocalDirector unit provides hot standby operation when the two are linked with a special



■ Figure 5. *Traffic flow in an L4/3 cluster.*

failover cable. The second unit is a dedicated spare and may not be used for other tasks. LocalDirector also provides failure detection and reconfiguration with regard to the server pool. In the event that a server stops responding to requests, LocalDirector removes it from its list of active servers and marks it as being in a testing phase. It then periodically attempts to contact the server. As soon as it is capable of contacting the server, it is brought back into active duty.

Through the use of its *sticky* flag, LocalDirector can be made to support some stateful services, such as SSL [11] (IBM's client affinity). When the sticky flag is set, multiple connections from the same client within a given period — five minutes by default — are directed to the same server. This allows servers and clients to share state information, such as SSL session keys, during the timeout period.



■ Figure 6. *An overview of LARD.*

LSNAT

LSNAT [6] from the University of Nebraska-Lincoln is a user-space implementation of the key points of RFC 2391, "Load Sharing Using Network Address Translation" [9]. LSNAT runs on standard hardware under the Linux operating system or any other modern UNIX system supporting `libpcap` [7] and POSIX threads. Operating entirely in user space, LSNAT achieves a throughput of 30 Mb/s [6]. While this is generally lower than LocalDirector, it may have more to do with poor packet capture performance on the test platform rather than the choice of a user-space or kernel-space implementation [12].

LSNAT also provides failure detection and reconfigures itself accordingly. In the event of dispatcher failure, unlike LocalDirector, LSNAT does not fail over to a dedicated hot spare. Rather, one of the servers in the server pool detects its failure and reconfigures itself as the dispatcher. It then uses a distributed state reconstruction mechanism to rebuild the map of existing connections. If one of the servers fails, LSNAT detects this and removes it from its list of active servers. Upon restarting, the server announces its presence and is placed back in the active server pool. This functionality is achieved with the aid of a small daemon.

L7 Clustering

While strictly L4/2 or L4/3 clustering may be considered solved problems, a great deal of research is currently ongoing in the area of L7 clustering. These approaches use information contained in OSI layer seven (application layer), typically to augment L4/2 or L4/3 dispatching. This is also known as *content-based dispatching* since it operates based on the contents of the client request. We examine LARD from Rice University [13], a Web Accelerator from IBM T. J. Watson Research Center [14], and a commercial hardware product from ArrowPoint Communications.

LARD

Researchers at Rice University have developed a Locality-Aware Request Distribution (LARD) dispatcher for a pool of Web servers. Since servers are selected based on the content of the protocol request, we classify LARD as an L7 dispatcher. LARD partitions a Web document tree into disjoint subtrees. Each server in the pool is then allocated one of these subtrees to serve. In this way, LARD provides content-based dispatching as requests are received. Figure 6 presents an overview of this processing. The first server is capable of handling requests of type △A; the second can handle requests of types 🄱 and Ⓒ. We see the dispatcher decomposing the stream of requests into a stream of requests for the first server and one for the second server, based on the content of the requests (i.e., type △A, 🄱, or Ⓒ).

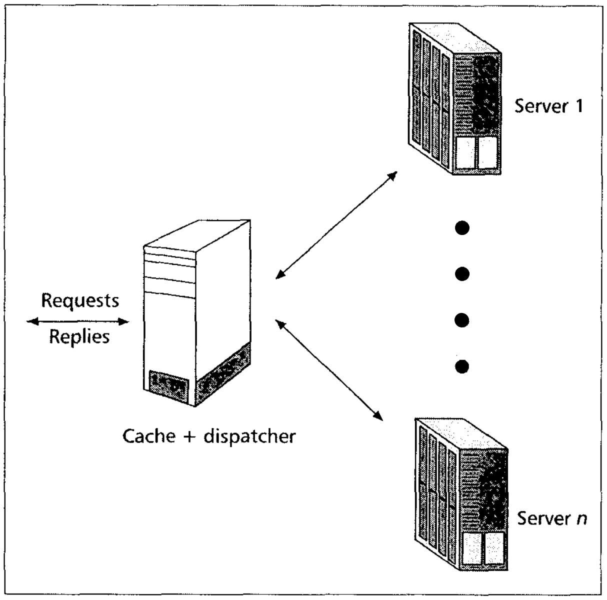
As requests arrive from clients for the clustered Web server, the LARD dispatcher accepts the connection as well as the request itself. The dispatcher then classifies the requested document and dispatches the request to the appropriate server. The dispatching is done with the aid of a modified kernel that supports a connection handoff protocol: after the connection has been established, the request known, and the server chosen, the LARD dispatcher informs the chosen back-end server of the status of the network connection, and the backend server takes over that connection (communicating directly with the client). In this way, LARD allows each server's file system cache to cache a separate part of the Web tree rather than having to cache the entire tree, as "ordinary" L4/2 and L4/3 clustering require. Additionally, it is possible to have specialized server nodes. For example, dynamically generated content could be offloaded to special compute servers while other requests are dispatched to servers with less processing power. While LARD requires a noncommodity operating system on the servers (they must be able to support the TCP handoff protocol), it does allow service providers to choose from commodity Web servers.

