Design and Performance Tradeoffs in Clustered Video Servers

Renu Tewari†* Rajat Mukherjee Daniel M. Dias Harrick M. Vin†

IBM Research Division
T. J. Watson Research Center
Hawthorne, NY 10532
{rajatm, dias}@watson.ibm.com

†Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
{tewari, vin}@cs.utexas.edu

Abstract

In this paper, we investigate the suitability of clustered architectures for designing scalable multimedia servers. Specifically, we evaluate the effects of: (i) architectural design of the cluster, (ii) the size of the unit of data interleaving, and (iii) read-ahead buffering and scheduling on the real-time performance guarantees provided by the server. To analyze the effects of these parameters, we develop an analytical model of clustered multimedia servers, and then validate it through extensive simulations. The results of our analysis have formed the basis of our prototype implementation based on a cluster of switch-connected RS/6000 machines. We briefly describe the prototype and discuss some implementation details.

1. Introduction

A wide range of applications in entertainment, business, and education require the development of multimedia servers that can support efficient means of storing and delivering digital audio and video objects to thousands of clients. The fundamental problem in developing such multimedia servers is that digital audio and video streams consist of a sequence of media quanta, such as video frames or audio samples, which convey meaning only when presented continuously in time (unlike text in which spatial continuity is sufficient). Hence, a multimedia server must ensure that the delivery/playback of media streams to and from disks proceed at their real-time rates.

Since current disk bandwidths are substantially larger than the data rate requirement of an isolated video stream (e.g., typical disk bandwidths are about 3-6 MB/sec, while an MPEG-2 compressed video rate is about 0.5 MB/sec), design of a single-user video server is relatively simple. However, the design of scalable multimedia servers that can service thousands of concurrent clients imposes several research challenges. The performance tradeoffs in the design of a scalable multimedia server based on a *processor-cluster* architecture is the subject matter of this paper.

1.1. Relation to Previous Work

In the recent past, several research projects have investigated techniques for designing small-scale multimedia servers, in which all the disks are assumed to be connected to a single node [2, 4, 5, 10, 11, 13, 16]. In such servers, however, the node becomes the bottleneck as it cannot sustain the bandwidth requirements of a large number of streams. Consequently, such servers cannot scale to support the number of streams required for high-end video-on-demand (VOD) type applications.

Recently, there have been some research efforts for designing scalable video servers. An ATM switch based storage server is described in [3]. Similarly, Freedman and DeWitt [6] have described the *SPIFFI* video server. Whereas their model of a cluster is appropriate for a set of terminals connected to a group of processors, it is unsuited for geographically separated clients that do not possess information about server internals. A detailed analysis of the issues involved in utilizing clustered architectures for video servers has not received much attention.

1.2. Contributions of this Paper

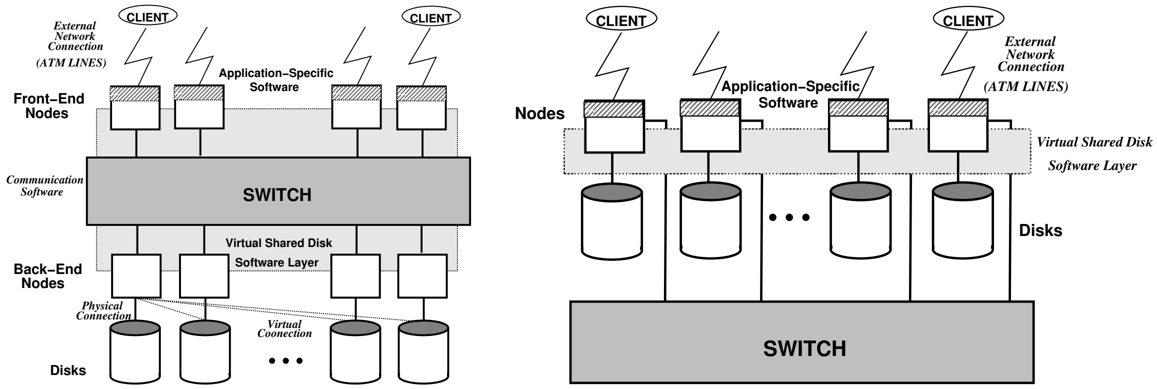
In this paper, we investigate the suitability of clustered architectures for designing scalable multimedia servers. Specifically, we evaluate the effects of: (i) architectural design of the cluster, (ii) the size of the unit of data interleaving, and (iii) read-ahead buffering and scheduling on the real-time performance guarantees provided by the server. To analyze the effects of these parameters, we develop an analytical model of clustered multimedia servers, and then validate it through extensive simulations. The results of our analysis have formed the basis of our prototype implementation based on a cluster of switch-connected RS/6000 machines.

