Techniques for Increasing the Stream Capacity of a Multimedia Server *

Divyesh Jadav
ECE Department & CASE Center
Syracuse University , Syracuse, NY 13244
divyesh@cat.syr.edu

Alok Choudhary
ECE Department & Technological Institute
Northwestern University, Evanston, Illinois 60208.
choudhar@ece.nwu.edu

Abstract

A server for an interactive distributed multimedia system may require thousands of gigabytes of storage space and high I/O bandwidth. In order to maximize system utilization, and thus minimize cost, the load must be balanced among the server's disks, interconnection network and scheduler. Many algorithms for maximizing retrieval capacity from the storage system have been proposed. This paper presents techniques for improving server capacity by assigning media requests to the nodes of a server so as to balance the load on the interconnection network and the scheduling nodes. Five policies for request assignment are developed. The performance of these policies on a server model developed earlier is presented.

1. Introduction

Digitalization of traditionally analog data such as video and audio, and the feasibility of obtaining networking bandwidths above the gigabit-per-second range are two key advances that have made possible the realization, in the near future, of interactive distributed multimedia systems. Multimedia data differs from unimedia data in the diversity of data sizes and the need to provide real-time guarantees for playback (video and audio data). One of the most pervasive interactive multimedia applications is *media-on-demand* which refers to the possibility of a consumer interactively retrieving multimedia data over high-speed networks from geographically distributed media servers. It is anticipated that a distributed media-on-demand (MOD) system will be built in a hierarchical manner, with clients connected to neighbourhood servers, which are connected to metropolitan servers, which in turn are connected to massive archive servers. This hierarchy of servers is similar to the memory hierarchy in a computer system. When the processor issues a request for data, the time penalty for retrieving the data is directly proportional to the hierarchical distance of the data from the processor, which constitutes the apex of the hierarchy. In a hierarcical MOD system, a similar relationship exists in this case with regard to the time delay for data retrieval by a client. On account of the real-time and storage capacity requirements of multimedia data, the servers are high-end machines with gigabytes of storage space and high I/O bandwidth. Moreover, higher up the hierarchy is the server, the higher its storage and I/O capacity. In order for the entire system to be cost effective, each server must be cost effective. Hence, it is essential to maximize the utilization of each resource type of the server. This paper deals with techniques to maximize the utilization of one of the resource types of a MOD server.

1.1. Related Work

Researchers have proposed various approaches for the storage and retrieval of multimedia data. [1] characterized the disk-level tradeoffs in a multimedia server. [2] proposed a model based on constrained block allocation. [3] proposed striping media data across several disks in a round robin fashion. The effective retrieval bandwidth is then proportional to the number of disks used. Our server model (section 2) is similar to this model. Issues in designing MOD servers are discussed in [4]. A component-wise instrumentation of the delays in a MOD [5], showed that variable delays become performance bottlenecks at high loads. Techniques for balancing the load on the storage devices of a MOD server were developed in [6] .Given the fact that a high performance MOD server consists of multiple processors (nodes) connected by an interconnection network, not much work has been reported on efficient use of the interconnection network so as to maximize server capacity. In this paper we ad-

*This work is supported by Intel Corporation, NSF grants CCR-9357840, CCR-9509143, and the New York State CASE Center at Syracuse University. The authors thank the Caltech CCSF facilities for providing access to the Intel Paragon.

dress this issue by developing five policies - round robin (RR), minimum link allocation (MLA), minimum contention allocation (MCA), weighted minimum link allocation (WMLA) and weighted contention allocation (WMCA). We have developed and implemented a logical model for a MOD server [5]. Performance results of the five policies under this model implemented on the Intel Paragon parallel computer are presented.

The rest of this paper is organized as follows : Section 2 explains the server and data scheduling model. Section 3 presents the five allocation strategies. We present and analyze performance results in Section 4. Conclusions are presented in Section 5.

