# A Scalable and Highly Available Web Server

Daniel M. Dias William Kish* Rajat Mukherjee and Renu Tewari

IBM Research Division
T. J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598

{*dias, c1kish, rajatm, c1renu* }*@watson.ibm.com*

## Abstract

*We describe a prototype scalable and highly available web server, built on an IBM SP-2 system, and analyze its scalability. The system architecture consists of a set of logical front-end or network nodes and a set of back-end or data nodes connected by a switch, and a load balancing component. A combination of* TCP routing *and Domain Name Server (DNS) techniques are used to balance the load across the front-end nodes that run the Web (httpd) daemons. The scalability achieved is quantified and compared with that of the known DNS technique. The load on the back-end nodes is balanced by striping the data objects across the back-end nodes and disks. High availability is provided by detecting node or daemon failures and reconfiguring the system appropriately. The scalable and highly available web server is combined with parallel databases, and other back-end servers, to provide integrated scalable and highly available solutions.*

## 1 Introduction

With the explosion of traffic on the World Wide Web [3, 12], the load on popular servers on the web is increasing rapidly. In this paper we describe and analyze various methods for building a scalable web server on a cluster of computing nodes, in which the throughput of the cluster can be increased by growing the size of the clustered system. We analyze the limits to scalability and the efficiency of these methods, and the trade-offs between the alternatives. As both the traffic and the commercial applications on the popular web servers increases, high availability of these servers becomes increasingly important. Given a clustered web server, we describe how the server can be made highly available, by taking over the load of a failed node by the remaining nodes in the cluster. Based on these methods, we describe a prototype of a scalable and highly available web server that we have built on an IBM Scalable Parallel SP-2 system.

*Consultant from Brown Cow Engineering Inc. He can be reached at kish@browncow.com

### 1.1 Relation to Previous Work

The NCSA prototype scalable HTTP server is described in [8, 10]. The method they use, illustrated in Figure 1, consists of having a set of HTTP servers on nodes in a cluster, that use the Andrew File System (AFS) [6] for sharing a set of HTML documents, and using the round-robin Domain Name Server (RR-DNS) [2, 13] for distributing accesses among the nodes in the cluster. Essentially, clients are presented a single name associated with the set of HTTP servers, and the RR-DNS maps this single name to the different IP address of the HTTP servers, in a round-robin manner; thus different clients will (ideally) be mapped to different server nodes in the cluster. In this way, the load is distributed among the servers, each of which access the same set of URLs through the distributed file system (AFS in this case).

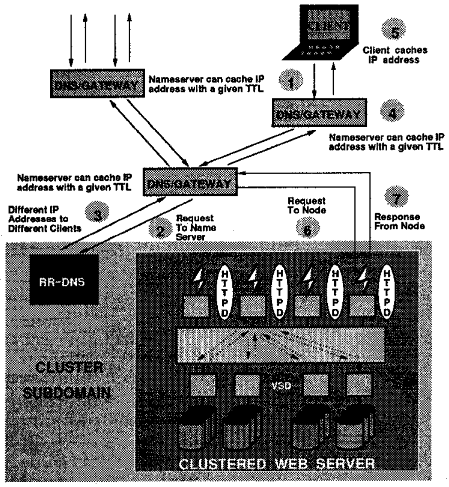
CLIENT
5
Client caches IP address
DNS/GATEWAY
Nameserver can cache IP address with a given TTL
1
DNS/GATEWAY
4
Nameserver can cache IP address with a given TTL
Nameserver can cache IP address with a given TTL
DNS/GATEWAY
Different IP Addresses to Different Clients
3
2
Request To Name Server
Request To Node
6
7
Response From Node
RR-DNS
HTTPD
VSD
CLUSTER SUBDOMAIN
CLUSTERED WEB SERVER