The remainder of the paper is organized as follows: In Section 2, we describe two possible cluster architectures for video servers, discuss the placement strategies and other assumptions made about the server design. Section 3 describes an analytical model of the system, which is validated and evaluated by extensive simulations in Section 4. Section 5 describes our prototype implementation. Finally, Section 6 summarizes our results and outlines directions for future work.

2. Clustered Server Architectures

A clustered architecture consists of a group of nodes connected by a switch (interconnection network). Each node has a local disk array attached to it. The nodes of a cluster can be logically divided into two categories: (i) front-end (or delivery) nodes and (ii) back-end (or storage) nodes. Front-end nodes admit stream requests from the external

*This work was done when teh author was an intern at the IBM T.J. Watson Research Center.



(a) Two-Tier (b) Flat

Figure 1: Server Architectures

network (e.g., ATM) based on some pre-defined or dynamic admission control strategy. Back-end nodes store the data across their local disk arrays and provide it to the front-end nodes when requested. Read-ahead buffering per client stream is done at the front-end nodes which transmit the data at regular intervals to clients over the external network.

There are two possible configurations of such a clustered architecture: (i) two-tier and (ii) flat (see Figures 1(a) and 1(b), respectively). In a two-tier architecture, 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, all nodes are identical, performing both storage and delivery functions. The node that initiates the request for a stream is called the front-end node while the one that services the request from disk is the back-end node.

Regardless of the architecture, each video object is divided into logical blocks, which are then distributed among the disks in the system. Blocks of a video stream 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*). Wide striping implicitly achieves higher disk-arm bandwidth and load balancing [2, 9], but is more susceptible to failure.

Successive blocks of a video object may be allocated to disks either using a *round-robin* or *random* placement algorithm. As per the round-robin placement, successive blocks of a video stream are placed on consecutive disks. With random placement, successive blocks are placed on disks using a random permutation, with no two consecutive blocks on the same disk. In both schemes, the first block of a stream could be allocated on any disk with equal probability. Random placement policy adapts to incremental growth (adding disks and nodes) since redistribution is simple, but has more unpredictability in performance. A detailed comparison of these two policies can be found in [15].

For the purpose of the analysis presented in this paper, we will assume that the server employs wide striping and random placement policies. Moreover, we will assume that the server operates in *push* mode; i.e., it provides data to the client at the desired play-out rate after connection establishment.

3. Analytical Model

In this section we describe an analytical model of the system that can be used to compare the two architectures, predict an optimal block size, and show the effect of read-ahead buffering on the real-time performance. The analytical model can be used to quickly prune the space explored by simulation and lend credence to moderately long simulation results.

3.1. Modeling the Server

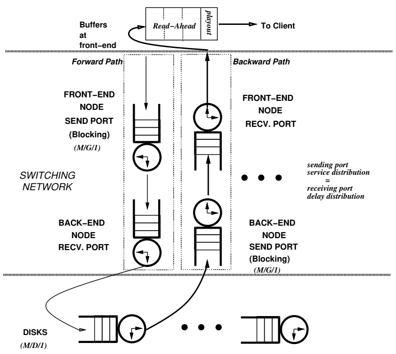


Figure 2: Analytical Model – Server Queues

The server can be modeled as three logical queues: i) the forward path queues of the switching network, ii) the disk queues, and iii) the backward path queues of the network (Figure 2). The external network is not considered a part of the server and is not modeled. To model such a server, let us consider a system consisting of N nodes and D disks, supporting S streams. Let the streams be periodic, with request arrival rates of $[\lambda_1, \lambda_2, \lambda_3 ..]$, and the mean arrival rate of a stream being $\lambda = \sum \frac{\lambda_i}{S}$ blocks/sec. Let the time required to transfer a large block across the switch be $1/\mu_{block}$, and for a small request be $1/\mu_{request}$. This includes a connection set up time and a transfer time based on block size.