2. Server and Scheduling Model

At the heart of the system is a high-performance server optimized for fast I/O. A parallel machine is a good candidate for such a server because of its ability to serve multiple clients simultaneously, its high disk and node memory, and the parallelism of data retrieval that can be obtained by data striping. We assume that (a) the server is connected to clients by a high-speed wide-area network, which delivers data to clients reliably and at the required bandwidth. (b) Clients have *hard* deadlines i.e. they cannot tolerate jitter. (c) The data are stored at the server in compressed digital form, with the decompression being done at the client end. The server consists of multiple nodes interconnected by a network. Each node is a computer in its own right, with a CPU, RAM and secondary storage. There are three types of nodes, **interface (I) nodes, storage (S) nodes** and the **object manager (O) node**. The object manager receives all incoming requests for media objects, and *delegates the responsibility* of serving a request to one of the interface nodes. *I* nodes are responsible for scheduling and serving stream requests that have been accepted. Storage nodes *store* multimedia data on their secondary storage devices in a striped fashion, and *retrieve* and transmit the data to an interface node on request. The assumption about the architecture of the interconnection network is that any node can transfer data to and from any other node with approximately the same latency, under conditions of light load. For the purposes of this paper, we assume that the interconnection network is a *direct* network[7].

The data is compressed and striped across the storage nodes in a round-robin fashion. The number of nodes across which a data object is striped is called the *stripe factor (SF)*. The collection of *SF* storage nodes that store an object is called a *striping group*. The data stored at a storage node consists of chunks of the object. The collection of chunks is called a *subobject*. Note that the collection of chunks of a subobject do not constitute a contiguous portion of the object; however the data *within* a chunk is a contiguous part of the entire object. This contiguous data is called a *stripe fragment*.

Symbol	*Description*	Units
R_{pl}	Required playback rate	bytes/sec
P_I	Size of packets sent by an I node	bytes
δ_I	Duration of a packet sent by an I node	sec
B_I	Buffer size at an I node	bytes
P_S	Size of packets sent by a S node	bytes
δ_S	Duration of data in B_I	sec
T_f	Period of issuing fetches to S nodes from I node	sec
SF	Stripe factor	-

Table 1. The parameters used in this paper

Table 1 shows the parameters used by our model. δ_I is the time for which a packet sent by an I node to a client will last at the client. Hence this is also the deadline by which the next packet from the I node must be received at the client. Its value is given by:

$$\delta_I = \frac{P_I}{R_{pl}} \tag{1}$$

Once the requested SF stripe fragments from the S nodes have arrived at the destination I node, the latter arranges them in the proper sequence and continues sending packets of size P_I to the client no less than every δ_I seconds. The buffer at the I node will last for δ_S time, before which the next set of stripe fragments must have arrived from the S nodes. The average time to retrieve P_S bytes from a S node is given by

$$\delta_{io} = \delta_{rq} + \delta_{avg_{seek}} + \delta_{avg_{rot}} + \delta_{tr_{P_S}} + \delta_{nw_{P_S}} \tag{2}$$

where δ_{rq} is the time delay for a request from an I node to reach a S node, $\delta_{avg_{seek}}$ and $\delta_{avg_{rot}}$ are the average seek and rotational latencies for the disks being used, $\delta_{tr_{P_S}}$ is the disk data transfer time for P_s bytes, and $\delta_{nw_{P_S}}$ is the network latency to transport P_s bytes from a S node to an I node. Note that equation 2 uses average seek and rotational latencies for disk accesses, for reasons explained in[5].

3. Stream Assignment Policies

In a given server configuration, only a finite number of nodes (interface nodes) would be connected to the high speed wide area network. Moreover, their position in the server architecture would be fixed a-priori. Secondly, since the secondary storage capacity for a given server configuration is finite, only a finite number of media objects can be stored in the secondary storage system at a time. In order to maximize the pool of potential clients, there would exist a tertiary storage system from which the most frequently requested objects are materialized on the secondary storage subsystem. However, the number of objects stored on secondary storage would be a slowly changing set. Given the fact that the position of the I nodes and the storage pattern of data on S nodes is fixed, the problem is one of assigning accepted media requests to I nodes so as to minimize the incremental workload due to the new requests on the server's resource types. This allows the server to maximize the number of streams that it can source. The network communication time is the sum of two factors - the network latency in the absence of blocking, and the blocking time due to link contention in the interconnection network i.e.,

$$\delta_{nw_{P_S}} = \delta_{nw_{comm}} + \delta_{nw_{bl}} \tag{3}$$

For a given message size and interconnection network, the former is fixed; while the latter depends on the network traffic. There is another variable delay in the retrieval time : when multiple requests arrive at a S node, only a finite number of them can be served at a given time; this causes a queueing delay at the S nodes. If δ_{S_Q} denotes this queueing delay, equation 2 requires to be modified to :

$$\delta t_{io} = \delta_{rq} + \delta_{seek} + \delta_{rot} + \delta_{tr_{P_S}} + (\delta_{nw_{comm}} + \delta_{nw_{bl}}) + \delta_{S_Q} \tag{4}$$

The effect of the variable delays $\delta_{nw_{bl}}$ and δ_{S_Q} on total retrieval time at various workloads was studied in[5]. At heavy workloads, these delays become the limiting factors on stream capacity. Any mechanism that reduces either or both of these quantities improves performance. In this paper we show that techniques that reduce $\delta_{nw_{bl}}$ by minimizing link contention translate into the ability to support more streams.