Figure 1: Domain Name Server: Flow

There are several problems that arise with this method. First, as illustrated in Figure 1, there are typically several name servers between clients and the RR-DNS that cache the resolved name-to-IP address mapping. In order to force a mapping to different server IP addresses, the RR-DNS can specify a *time-to-live* (TTL) for a resolved name, such that requests made after the specified TTL are not resolved in the local nameserver, but are forwarded to the authoritative RR-DNS to be re-mapped to the IP address of a different HTTP server. Multiple name requests made during the TTL period will be mapped to the same HTTP server. Thus, bursts of requests from new clients can appear at the same HTTP server, leading to significant load imbalance. If the TTL is made very small, there is a significant increase in network traffic for name resolution. Therefore, name servers often impose their own minimum TTL, and ignore very small TTLs (e.g., 0) given by the RR-DNS. A second problem is that clients cache the resolved name-to-IP address mapping (henceforth referred to as address caching). Since the clients may make future requests at any time, the load on the HTTP servers cannot be controlled subsequently and will vary due to statistical variations in client access patterns. Further, clients make requests in bursts as each web page typically involves fetching several objects including text and images, and this burst is directed to a single server node, increasing the skew. We show that these effects can lead to significant dynamic load imbalance, requiring that the cluster be operated at lower mean loads in order to be able to handle peak loads. Another issue relates to load balancing is the distributed file server used by the HTTP servers. Typically, there is a skew in the file accesses, with some URLs being more popular than others, leading to a skew in the file server nodes. This becomes more significant when large documents are accessed and file system caching does not suffice.

The remainder of the paper is organized as follows: In Section 2 we present a scheme that combines the RR-DNS with a set of so called *TCP routers* [14] to better balance the load among the HTTP servers. This combination also allows for better scalability. We also describe a method that stripes the files across (data server) nodes in the cluster [5, 4, 16] thereby balancing the data access load. In Section 3, we quantitatively examine load balancing using the RR-DNS approach, and quantify the performance and scalability of the web server that we have prototyped. In Section 4 we describe how high availability is achieved in the prototype. Finally, concluding remarks appear in Section 5.

## 2 Scalable Web Server Architectures

The scalable web server architecture is illustrated in Figure 2, and consists of multiple web servers running on a clustered architecture. A clustered architecture consists of a group of nodes connected by a fast interconnection network, such as a switch. Each node in the cluster has a local disk array attached to it. The disks of a node can either maintain a local copy of the web documents or a share it among the nodes. For better performance and load balancing, the shared web documents are striped across multiple nodes. As shown in Figure 2, the nodes of a cluster are divided into two logical categories: (i) front-end

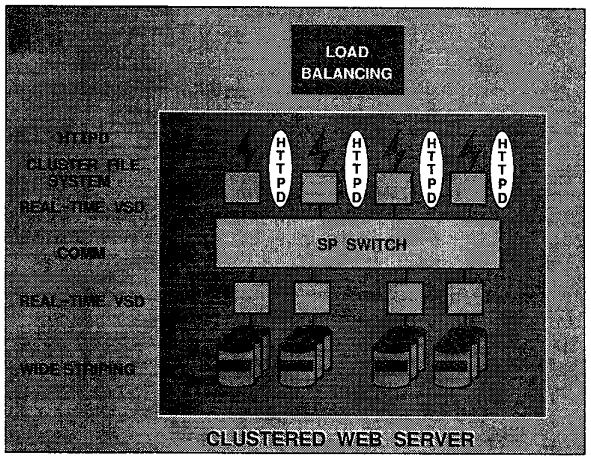

Figure 2: Scalable Web Server Architecture

(delivery) nodes and (ii) back-end (storage) nodes. A load balancing component is used to distribute incoming requests from the external network to the front-end nodes, which also run httpd daemons. The (logical) front-end node then forwards the required commands to the back-end nodes that have the data (document), using a shared file-system. We assume an underlying software layer (e.g., virtual shared disk) which makes the interconnection architecture transparent to the nodes [1]. The results are then sent back from the back-end node to the front-end node through the switch and transmitted to the user.

The clustered architecture outline above can be applied to several applications, and is similar to the design of a scalable video server. Essentially, the same architecture applies to a video server as well, with the httpd daemon replaced by a daemon that reads video objects, and plays them out to clients across the network at the required rate. For both web and video servers, two possible configurations can be used: (i) two-tier and (ii) flat. In the two-tier architecture of Figure 3, the logical front-end and back-end nodes are mapped to different physical nodes of the cluster and are distinct. In a flat architecture, on the other hand, each physical node can serve as both the logical front-end and back-end [16]. All nodes are identical, performing both storage and delivery functions. For a clustered web server we will principally consider the two tier architecture only.

### 2.1 Load Balancing

A key requirement in order to achieve scalability of the web server is that of balancing the load across the multiple front-end nodes and the back-end nodes.