3.1.1. The Disk Queue

Each disk in the disk subsystem can be modeled as a single server queue. Under the assumption that streams are independent, have random startup times and positions, and the number of streams is large, we can approximate the arrival process for read requests seen by the disk subsystem to be Poisson, with mean λS. Since a disk is randomly selected by a stream the arrival process at a particular disk is Poisson with a mean of $\lambda_d = \lambda S/D$. The service time at the disk is assumed to be deterministic (since the block size is large) and equals the mean service time $1/\mu_d$, which includes seek and rotational latencies, and the transfer time. Each disk thus behaves like an M/D/1 queue, given that we are only interested in the tail of the distribution. From the results for an M/D/1 queue [7] we know the queue length distribution, p_n, from which the delay distribution can be derived. If the block size on the disk is variable, the disk model can be modified to be an M/G/1 queue, with the disk service time distribution depending directly on the block size distribution.

3.1.2. The Forward and Backward Switch Path

The switching network consists of N nodes connected by a cross-point switch. Each node has two ports connected to the switch; a sending port and a receiving port. The forward path consists of a small read request sent from a logical front-end node's sending port to a logical back-end node's receiving port.

The forward path is modeled as two queues, a blocking sending queue and a non-blocking receiving queue. Since the service time at the sending queue depends on the queuing delays at the receiving end, we use the *delay distribution* of the receiving queue to determine the *service time distribution* of the sending queue. Each back-end receiving port is modeled as an M/D/1 queue with one server with the service time being $1/\mu_{request}$. The arrival rate at the back-end receiving queue is $\lambda S/(N_{be})$, where N_{be} is the number of back-end nodes. The front-end sending port is modeled as an M/G/1 queue, with the service time distribution being equal to the delay distribution at the receiving port. The arrival rate at the front-end sending queue is $\lambda S/(N_{fe})$, where N_{fe} is the number of front-end nodes.

The backward path consists of the transfer of a large data block from the logical back-end node's sending port to the requesting front-end node's receiving port. The backward path can be modeled in a similar fashion as the forward path, with the roles of front-end and back-end nodes reversed. Also, the transfer time for a large block is much larger than that for the small request in the forward path.

3.1.3. Mapping Different Architectures

The server model can be directly applied to the two-tier architecture and be used to compute the utilization of the ports. The logical front-end and back-end nodes can be mapped to physical front-end and back-end nodes. Similarly, the forward and backward paths can be mapped to two separate links on the switch.

In the flat architecture each node functions as a logical front-end or back-end. The links in the network are common for both forward and backward paths. Since there are two types of messages at every port, the arrival rate at any port (sending or receiving) is, $\frac{2\lambda S}{N}(1-1/N)$, given that $1/N^{th}$ of the requests per stream are serviced locally. The receiving port model is modified to be an M/G/1 queue, with two service times, both equally likely. The mean service time, $1/\mu_r$, equals the average time to send a message (short request and large block) across the switch. The sending port model is similar to that defined in Section 3.1.2.

3.2. Computing Loss Probability

The QoS criterion assumed to evaluate the server design, is the loss probability of the streams. The *loss probability* is defined as the probability that a block requested by a stream is not available for play-out by its deadline. To compute it easily, we assume that the disk delay, $1/\mu_d$, is the dominant part of the overall delay (an accurate model involves the convolution of delay distributions). We reduce the total delay bound (deadline), by the mean delay at the switch, δ_{net}, and the mean processing delay, δ_p, to get the delay bound at the disk. Thus, if we assume that the deadline of a request based on stream requirements is, D_{ddl}, then the maximum delay, D_{max}, that a block can encounter at the disk and still meet its deadline is, $D_{ddl} - \delta_{net} - \delta_p$. The loss probability can be computed from the disk queue length distribution, p_i, and is given by $P[loss] = P[delay > D_{max}] = 1 - \sum_{i=0}^{i/\mu_d \leq D_{max}} p_i$.

4. Analysis and Evaluation

For completeness and validation of the analytical model, we describe a simulation model and compare the results.