In experiments, LARD has achieved 2200 connections/s with an aggregate throughput of 280 Mb/s and a utilization of 60 percent on the dispatcher [13]. This suggests that with enough servers in the pool, the dispatcher would be capable of handling nearly 4000 requests/s on a Pentium II-300.

IBM's Web Accelerator

IBM's Web Accelerator, developed at T. J. Watson Research Center, combines content-based dispatching, based on *layer seven and four switching with layer two packet forwarding* (L7/2), with Web page caching [14]. However, page caching comes at the cost of reduced parallelism in the cluster.

The Web Accelerator runs on the same node as the IBM



■ Figure 7. *An overview of IBM's Web Accelerator.*

eNetwork Dispatcher. When a client attempts to connect to the clustered Web server, the Accelerator accepts the connection and the client request. If possible, this request will be served out of an in-memory cache on the dispatcher. In the event that there is a cache miss, the dispatcher contacts a server node and performs the same request as the client. It then caches this response and issues the response back to the client.

The Accelerator can serve 5000 pages/s on a PowerPC 604-200, but this performance metric decreases rapidly as response size increases [14]. For example, with a 10-kbyte response, the Accelerator is capable of serving 3200 requests/s. With a 100-kbyte response, the requests per second handled drops to about 500. This is due to the fact that unlike LARD, all outgoing traffic is issued from the Accelerator. Thus, service providers cannot fully exploit the latent parallelism in the cluster since all responses must now travel through the dispatcher, as shown in Fig. 7. Note that the traffic flow through the system looks similar to L4/3 clustering, rather than L4/2 clustering (as is the case for the IBM eNetwork Dispatcher executing without the Accelerator).

ArrowPoint

Web Switches In one of the first hardware devices to incorporate content-based routing, ArrowPoint's Web switches employ a caching mechanism similar to IBM's Web Accelerator. ArrowPoint's Web switches also provide *sticky* connections in order to support some stateful services.

ArrowPoint's CS-800 specification sheet claims a maximum connection rate of 20,000 connections/s (HTTP) with a maximum throughput of 20 Gb/s. Moreover, the CS-800 switch supports a hot standby unit and fault masking on the server nodes.

Cabletron, Intel, and others provide similar products.

	L4/2	L4/3	L7
Mechanism	Link-layer address translation	Network address translation	Content-based routing
Flows	Incoming only	Incoming/outgoing	Varies
Fault tolerance	Yes	Yes	Yes
Restrictions	Physical interface on every network with server; incoming traffic passes through dispatcher	Dispatcher lies between client and server	All incoming traffic passes through dispatcher; all outgoing as well, if caching
Performance	Thousands of c/s; hundreds of Mb/s	Hundreds of chips/s; tens of Mb/s	Tens of thousands of chips/s; Gb/s
Bottleneck	Connection dispatching	Integrity code calculations	Connections dispatching, dispatcher complexity
Advantage	Simple	Flexible	Server specialization, caching

■ Table 2. *A summary of transparent clustering techniques.*

Conclusion

Web server clustering has received much attention in recent years from both industry and academia. In addition to traditional custom-built solutions to clustering, transparent server clustering technologies have emerged that allow the use of commodity systems in server roles. We broadly classified transparent server clustering into three categories: L4/2, L4/3, and L7. Table 2 summarizes these technologies as well as their advantages and disadvantages.

Each approach discussed has bottlenecks that limit scalability. For L4/2 dispatchers, system performance is constrained by the ability of the dispatcher to set up, look up, and tear down entries. Thus, the most telling performance metric is the sustainable request rate. L4/3 dispatchers are more immediately limited by their ability to rewrite and recalculate the checksums for the massive numbers of packets they must process. Thus, in the absence of dedicated checksumming hardware, the most telling performance metric is the throughput of the dispatcher. Finally, L7 solutions are limited by the complexity of their content-based routing algorithm and the size of their cache (for those that support caching). However, by localizing the request space each server must service and caching the results, L7 dispatching should provide higher performance for a given number of back-end servers than L4/2 or L4/3 dispatching alone.