#### 2.1.1 Front-End

The front-end nodes run the web daemons and are connected to the external network. To balance the load among them, the client requests need to be spread evenly across nodes. Several methods can be used to

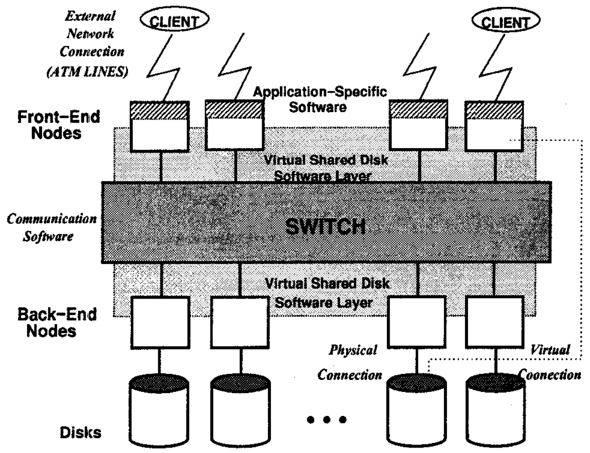
External Network Connection (ATM LINES)
CLIENT
CLIENT
Application-Specific Software
Front-End Nodes
Virtual Shared Disk Software Layer
Communication Software
SWITCH
Virtual Shared Disk Software Layer
Back-End Nodes
Physical Connection
Virtual Connection
Disks

Figure 3: **Two-tier Server Architectures**

achieve this, with varying degrees of effectiveness. In Section 1, we outlined the round-robin domain name server (RR-DNS) based solution similar to the NCSA approach. To summarize, in this scheme all the front-end nodes are given a single logical name, and the RR-DNS maps the name to multiple IP addresses. However, due to address caching in the client, the bursty nature of client accesses, and TTL values imposed by nameservers, the load balancing achieved is coarse, as quantified in Section 3.

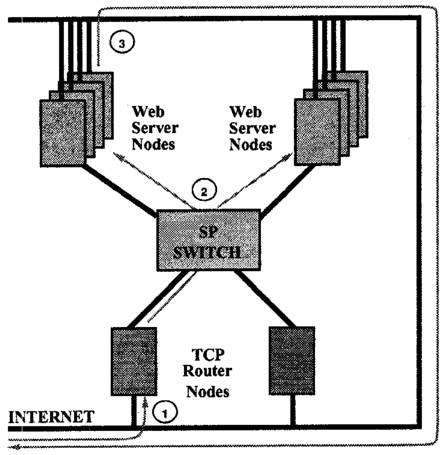

Figure 4: Scalable Web Server (Router): Flow

Another method for achieving load balancing is based on routing, and is illustrated in Figure 4. One or more nodes of the cluster serve as *TCP router(s)*, forwarding client requests to the different front-end nodes in the cluster in a round-robin order. The name and IP address of the router is public, while the addresses of the other nodes in the cluster is private. (If there is more than one router node, a single name is used and RR-DNS is used to map the name to the multiple routers.) This has been referred to as the *encapsulated cluster* in [14]. The client sends requests to the router node which in turn forwards all packets belonging to a particular TCP connection to one of the server front-end nodes. The router can use different algorithms based on load to select which node to route to, or use a simple round-robin scheme. In the prototype (see below), the router resides just above the IP layer as a kernel extension. The server nodes directly send the request back to the client without using the router. However, the server nodes change the source address on the packets sent back to the client to be that of the router node. This makes the entire routing action transparent to the clients. Note that, the response packets are large compared to the request packets, especially for images and video clips, and these bypass the router. Thus, the overhead added by the router is small (and is quantified later).

One advantage of the router scheme over the DNS based solutions is that fine grained load balancing can be achieved and there is no problem of client or nameserver caching. One potential issue with both the RR-DNS and routing approaches is that the router node could become a bottleneck, as could the domain name server. These performance aspects are quantified in Section 3.

To further improve scalability, the router and DNS schemes can be combined in various ways. First, a number of router nodes can be used, as shown in Figure 4, and the RR-DNS method can be used to map different clients to different router nodes. This hybrid scheme can tolerate the coarse grained load balancing achieved using RR-DNS because the corresponding router will route any burst of requests, that were mapped by the RR-DNS to the same router, to different front-end server nodes. It achieves good scalability because (i) a long TTL can be used so that the node running the RR-DNS does not become a bottleneck, and (ii) several router nodes can be used achieving scaling beyond that of a single router.