4.1. Simulation Model

In the simulation model, all resources (nodes, ports, disks) are modeled as single server queues. The disk service time includes the seek time (square root function of the arm distance), the rotational latency and the block transfer time. The disks are assumed to be re-calibrated at regular intervals, blocking all requests during that period. We assume that the disks are zoned with different transfer rates per zone. The parameters selected for the switch are similar to a commercially available FCS product. More details of the configuration can be found in [15].

The arrival pattern of the streams is a distribution function handling various lengths of play with pause and resume requests. The simulation model is implemented using CSIM [14], and Table 1 gives the default parameter values used.

Description	Default
Front-End Processing Delay	8.192 msec.
Back-End Processing Delay	8.96 msec.
Block Size	256 KB
Number of Streams	500
Stream Data Rate	0.2-0.5 MB/sec
Processor Speed	100 MHz
Switch Bandwidth	20MB/sec
Switch setup overhead	2 msec.
Front-end s/w overhd.	6.2 cyc./byte
Back-end s/w overhd.	3.5 cyc./byte
Mean Seek Time	10 msec.
Number of Zones	10
Disk RPM	7200
Calibration delay/ 2mins	200 msec.

Table 1: Default Parameters Used

To validate the two models, we compare the the overall loss probability of streams as predicted by the analytical model with that observed by the simulation model. As

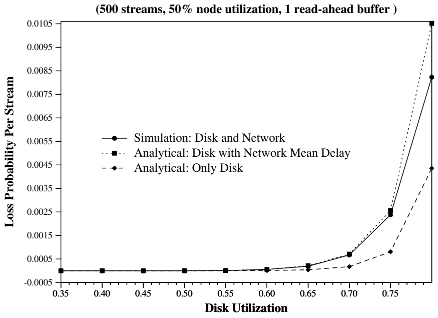


Figure 3: Loss Probability—Analytical vs. Experimental

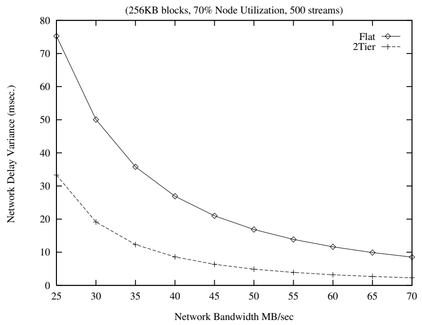


Figure 4: Comparison of Switch Delay Variance- Analytical Model

depicted in Figure 3, the analytical model using only the disk queue predicts a lower loss probability than the actual simulation. The modified disk queue model (with the mean switching network and processing delays) predicts a slightly higher loss probability than the simulation and is within 2% for disk utilization up to 75%.

4.1.1. Comparison of Architectures

The analytical and simulation models of the network can be used to provide an insight into the performance of the flat and two-tier architectures. It seems intuitive that the port utilization in a two-tier architecture in the backward path (large block) will be much greater than the forward path (small request). On the other hand, in a flat architecture, ports are uniformly utilized, as both types of messages are handled at each port. However, the flat architecture will have more variance in the switching delay, due to two types of messages.

No. of Nodes	Two-Tier				Flat	
	Front-End		Back-End		SND	RCV
	SND	RCV	SND	RCV		
56	7.2%	42%	57.2%	7%	29.1%	24.5%
50	8.3	48	70	8	34.3	28
44	9.4	54	85	9	40	31.5
40	10.5	60	100.5	10	46	35
36	11.6	66	130	11	52.6	38.5
32	12.8	72	164	12	59.8	42

Table 2: Port Utilization- Analytical Results

We can analytically compute the port utilization for the two architectures for different node utilization, assuming a 256KB size block and a switch bandwidth of 20MB/sec. Table 2 shows that the port utilization of the two-tier architecture are highly skewed, with the back-end sending port saturating for a 40 node system (62% node utilization), while the corresponding flat architecture ports are less than 50% utilized. As the number of nodes is decreased, the node utilization increases and the saturation of the ports in the two-tier architecture becomes more severe. Thus, the two-tier architecture has poor performance when the switch bandwidth is low and the node utilization is high.