3.1. RR Assignment Policy

This is the simplest policy. If n is the total number of interface nodes, and the ith request was assigned to interface node k, then the $(i+1)$th request will be assigned to interface node $(k+1) \bmod n$. This policy is simple, requiring $O(1)$ time to execute. The maximum workload imbalance in terms of number of streams served per I node is at most 1. However, this policy does not balance or minimize the load imposed on the interconnection network.

3.2. MLA Policy

This policy aims to minimize the total number of links that the data for an object has to travel from the SF storage nodes on which it is stored to the I node which sends it to the outside world. If $l_{I_i S_j}$ denotes the number of links between interface node I_i and storage node S_j, then the cost of assigning a stream request to interface node I_i under this policy is :

$$CA_{MLA}(I_i) = \sum_{j=1}^{SF} l_{I_i S_j} \tag{5}$$

This policy will assign a request to interface node p,

$$p = i : (\min(CA_{MLA}(I_i)) \qquad i = 1, 2, ..., n) \tag{6}$$

i.e., the request is assigned to that interface node which is closest in terms of the total number of links that need to be traversed from the S nodes to the I node. This policy tries to minimize the number of streams using a given link. But, it does not take into account the preexisting link load. Nor does it balance the total stream load among all the interface nodes.

3.3. MCA Policy

In this policy, state information is maintained about the usage of each link. Specifically, whenever a new request is assigned to an interface node, the load imposed on the interconnection network by data traffic due to that stream is calculated and the total load on the network due to all streams is updated. When a stream terminates, the load due to it is decremented from the total network load. If $c_{I_i S_j}[k]$ is the cost of using the kth link on the path from storage node S_j to interface node I_i, then the cost of assigning a stream request to interface node I_i under this policy is :

$$CA_{MCA}(I_i) = \sum_{j=1}^{SF} \sum_{k=1}^{l_{I_i S_j}} c_{I_i S_j}[k] \tag{7}$$

This policy will assign a request to interface node p,

$$p = i : (\min(CA_{MCA}(I_i)) \qquad i = 1, 2, ..., n) \tag{8}$$

The cost of using a link is directly proportional to the traffic that the link carries. The link traffic due to accepting a new request is updated as follows :

```
for ( $j = 1$ to $SF$ )
    for ( $k = 1$ to $l_{I_p S_j}$ )
        $c_{I_p S_j}[k] = c_{I_p S_j}[k] + ld$        (9)
```

where ld is a scalar that reflects the load imposed by the new stream on link k. Its value is implementation-dependent : it depends on the network, the size of packets being transferred and the playback rate of the stream. The premise behind this policy is that the load should be distributed evenly over the interconnection network. If some links are more heavily used than others, contention in these links increases network blocking effects, which in turn degrades server performance. Note that since the traffic pattern in the interconnection network for data packets consists of storage nodes sending data to interface nodes, there is a possibility of hot spots developing at the links around the interface nodes. This policy tries to prevent the formation of such hot spots by allocating requests to interface nodes so that aggregate link traffic is distributed as evenly as possible over the entire interconnection network.

3.4. WMLA and WMCA Policies

The MLA policy tries to minimize the total number of links that data for a stream will have to travel, while the MCA policy tries to minimize link contention by distributing traffic over more lightly used links. However, neither of them tries to balance the load across the interface nodes. An interface node can source only a finite number of streams; beyond this limit client deadlines may be missed due to excessive scheduling overhead. The weighted MLA and MCA policies try to balance the load across both the network and the I nodes. This is done by factoring in the number of streams that a candidate I node is serving in the cost equation. Specifically, if M_{I_i} is the number of streams being served by interface node I_i, then the cost of assigning to it the responsibility of serving a request under WMLA and WMCA (respectively) is :