Another hybrid router-DNS scheme combines the logical router and front-end nodes: The DNS based solution is used to get the coarse grained load balancing. Each front-end server node then acts as a router; if the load on the server is low, then the request is handled locally at that server, otherwise the router forwards the request to another node based on its load or a global load balancing algorithm. This scheme eliminates the forwarding by the router unless the load on the front-end node requires it.

### 2.1.2 Back-End

The back-end nodes function as the server nodes for the shared file-system (e.g., AFS, NFS etc.) that is used by the front-ends to access the data. For instance, the NCSA prototype [8] uses AFS for the data server. With traditional shared file systems, the total document set is partitioned among the different back-end servers. The partitioned approach could lead to load imbalances based on the access skew among the documents. Back-end servers that store the *hot* documents will get overloaded, resulting in hot spots. The file-system caching at the front-end could alleviate this problem for small documents that fit in the local cache. Only the initial requests need to go to the back-end server and the remaining can be serviced from the local front-end cache. For larger documents, however, the back-end server nodes will be unevenly hit.

To balance the load among the back-end nodes different methods ranging from replication to striping can be used. If the data is replicated across each back-end nodes, the logical front-end and back-end can be mapped to same physical node making it a flat architecture. The scheme described for front-end load balancing can be used directly. Full replication of data is expensive in terms of space utilization. Also, if the data is modified frequently, some form of consistency control is required.

Instead of replication, each document is divided into logical blocks, which are then distributed among the disks in the system. The logical block represents the size of the striping unit, and could be mapped to multiple packets within the switch communication layer and to multiple physical blocks on disk (we use the term block, to refer to a logical block). The blocks belonging to a document may either span all the disks in the system (referred to as *wide striping*) or may be confined to a smaller subset of disks (referred to as *short striping*). The block sizes are typically large ($>$ 64KB) to minimize the access latencies. Successive blocks of an object are allocated to consecutive disks using a sequential placement. The shared file system has additional support to access the striped documents.

Wide striping implicitly achieves higher disk-arm bandwidth and load balancing [5, 16]. However, small documents on the server cannot be striped across all the disks in the system. Such documents are striped only on a subset of the disks in the system and replicated on the remaining disks sets. Unlike short striping, however, in a server that employs wide striping, a single disk or node failure affects all the streams being accessed. Back-end node failures can be masked using the availability schemes described in Section 4.

## 3 Performance and Scalability

In this section, we study the performance issues relating to the scalability of cluster-based solutions using the RR-DNS and router. The performance of the RR-DNS was measured independently from that of the web server. Web server performance was derived from the Webstone benchmark [18], and the RR-DNS performance was measured via a stand-alone RR-DNS benchmark. Studies of the load imbalances using the RR-DNS and router strategies were performed via CSIM [15] simulations of a networked system.

### 3.1 Load Imbalance Due to Client and Network Effects

We have qualitatively discussed the factors that affect load balancing across nodes in Section 2. To quantitatively study the effect of dynamic load imbalance we simulate a networked system and vary the number of web page requests per client session, TTL at the nameservers and the number of nodes in the server. The simulation uses a burst model and a single level of caching nameservers between the clients and the server. For each page requested by a client a burst of small requests, representing the objects within a page, are sent to the server. Each client resolves the name of the server only once during a session. The number of remote nameservers is typically much larger than the number of nodes in the server. We illustrate the impact of the TTL with a single level of caching nameservers. The simulation parameters are shown in Table 1.

| Parameter | Default |
|---|---|
| Number of Nodes | 16 |
| Number of Nameservers | 150 |
| Levels of Caching Nameservers | 1 |
| Clients per node | 80 |
| Server node capacity (requests are of the same size) | 50 req./sec |
| Mean interarrival time (between page requests per client) | 15 secs. |
| Mean number of bursts per page | 10 |
| Mean # of Page Requests/client session | 20 |
| TTL | 300 secs. |