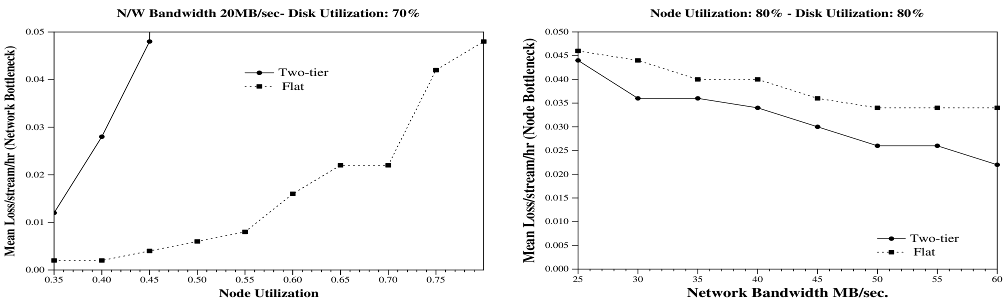
The analytical model can also be used to compute the variance in the round-trip switching delay for the two-tier and flat architectures. The flat architecture has a much larger delay variance in the forward path when compared with that of the two-tier architecture as there are only small messages in the two-tier forward path. The total variance is higher for the flat architecture. The variance computed for the two-tier architecture is less than half of that for the flat architecture as shown in Figure 4.

The overall performance of the two architectures depends on the network and node parameters. We use the simulation model to quantify the stream loss rates depending on what (node or network) becomes the system bottleneck. Figure 5(a) shows the loss rates increasing rapidly due to saturating ports, when the node utilization is just 50%. When the switching network is not a bottleneck (i.e., switch bandwidth is high), the port utilization is small and the delay variance dominates. The two-tier has smaller variance, leading to lower loss rate as shown in Figure 5(b). Thus, to summarize, the flat architecture is preferred due to the more optimal port utilization, when the switching network is a bottleneck. When the network is not a bottleneck, the two-tier architecture is desirable for practical considerations as it is simple to configure and manage and has better performance.

4.1.2. Optimal Block Size Determination

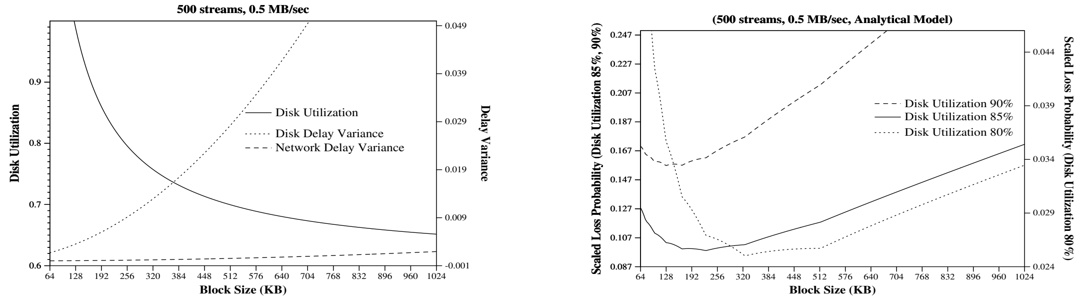
The analytical model can be used to determine the optimal block size on disk, that will minimize the probability of a request missing its deadline. As the block size increases, the thruput of the disk (or switch) increases due to smaller overhead of disk read latency (or switch startup delays). Also, for the same play-out rate of the streams, the number of blocks requested per second decreases. All this favors a larger block size. However, when the block size is large, the variance in the response times at the disk and network increases. Even a small queue length results in long delays. This results in a smaller block size being favored. Figure 6(a) shows the two phenomena for different block sizes. The tradeoff between disk thruput and delay variance results in an optimal block size [17].