It seems clear that in the future, L7 hardware solutions such as the ArrowPoint switches will continue to dominate software products in terms of performance. The question one must ask is, how much performance is needed from the Web server for a given application and network configuration? As we have seen, even the L4/2 switch LSMAC — a software application running in user-space on COTS hardware and software — is capable of saturating an OC-3 (155 Mb/s) link. Apart from the Internet backbone itself, few sites have wide-area connectivity at or above this level. In boosting server performance to the levels supported by L7 hardware solutions (e.g., ArrowPoint switches), the bottleneck is no longer the ability of the server to generate data, but rather the ability of the network to get that data from the server to the client.

New research on scalable Web servers must take into account wide area network bandwidth as well as server performance. Industry and academic researchers have just begun to examine this problem. Cisco's DistributedDirector is an early example of a product that exploits geographic distribution of servers to achieve high aggregate bandwidth with low latency over the wide area in addition to a greater degree of fault tolerance.

References

[1] A. Fox *et al.*, "Cluster-Based ScalableNetwork Services," *Proc. 16th ACM Symp. Op. Sys. Principles*, Oct. 1997.

[2] L. Harbaugh, "A Delicate Balance," *Info. Week Online*, Jan. 25, 1999.

[3] D. Dias *et al.*, "A Scalable and Highly Available Server," CAPCON 1996, 1996.

[4] O. Damani *et al.*, "Techniques for Hosting a Service on a Cluster of Machines," *Proc. 6th Int'l. WWW Conf.*, Apr. 1997.

[5] G. Hunt *et al.*, "Network Dispatcher: A Connection Router for Scalable Internet Services," *Comp. Networks and ISDN Sys.*, Sept. 1999.

[6] X. Gan *et al.*, "LSMAC vs. LSNAT: Scalable Cluster-based Web Servers," to appear in *Cluster Comp.: J. Networks, Software Tools and Apps.*, 2000.

[7] Lawrence Berkeley Lab. Packet Capture Library, ftp://ftp.ee.lbl.gov/libcap.tar.Z

[8] DAEMON9, Libnet: Network Routing Library, Aug. 1999; http://www.packetfactory.net/libnet

[9] P. Srisuresh and D. Gan, "Load Sharing Using Network Address Translation," *The Internet Society*, Aug. 1998.

[10] E. Anderson, D. Patterson, and E. Brewer, "The Magicrouter, an Application of Fast Packet Interposing," Submitted for publication in the *2nd Symp. Op. Sys. Design and Implementation*, May 17, 1996.

[11] A. Frier, P., Karlton, and P. Kocher, "The SSL 3.0 Protocol," Tech. rep., Netscape, Nov. 18, 1996.

[12] D. Song and M. Undy, "NFR Performance Testing," Feb. 1999, http://www.anzen.com/research/research_perform.html

[13] V. Pai *et al.*, "E. Locality-Aware Request Distribution in Cluster-based Network Servers," *Proc. ACM 8th Int'l. Conf. Architectural Support for Prog. Langs. and Op. Sys.*, Oct. 1998.

[14] E. Levy-Abegnoli *et al.*, "Design and Performance of a Web Server Accelerator," *IEEE INFOCOM '99*, 1999.

Biographies

TREVOR SCHROEDER (tschroed@cse.unl.edu) received his B.S. in computer science from Wayne State College in 1998. He is currently an 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 care of UNIX machines. His research interests include distributed systems and especially security in such environments. He is a member of the ACM (S).

STEVE GODDARD (goddard@cse.unl.edu) received a B.A. degree in computer science and mathematics from the University of Minnesota (1985). He received 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 & Engineering faculty at the University of Nebraska-Lincoln (UNL) in 1998. He is a 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, computer networks, and software engineering.

BYRAV RAMAMURTHY (byrav@cse.unl.edu) received his B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Madras in 1993. He received his M.S. and Ph.D. degrees in computer science from the University of California at 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 a founding co-director of the Advanced Networking and Distributed Experimental Systems (ANDES) Laboratory at UNL. He serves as a co-guest editor of an upcoming special issue of *IEEE Network*. His research areas include optical networks, distributed systems, and telecommunications.