Table 1: **Simulation Parameters**

Figure 5 shows the effect of varying the number of requests per client session on the dynamic load imbalance at the server for different TTL values at the nameservers. Each request to the server is assumed to be of the same size so that the load imbalance observed is not due to varying request size. The dynamic load imbalance is measured as a ratio of the maximum number (99% percentile) of requests serviced by a node in any sampling interval to the mean computed over the total simulation time period. (Note that, both the RR-DNS and router schemes lead to good average load balance across the nodes over an extended period of time. We focus on the dynamic load balance which can lead to very different peak loads on nodes in the cluster using the two schemes.) The inter-arrival times between page requests per client session is derived from an exponential distribution with a mean of 15 seconds. For a TTL value of 0, as the mean number of page requests per client session increases from 1 to 20, the effect of client address caching becomes more pronounced, resulting in a load imbalance increase from 8% to about 30%. Note, each page requests consist of a burst of requests (mean of 10) to the server. With an increase in the TTL value from 0 to 300 seconds, without client address caching (number of page request per client is one), the load imbalance increases from 8% to about 33%, sharply at first and then gradual. The effect of increasing TTL at the nameservers or that of increasing the duration of a client session is adverse on the load imbalance at the server nodes. The router approach results in a much lower load imbalance of about 8% caused by the bursty nature of client requests and random arrivals.

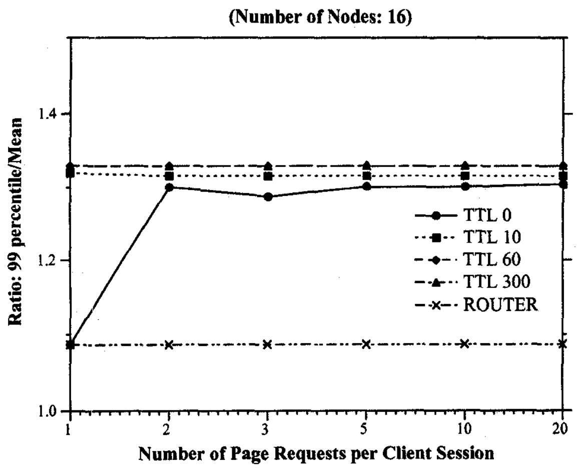

Figure 5: Load Imbalance of RR-DNS

The load imbalance is also a function of the number of nodes in the web server. As the number of nodes increases the load imbalance using the router decreases. Figure 6 shows the effect of varying the number of nodes on the load imbalance using the RR-DNS and router approaches. When the number of nodes is small (e.g., 2) the load imbalance due to the bursty nature of the requests and client random arrivals is high for both the router (22%) and the RR-DNS (30%). As the number of nodes increases, the effect diminishes for the router as it forwards the requests to different nodes. For the RR-DNS approach however, the load imbalance due to client and nameserver address caching contribute largely to the load imbalance and it gradually increases (31%) as the nodes are increased. Thus, for a large scale system a router based approach is recommended as the load imbalance is significantly smaller for the router than the DNS approach. For smaller systems both approaches are viable. Also in the simulation study each client randomly selects a nameserver, whereas in practice some nameservers may be much more utilized than others, causing larger imbalances using the RR-DNS approach.

## 3.2 Scalability of RR-DNS and Router

Figure 7 shows the scalability achievable via the RR-DNS solutions for different workloads (file sizes) for different ratios of new accesses to total accesses made to the web server. The RR-DNS is accessed only upon requests from new clients, since the address of the web server node is cached at the client after the first access. We varied the percentage of new accesses from 100% (every access is from a new client) to 1% (a client makes a hundred requests to the web site before retiring). Since every document typically consists of several constituent objects, the latter number reflects a few pages accessed per client.

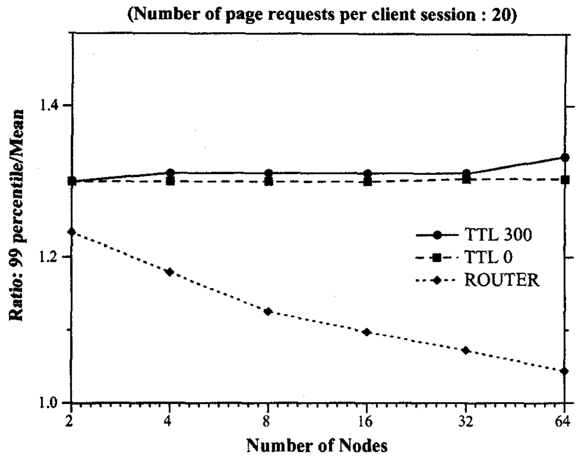

Figure 6: Effect of System Size on Load Imbalance

Since each node can handle fewer requests per second when the documents are large, we see that a single RR-DNS can handle the request to a larger number of server nodes when the average file size is larger. In absolute terms, the RR-DNS scales to a large number of nodes (100-200) even at smaller file sizes, assuming that 10-20% of the RR-DNS accesses are from new clients.