We can compute the optimal block size from the delay distribution at the disk. The delay distribution and the service time distribution at the disk can be modified to be a function of the block size. The loss probability can be then computed as a function of block size using the method described in Section 3.2. Figure 6(b) shows the loss probability using the analytical model for disk utilization of 80, 85 and 90%. The loss probability is scaled based on the block size as the loss of a larger block involves losing more data. For a



(a) Switching Network Bottleneck (b) Node Bottleneck

Figure 5: Flat Vs. Two-Tier-Simulation Results



(a) Block size Vs. Disk Performance and Delay Variance (b) Optimal Block Size Determination

Figure 6: Effect of Block Size

80% disk utilization a block size in the range of 250-590KB is within 5% of the optimal. When the disk utilization is increased to 85% the optimal block size range shifts to 130-330KB. A further increase in disk utilization to 90%, results in a block size range of 64-290KB to be within 5% of the optimal. As the disk utilization is increased, the effect of disk delay variance on the loss probability increases, shifting the optimal range to smaller block sizes. The optimal block size selected is the one that lies in the intersection of the optimal size range for the different operational disk utilization.

4.1.3. Effect of Read-ahead Buffering

Due to the sequential nature of stream access, the blocks required later for play-out can be pre-fetched and stored in the read-ahead buffers at the front-end node. Read ahead buffering is a simple approach to increase the deadline bound of a request, leading to a decrease in the loss probability, while adding some extra cost. The delay bound of a block request is the time interval between when the request is sent and the time at which the block is transmitted. Specifically, if there are k read-ahead buffers associated with a stream, the delay bound of a request increases by a factor of k from the original bound which was without buffering. The loss probability with k read-ahead buffers can be derived using the method described in Section 3.2. Observe that this loss probability does not capture congestion at startup to fill multiple buffers. It is valid only when the play-out of a stream begins after all read-ahead buffers are full.

The optimal block size on disk also depends on the amount of read-ahead buffering. If a fixed size buffer space[1] is available per stream, there is a tradeoff between the disk performance favoring a large block size and the lower loss probability due to multiple read-ahead buffers favoring smaller blocks. For a given amount of memory, a single read-ahead buffer with larger block sizes, performs worse than multiple buffers with smaller blocks. However, when the available memory is large, the gain by proportionately increasing the number of read-ahead buffers is small, consequently the block size gets governed more by disk performance than read-ahead memory size.

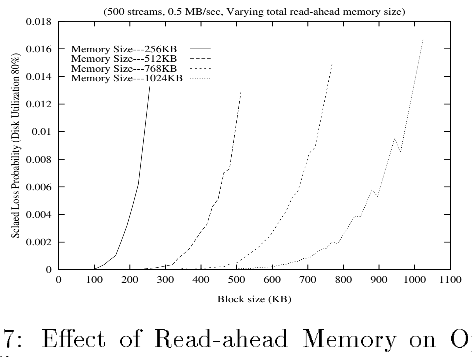


Figure 7: Effect of Read-ahead Memory on Optimal Block Size

We use the analytical model in Figure 7 to show the effect of read-ahead memory size on the optimal block size. In Figure 6 in the previous section, the optimal block size was determined based on the disk performance and variance

[1] play-out + read-ahead buffers

and at 80% disk utilization it was in the range of 250-350 KB. If the read-ahead buffering is factored in computing the block size, the optimal block size shifts to a smaller size (less than 90KB), for a small sized (0.25MB)read-ahead memory. As the memory size increases to 1MB the optimal block size also increases (to less than 500KB).

4.1.4. Scheduling Effects

We quantify the effects of different scheduling policies on the stream loss rates. Typical I/O device drivers assume a FCFS (First Come First Served) schedule. With EDF (Earliest Deadline First [12]), each stream request for a block needs to be tagged with a deadline, which is a function of the read-ahead buffer size and the play-out rate of the stream. It is intuitive that EDF will perform better than FCFS for real-time requests. However, implementing EDF requires significant modifications to existing I/O subsystem software [1] that uses standard UNIX interfaces. If the arrival time of a request is proportional to its deadline, FCFS and EDF are nearly equivalent; however, due to a clustered architecture and switch delays, requests from different nodes may arrive out of order at the disk. This effect is even more pronounced when the play-out rates of streams are different.