$$CA_{WMLA}(I_i)' = \alpha * M_{I_i} + \beta * CA_{MLA}(I_i) \quad (10)$$

$$CA_{WMCA}(I_i)' = \alpha * M_{I_i} + \beta * CA_{MCA}(I_i) \quad (11)$$

where α and β are fractions that sum to 1, and $CA_{MLA}(I_i)$ and $CA_{MCA}(I_i)$ are given by equations 5 and 7 respectively. The criterion for selecting a candidate I node is similar to that for the respective unweighted cases (equations 6 and 8 respectively); so are the running times. The value to assign to the weight is a design choice that depends on the network size and topology, routing strategy and the maximum number of streams that an I node can source. Note that WML(C)A with $\alpha = 1$, $\beta = 0$ is equivalent to RR, while WML(C)A with $\alpha = 0$, $\beta = 1$ is equivalent to ML(C)A.

4. Performance Evaluation

We have implemented our logical server model on the Intel Paragon parallel computer. The Intel Paragon is a mesh-based architecture with Intel i860XP microprocessors. Interprocessor communication is done using *wormhole routing* [7]. The data access pattern is

Description	Value
Required playback rate (R_{pl})	1.5 Mbits/sec
Size of packets sent by an I node (P_I)	80 Kbytes
Size of packets sent by a S node (P_S)	160 Kbytes
Minimum disk seek time	4 msec
Maximum seek time	30 ms
Time for one rotation	10 ms
Disks per storage node	2
Disk data transfer rate	10 MBytes/sec
Stripe Factor (S)	4
Num. of Interface nodes	11
Num. of Storage nodes	36
Num. of media objects	100
Evaluation machine	Intel Paragon

Table 2. The parameter values used

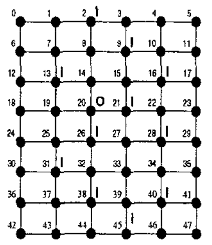

Figure 1. Distribution of I, S and O nodes

assumed to follow a Zipfian distribution with parameter 0.271[8]. Due to storage space and availability of real-world data limitations, the disk access part was simulated by elapsing the system timer on each storage node. a random placement model. Table 2 shows the values of the parameters defined in table 1 that we used for our simulation. The traffic was generated as follows. Initially, requests for videos are sent to the object manager at random times, with average interarrival time of 2 seconds. Each video lasts for about 10 minutes. As soon as a video terminates, a new request for a video is sent to the object manager. Figure 1 shows the distribution of I nodes and S nodes used[1]. The value of the parameter *ld* for the MCA policy (subsection 3.3) we used was 1, since the value of P_S and R_{pl} is the same for all streams. The load on the server was increased by incrementally increasing the number of object requests. The same data distribution and request pattern were used for each experiment.

4.1. Comparison of load balancing ability

In order to compare the distribution of stream requests to the I nodes, the number of streams served by each interface node was measured for the same total number of streams served for each policy. Figure 2 illustrates these values for each I node in figure 1 for a total server load of 565 streams for the RR policy and for a load of 630 streams for the other four policies. (the maximum number of streams supported by RR

[1]Numerous tradeoffs are possible with respect to the data partitioning strategy, which are well reported in [3]. However, these are not the subject of this paper.

Policy	Average streams per I node	Standard Deviation (σ_i)
RR	51.36	0.48
MLA	57.27	33.75
MCA	57.27	22.49
WMLA ($\alpha = \beta = 0.5$)	57.27	8.85
WMCA ($\alpha = \beta = 0.5$)	57.27	4.07

Table 3. Standard Deviation of stream load

was only 565. Each of the other 4 policies supported at least 630 streams). We first compare the RR, MLA and