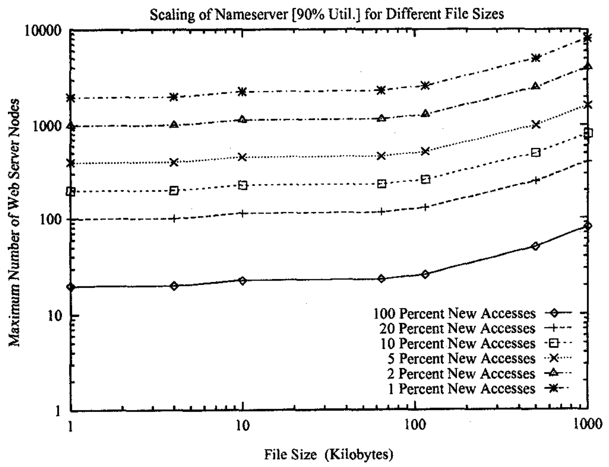

Figure 7: Scalability of RR-DNS

For the router, we measured the CPU utilization during the execution of the Webstone benchmark with four server nodes configured. For a given server node at 90-95% utilization, the CPU utilization of the router node is about 2.2% when the file-size is small (1 KByte) and 1.2% for large files (1 MByte). Thus, based on the average file size of the web server workload, a single router node (operating at 90% utilization) can support between 40 and 75 nodes, more than sufficient for practical systems. Figure 8 shows the scaling of a single router node for documents with different file sizes. If larger configurations are desired, multiple router nodes can be used, with load balancing across the routers being achieved via RR-DNS. As shown in section 3.1, the router solution has better load balancing properties compared to the RR-DNS solution.

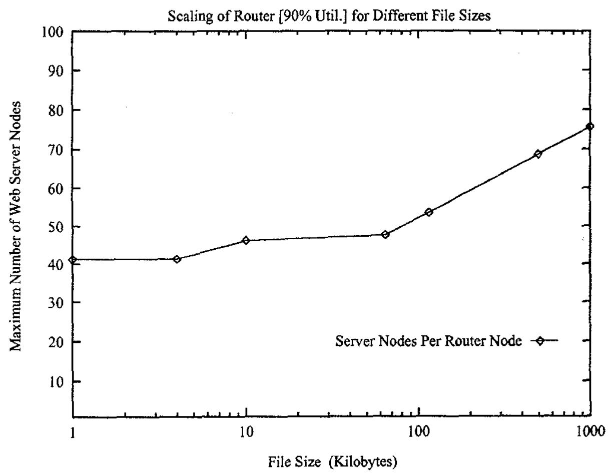

Figure 8: Scalability of a Single Router Node

## 4 High Availability

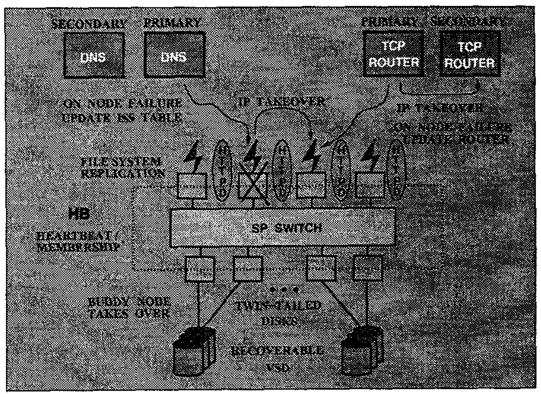

Figure 9: High Availability

As the traffic and commercial applications on popular web servers increase, providing highly available web servers becomes increasingly important. One of the advantages of a clustered system is the possibility of providing high availability by masking node and other failures. This is done by detecting failures and reconfiguring the system such that the workload is taken over by the remaining nodes in the cluster. For the scalable web server, in order to guarantee high availability, we need to mask both front-end and back-end node failure. The method we use for making the web server highly available is illustrated in Figure 9. Node failures are detected by using a heart-beat and membership protocol [7], that essentially eliminates nodes from the membership group in a consistent manner when (several) heartbeat messages from the node are not received. Network failures are detected by using heartbeats on multiple networks. Other resource failures are detected by monitors. On detection of node or other resource failures, a distributed recovery driver component [11] initiates recovery by coordinating a series of recovery commands or scripts across the remaining nodes in the system. The actions taken for specific failures is outlined below.