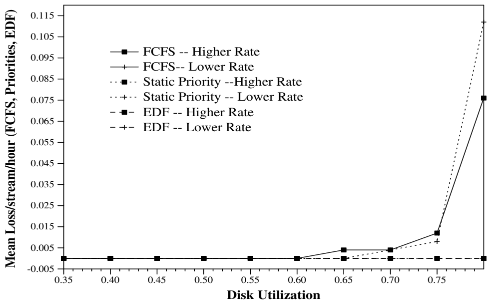


Figure 8: Scheduling Effects–Different Playout Rates

We simulate 3 different scheduling policies at the disk: FCFS, Static Priority Scheduling and EDF, with streams having 2 play-out rates (typical data rates of MPEG-2, 0.5 MB/sec and MPEG-1, 0.2 MB/sec). Figure 8 shows that the disk utilization can be increased by 15% by using EDF instead of FCFS or static priority. The FCFS policy cannot distinguish between requests of different rates, increasing the losses of the faster rate, while the static priority causes starvation of the lower rate.

4.1.5. Scalability

An important parameter in designing large scale servers is the capacity to which the system can scale. The factors that affect scalability are the striping width, load imbalances, and the interconnection network bottlenecks. With wide striping and balanced load the number of streams scales linearly with the number of nodes in the system, till the maximum switch capacity is reached. However, scalability could be moderately affected by the presence of a number of short object clips (less than the stripe width) or pausing streams, leading to load imbalances. The simulation model can be used to measure the loss rates as the system is linearly scaled up with load imbalances.

Figure 9 shows the number of streams per node that can be supported for a mean loss rate of 0.01 per streams/hour,

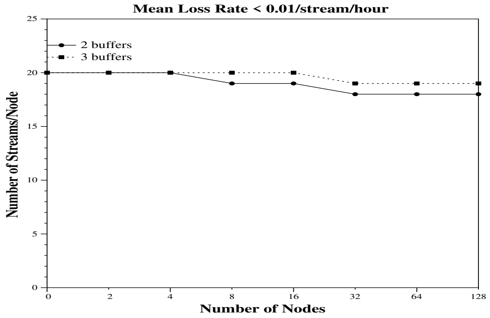


Figure 9: Scalability–Streams per node

as the system size is increased from 2 to 128 nodes. The clip size distribution is exponential with a mean of 5 minutes. At 100% utilization, each node can support 20.6 streams. At 20 streams per node (97% node utilization) the system scales to 4 nodes. The scalability increases to 32 nodes with 19 streams per node (92% node utilization) and to 128 nodes with 18 streams per node (87% node utilization). Increasing the number of read ahead buffers from two to three, results in a 4 to 8 fold increase in scalability.

5. Prototype Implementation

The results obtained from the analysis of the various parameters in a clustered architecture formed the basis of our scalable video server prototype. The switch used in our prototype is a high performance SP switch connected to multiple RS/6000 machines. To provide for a "single system image" to the VOD application, in such a loosely coupled architecture, it is necessary that every node has an identical view of all the I/O resources in the cluster. This is provided by a layer of software called *virtual shared disk* (VSD) [1], which is a software disk proxy that allows a remote I/O device to appear as if it was a local device. The VSD software is packaged as a device driver and appears to the system as a logical volume or hard disk, providing the same interface to the application as a local device. Furthermore, all processing for a request at the VSD server is done at the interrupt level; there is no associated kernel process or daemon. This is in contrast to network file-systems where daemons are used at server nodes.

The VSD software makes the cluster architecture and remote I/O handling transparent to the nodes. A standard UNIX based I/O interface is provided by VSD with the requests being ordered by their arrival time. The enhanced version, called real-time VSD, handles requests with deadlines. This is done by specifying an *ioctl* interface, which passes a deadline along with the block addresses. The deadline based ordering is done at the VSD server queue. A layer of application software runs on the front-end nodes that interacts with VSD to fetch the blocks of a video object at a given rate and a given priority level.

The software layer at the front-end nodes maintains the meta-data associated with each video object in the system. The logical block represents the size of a stream request unit to the VSD. Typically a block is split into multiple 64 Kbyte packets at the switch communications layer. We use wide striping across nodes in our prototype with random placement within a stripe group. Since video objects could be large (> 2 GB) VSD supports large video objects even on 32-bit machines. The prototype allows for either flat or two-tier architectures. The size of a logical block on disk is flexible, although the typical size used is 256KB which is in the optimal size range discussed in Section 4.1.2. The read-ahead buffers are allocated from a shared memory segment, which is also used to cache the data read from disk using a simple LRU scheme [8].