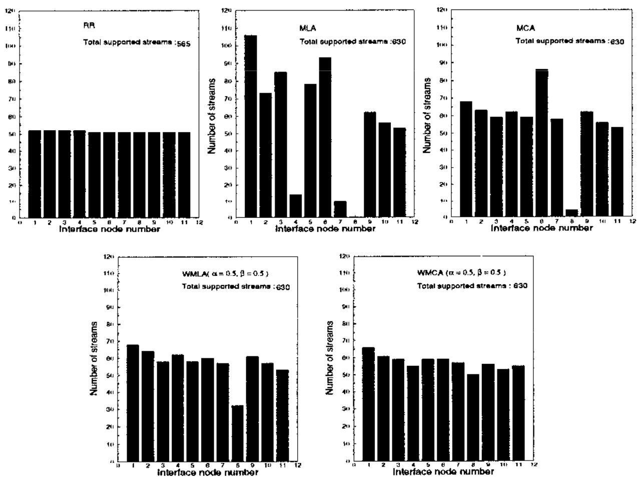

Figure 2. Comparison of request assignment

MCA policies. We note from the figure that the RR policy performs best in terms of balancing stream load across the interface nodes. A measure of the degree to which a request assignment policy balances stream load across the interface nodes is the standard deviation of the number of streams per interface node, σ (Table 3). The standard deviation of the number of requests per I node for MLA, σ_{MLA}, is the worst among the standard deviations of RR, MCA and MLA. The reason for this is the skewed data access pattern. Consider now the WMLA and WMCA policies. The graphs in figure 2 for the WMLA and WMCA policies are for values of α and β of 0.5 each. In this case too, the load balancing of WMCA is better than that of WMLA. Although σ_{WMLA} (8.85) is lesser than σ_{MLA} (33.75), it is still greater than σ_{WMCA} (4.07). In summary, the weighted assignment policies improve the load balancing ability at the I nodes as compared to the pure schemes; however MCA gives better performance than MLA, and WMCA gives better performance than WMLA.

4.2. Comparison of Network Blocking Time

With reference to equation 4, the networking blocking time for each packet requested by an I node from a S node, $\delta_{nw_{bl}}$, was measured as follows : δ_{seek} and δ_{rot} were measured at run time. Given a disk and a value of P_S, $\delta_{tr_{P_S}}$ can be computed. δ_{S_Q} is given by $\delta_{S_Q} = \Delta_t - (\delta_{seek} + \delta_{rot} + \delta_{tr_{P_S}})$, where Δ_t is the time interval between arrival of the packet request at the S node, and the time when the packet is sent to the requesting I node. $\delta_{nw_{comm}}$ is a known when P_S and network bandwidth in the absence of blocking is fixed. The round trip time for the sequence of events represented by equation 4, $\delta_{io'}$, is measurable at run time. Hence, the only unknowns in equation 4 are δ_{rq} and $\delta_{nw_{bl}}$, from which the latter can be approximated (by neglecting δ_{rq}). Figure 3 shows the distribution of packet network blocking time, $\delta_{nw_{bl}}$, for the 5 policies. Bins of size 10 ms each (horizontal axis) were used

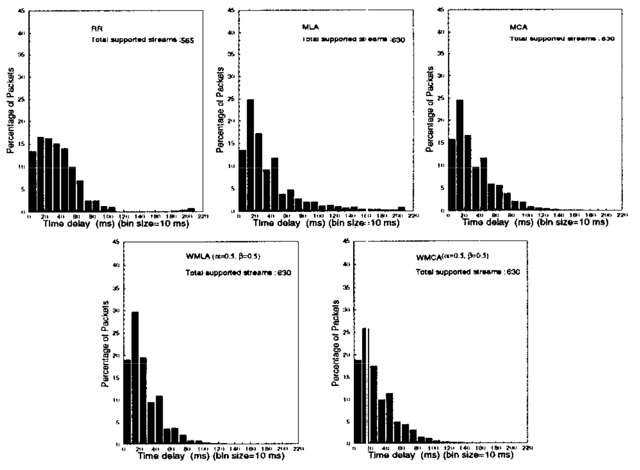

Figure 3. Frequency distribution of packet network blocking time

to count the distribution of network blocking time for each packet. The vertical axis shows the percentage of packets that fell in each bin. For real-time retrieval of data with a high quality of service (QOS), it is desirable that the variable components in equation 4 be bounded and of minimal value. The higher the cumulative percentage of packet blocking times falling in the leftmost bins, the better is the performance of the policy. Accordingly, the performance of the policies with respect to this metric (in ascending order) is RR, MCA, WMCA, WMLA and MLA. The frequency distribution of blocking times for the last four are not very different from each other, suggesting that the the number of supportable streams for (W)MLA and (W)MCA policies should be nearly the same. However, this is not the case, as shown below.

4.3. Stream Sourcing Capacity

We now compare the policies with respect to the more important metric of stream sourcing capacity.

Policy	Max. # of streams	improvement over RR
RR	565	-
MLA	633	12.0 %
MCA	647	14.5 %
WMLA ($\alpha = 0.5$, $\beta = 0.5$)	736	30.3 %
WMCA ($\alpha = 0.5$, $\beta = 0.5$)	757	34.0 %

Table 4. Maximum streams supported.