For front-end node failure, if the DNS or router solution is used for load balancing, the failed node is removed from either the name server tables or the router configuration table. Once these tables are updated to reflect the failure, all remote clients that try to make a connection after the failure will not be directed to the failed node. However, in the DNS approach some clients may have cached the address of the failed node and will continue to try connecting to it. Similarly, with the router based approach a router node may fail causing clients to see a failure. To handle these scenarios each front-end node and router node has a "buddy" which takes over its IP address after failure. Remote clients that have cached the address of the failed node will connect to the buddy transparently. For IP takeover the nodes have a redundant adapter. One of the adapters is used for servicing all the external IP requests and one for booting. The IP address of the main adapter is taken over by a spare adapter in the buddy node. The switching is done in software and is orchestrated by the recovery driver, as outlined earlier.

To handle the back-end server failure we use a hardware approach, namely twin-tailed disks. Other approaches like replication and software-RAID can be used for large scale servers with real-time requirements [17]. The disks attached to a node are twin-tailed to a buddy node. When a node fails the buddy node takes over the disk. The front-end nodes forward the requests to the buddy. Twin-tailing requires either special disks or extra SCSI adapters on the nodes that connect to the shareable disks. With twin-tailing the load on the buddy can double in the worst case. Multi-tailing can reduce the overload to some extent.

## 5 Conclusions and Future Work

Based on the analysis and design described above we have built a prototype scalable and highly available web server on an IBM SP2 system, illustrated in Figure 10. This system is essentially an instantiation of the architecture shown in Figure 2, and is a two-tier system with four front-end and four back-end nodes. The disks are connected to the back-end nodes. The disks are (logically) shared among the front-end nodes using the Virtual Shared Disk software [1]. The data for web files are striped across the back-end nodes and disks, thereby achieving load balancing across the back-end nodes based on work by [4]. The combination of the TCP router and RR-DNS described earlier is used for load balancing across HTTP daemons running at the front-end nodes. The TCP router is based on the encapsulated cluster prototype [14]. The scalability data presented in Section 3 is based on measurements on the prototype. The methods for achieving high availability outlined in the previous section have been implemented in the prototype.

The hardware and software infrastructure is shared between the scalable web server and a prototype scalable video server: the video objects are striped across the same nodes and disks, the HTTP daemons are replaced by video push daemons, and load balancing is achieved by a control server [9] that assigns the client streams to front-end nodes. We present forms with video selections on the web, and allow the web client to start up (out of band) play-out of video objects over separate high bandwidth channels. Currently, the constrained bandwidth to the client and also in the network prevent any significant real-time video access on the web. Over time, we believe that real-time video over the web will become feasible; the prototype's ability to support scalability for both web and video will then become increasingly important.

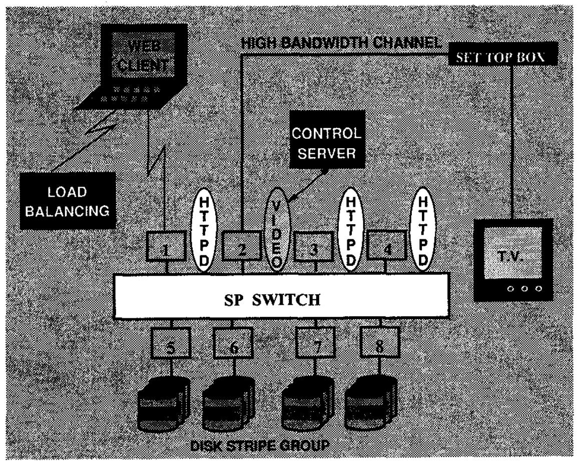

Figure 10: SP Video and Web Server Prototype

We quantified the scalability of this solution, and examined the performance of the (known) RR-DNS approach. We showed through simulations that the pure DNS approach can be significantly impacted by dynamic load imbalance among the nodes in the cluster. As a specific example, due to client caching of the mapping from the name to IP address, the peak load on nodes in the cluster can be 30% to 40% higher than the mean load on the nodes, because bursts of requests from clients lead to dynamic load imbalance. With higher values of the TTL at nameservers, the load dynamic load imbalance with RR-DNS gets even worse. This means that, using RR-DNS, the cluster nodes would need to be configured such that the mean load is less than 60% in order to be able to sustain the peak loads that occur. We showed that the router solution largely eliminates this problem, and provides excellent dynamic load balancing; this allows the cluster nodes to operate at much higher mean loads, and still sustain the peak loads.