Front-end nodes have a T1 adapter which connects the node to a set-top box via a T1 (1.53 Mbits/sec) line. The set-top box supports hardware MPEG decoding and plays the output to a TV set. The nodes also have an ATM adapter that connects to another client RS-6000 via an ATM (OC3) connection. Since the number of clients with hardware support is small in the prototype, the streams can be played out to a dummy device to measure the performance of the system. Each node can push around 20 MBytes/sec of stream data at about 80% CPU utilization.

6. Conclusions and Future Work

We have examined several performance issues in the design of clustered video servers that support thousands of concurrent streams. Specifically, we examined different cluster architectures, determination of optimal block size on disk, read-ahead buffering and scheduling policies.

We described an analytical model for the disk and switch subsystem. The model was validated using extensive simulations. We examined the performance of different cluster configurations and concluded that flat architectures are practical for low bandwidth switches as the port utilization are better balanced. When the switch bandwidths are high and the processing nodes become the bottleneck, the two-tier architecture, which has lower delay variance, becomes viable. We compute the optimal striping unit size on disk using the analytical model and show the tradeoff between disk performance and loss rates. We studied the relation of the block size with the amount of memory available for read-ahead buffers and its effect on the real-time performance. We also quantified the performance gain in using EDF scheduling instead of simple FCFS or static priorities.

We described the implementation of our prototype design based on the results of the analysis. In our prototype we use wide striping of video-objects across all the nodes of the cluster. The virtual shared disk software layer provides a single image of the I/O subsystem, that includes all the disks in the cluster, to all the nodes in the system.

Acknowledgments: We would like to thank Christos Polyzois, Martin Kienzle, and Roger Haskin for their comments and ideas during several discussions about the architecture model and model parameters. We would also like to thank Manoj Kumar for his help in selecting disk parameters.

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*, November 1994.

[2] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered striping in multimedia information systems. *Proceedings of the 5th SIGMOD*, May 1994.

[3] M. M. Buddhikot, G. M. Parulkar, and J.R. Cox. Design of a large scale multimedia storage server. *Journal on Computer Networks and ISDN Systems*, 1995.

[4] E. Chang and A. Zakhor. Scalable video data placement on parallel disk arrays. *Proceedings of Storage and Retrieval for Image and Video Databases II*, February 1994.

[5] H.-J. Chen and T.D.C. Little. Physical storage organizations for time-dependent multimedia data. *Proceedings of the fourth international conf. on the foundations of data organizations and algorithms*, October 1993.

[6] C. Freedman and D. DeWitt. The SPIFFI scalable video-on-demand server. *Proceedings of SIGMOD*, June 1995.

[7] D. Gross and C. M. Harris. *Fundamentals of Queueing Theory*. John Wiley and Sons, 1985.

[8] R. Haskin. Personal Communication, 1994.

[9] R. Haskin and F. L. Stein. A system for delivery of interactive television programming. *Proceedings of COMPCON*, Spring 1995.

[10] J. Hsieh, M. Lin, J.C.L. Liu, David D.C. Du, and T. M. Ruwart. Performance of a mass storage system for video-on-demand. *Proceedings of INFOCOM-1995*, March 1995.

[11] J. Hsieh et.al. Performance of a mass storage system for video-on-demand. *Journal of parallel and distributed computing*, submitted.

[12] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46 – 61, January 1973.

[13] L.A. Rowe and B.C. Smith. A continous media player. *Proceedings of the 3rd Intl. NOSSDAV workshop*, November 1992.

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

[15] R. Tewari, D. Dias, R. Mukherjee, and H. M. Vin. Real-time issues for a clustered multimedia server. *IBM- Research Report - RC-20020, also available from http://www.cs.utexas.edu/users/dmcl*, April 1995.

[16] H.M. Vin and P.V. Rangan. Designing a multiuser hdtv storage server. *IEEE journal on selected areas of communications*, January 1993.

[17] H.M. Vin, S.S. Rao, and P. Goyal. Optimizing the placement of multimedia objects on disk arrays. *Proceedings of the Second IEEE International Conference on Multimedia Computing and Systems, Washington, D.C.*, pages 158–165, May 1995.