Table 4 shows the maximum number of streams that were supported by each policy, together with the percentage improvement over the RR policy. As expected, the RR policy performs the worst. Although it best balances the stream among the I nodes (minimum σ), it makes no effort to balance the load on the interconnection network. At the other end of the spectrum are the MLA and MCA policies : they try to reduce load on the interconnection network by minimizing link contention; however, they do not try to balance the load across the I nodes. In spite of this, they outperform RR by 12.0 % and 14.5 %, respectively. In between RR, on the one hand, and MLA and MCA, on the other, are the WMLA and WMCA policies that try to balance the load on both the network as well as the I nodes. This translates into superior performance over RR, MLA and MCA. The WMCA policy with $\alpha = 0.5$ gave the highest throughput of 757 streams among the 5 cases shown, corresponding to a 34.0 % improvement over RR. In summary, although the performance of (W)MLA is similar to that of (W)MCA as far as network blocking time is concerned, the load imbalance on the I nodes is much higher for the former than for the latter (table 3). This explains why (W)MCA consistently outperforms (W)MCA.

5. Conclusions

In this paper we developed five policies for assigning requests to the interface nodes in a high-performance multimedia server. MLA, MCA, WMLA and WMCA each outperformed RR in terms of number of streams. RR best balances inter I node load, closely followed by MLA and WMLA. Although MCA and WMCA give worst performance on this count, WMCA with proper choice of weights gave highest throughput. The (W)MCA policy is a global one, as it takes into account the load on a link due to the existing traffic. (W)MLA, on the other hand is a local optimization that is oblivious of the load imposed by other nodes. This explains why WMCA gave the best throughput. Figure 4 shows the effect of varying the weight values on the maximum number of supported streams for the five polcies. For the parameters and data access pattern considered in this paper, $\alpha = 0.75$, $\beta = 0.25$ gave the best performance for both WMLA and WMCA policies. The optimum values to assign to the weights

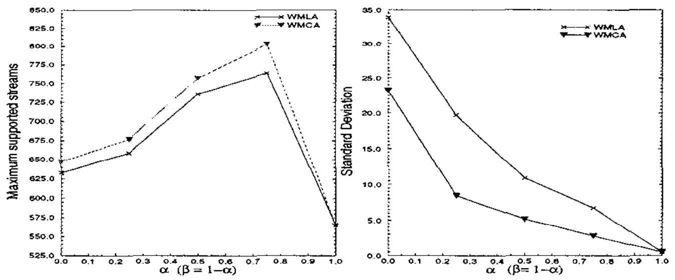

Figure 4. Effect of changing weight values.

is an implementation-dependent problem that depends on the network topology, routing strategy and the maximum streams that an I node can source. However, changing the ratio values of α and β in one direction makes the assignment criterion tend to RR ($\alpha = 1$, $\beta = 0$), while changing the ratio in the opposite direction will make it tend to ML(C)A ($\alpha = 0$, $\beta = 1$). We have shown that values in between give better performance than these extremes. This is so because such values try to balance both the load on the I nodes and the load on the interconnection network, unlike the extreme cases, which balance one or the other.

References

[1] A. Reddy and J. Wyllie. I/O issues in a multimedia system. *IEEE Computer*, vol. 27, No. 3, pp. 69-74, March 1994.

[2] P. V. Rangan and H. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering* , Vol. 5, No. 6, August 1993.

[3] S. Ghandeharizadeh and L. Ramos. Continuous retrieval of multimedia data using parallelism. *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, No. 4, August 1993.

[4] D. Jadav and A. Choudhary. Design issues in high performance media-on-demand servers. *IEEE Parallel and Distributed Technology Systems and Applications*, Summer 1995.

[5] D. Jadav, A. Choudhary, P. Bruce Berra and C. Srinilta. An evaluation of design tradeoffs in a high performance media-on-demand server. *CASE Center Technical Report # 9502*, CASE Center at Syracuse University, February 1995).

[6] A. Dan and D. Sitaram. An Online Video Placement Policy based on Banwidth to Space Ratio. *Proceedings of the ACM 1995 Intl. Conference on the Management of Data*, pp. 376-385, May 1995.

[7] L. Ni and P. McKinley. A survey of wormhole techniques in direct networks. *IEEE Computer*, vol. 26, No. 2, pp. 62-76, February 1993.

[8] A. Dan, D. Sitaram and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. *Proceedings of ACM Multimedia '94*, pp. 15-23, 1994.