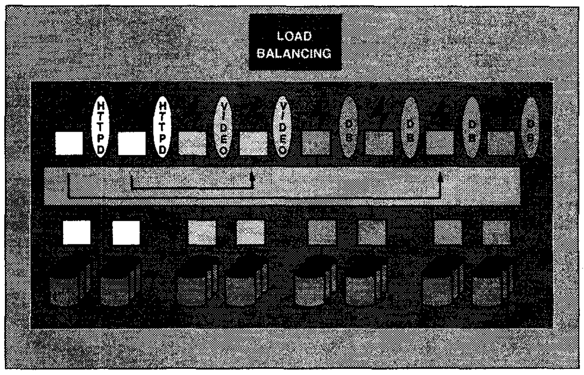

Figure 11: Integrated Scalable Solutions

In this paper we have focused on a scalable and highly available web server, with web pages stored on a shared file system. For many applications, access to back-end servers such as databases and on-line transaction processing via the web is required. The scalable web server described above can be integrated with parallel/scalable back-end servers, to create integrated scalable solutions, as illustrated in Figure 11. The figure illustrates the scalable web-server integrated in our prototype with a scalable video server and a parallel database, all in one management and high-availability domain.

**Acknowledgments:** The prototype draws on components developed by many people, especially C. R. Attanasio, S.C. Smith, C. Polyzois, R. Haskin, D. McNabb, F. Schmuck, J. Wyllie, M. Devarakonda and S. Fakhouri.

## References

[1] C.R. Attanasio, M. Butrico, C.A. Polyzois, S.E. Smith, and J.L. Peterson. Design and Implementation of a Recoverable Virtual Shared Disk. IBM Research Report RC 19843, T.J Watson Research Center, Yorktown Heights, New York, 1994.

[2] T. Brisco. DNS Support for Load Balancing. RFC 1794, Rutgers University, April 1995.

[3] Berners-Lee T. et al. The World-Wide Web. *Communications of the ACM*, 37(8):76 - 82, August 1994.

[4] Roger Haskin. Personal Communication, 1994.

[5] Roger Haskin and Frank L. Stein. A System for Delivery of Interactive Television Programming. In *COMPCON*. IEEE, March 1995.

[6] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and Performance in a

Distributed File System. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[7] F. Jahanian, S. Fakhouri, and R. Rajkumar. Processor Group Membership Protocols: Specification, Design and Implementation. In *Proceedings of the 12th Symposium on Reliable Distributed Systems*, pages 2 – 11, Princeton, NJ, October 1993. IEEE Computer Society.

[8] Eric Dean Katz, Michelle Butler, and Robert McGrath. A Scalable HTTP Server: The NCSA Prototype. *Computer Networks and ISDN Sytems*, 27:155 – 163, 1994.

[9] Martin Kienzle. Personal Communication, 1995.

[10] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. Ncsa's world wide web server: Design and performance. *IEEE Computer*, pages 68 – 74, November 1995.

[11] Avraham Leff, Richard P. King, and Daniel M. Dias. HAV: Infrastructure for Building Highly Available Clustered Systems. In Preparation, 1996.

[12] Robert E. McGrath. What We Do and Don't Know
About the Load on the NCSA WWW Server. URL < http://www.ncsa.uiuc.edu/ Information-Servers/Colloquia/28.Sep.94/ Begin.html>, September 1994.

[13] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, USC Information Sciences Institute, November 1987.

[14] Attanasio C. R. and Smith S.E. (a virtual multiprocessor implemented by an encapsulated cluster of loosely coupled computers ). *IBM Research Report RC18442*, 1992.

[15] Herb Schwetman. CSIM Reference Manual (Revision 16). MCC Technical Report ACT-ST-252-87, Microelectronics and Computer Technology Corporation, Austin, Texas 78759, May 1992.

[16] Renu Tewari, Daniel M. Dias, Rajat Mukherjee, and Harrick Vin. Real-Time Issues for Clustered Multimedia Servers. IBM Research Report RC 20020, T.J. Watson Research Center, April 1995.

[17] Renu Tewari, Rajat Mukherjee, Daniel Dias, and Harrick Vin. High Availablity in Clustered Multimedia Servers. In *International Conference on Data Engineering*, New Orleans, February 1996. IEEE.

[18] Gene Trent and Mark Sake. WebSTONE: The First Generation in HTTP Server Benchmarking. URL < http://www.sgi.com/Products/ WebFORCE/WebStone/paper.html >, February 1995.