Techniques for Increasing the Stream Capacity of A High-Performance Multimedia Server

Divyesh Jadav, *Member*, *IEEE*, Alok N. Choudhary, *Member*, *IEEE*, and P. Bruce Berra, *Fellow*, *IEEE*

Abstract—High-performance servers and high-speed networks will form the backbone of the infrastructure required for distributed multimedia information systems. 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, it is essential that the load be balanced among each of the server's components viz., the disks, the interconnection network, and the scheduler. Many algorithms for maximizing retrieval capacity from the storage system have been proposed in the literature. 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—round-robin (RR), minimum link allocation (MLA), minimum contention allocation (MCA), weighted minimum link allocation (WMLA), and weighted minimum contention allocation (WMCA)—are developed. The performance of these policies on a server model developed earlier is presented. We also consider the issue of file replication, and develop two schemes for storing the replicas, the Parent Group Based Round-Robin Placement (PGBRRP) scheme, and the Group Wide Round-Robin Placement (GWRRP) scheme. The performance of the request assignment policies in the presence of file replication is presented.

Index Terms—Parallel input/output, media-on-demand server, dynamic resource allocation, real-time data retrieval, file replication.

1 Introduction

1.1 Motivation

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. However, a number of problems need to be solved for developing cost-effective systems. These problems arise because contemporary computers have been designed to process predominantly unimedia (text) type of data, which differs greatly from multimedia data. The most important differences are the diverse data sizes of multimedia data types, and the need to provide real-time guarantees for playback of multimedia data such as video and audio. One of the most pervasive applications of the coming interactive age is *media-on-demand* which refers to the possibility of a consumer interactively retrieving multimedia data (e.g., movies, songs, images, digitized encyclopedias, etc.) over high-speed networks from geographically distributed media servers, and viewing the data at home or work.

A 90-minute long MPEG-1 [2] digitized movie requires nearly 1 gigabyte of storage. It is anticipated that a distributed media-on-demand (MOD) system will be built in a hierarchical manner, with high-capacity metropolitan servers interconnected by a high-speed network, and less powerful neighborhood servers connected to a metropolitan server (Fig. 1). The clients interact with the neigbourhood server to store and retrieve data. This hierarchy of servers is similar to the memory hierarchy in a computer system. In the latter, the hierarchy consists of processor register memory, cache memory, main memory, secondary storage, and finally, tertiary storage. When the processor issues a request for data, the penalty (in terms of time) 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 MOD system as shown in Fig. 1, the analogous hierarchy consists of client terminal (set-top box), neighborhood server, metropolitan server, and archival server. A similar relationship exists in this case with regard to the time delay for data retrieval by a client.

On account of the stringent 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 very important to optimize 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.

1.2 Related Work

Researchers have proposed various approaches for the storage and retrieval of multimedia data. Anderson et al. [1] have proposed file system design techniques for providing hard performance guarantees. Reddy and Wyllie [3], [4]

• *D. Jadav is with the IBM Almaden Research Center, 650 Harry Rd., San Jose, CA 95120. E-mail: divyesh@almaden.ibm.com.*
• *A.N. Choudhary is with the Department of Electrical and Computer Engineering, Northwestern University, Technological Institute, Evanston, IL 60208. E-mail: choudhar@ece.nwu.edu.*
• *P.B. Berra is with the Department of Computer Science and Engineering, Wright State University, Dayton, OH 45435. E-mail: berra@cs.wright.edu.*

Manuscript received 31 May 1996; revised 13 Sept. 1997.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104320.

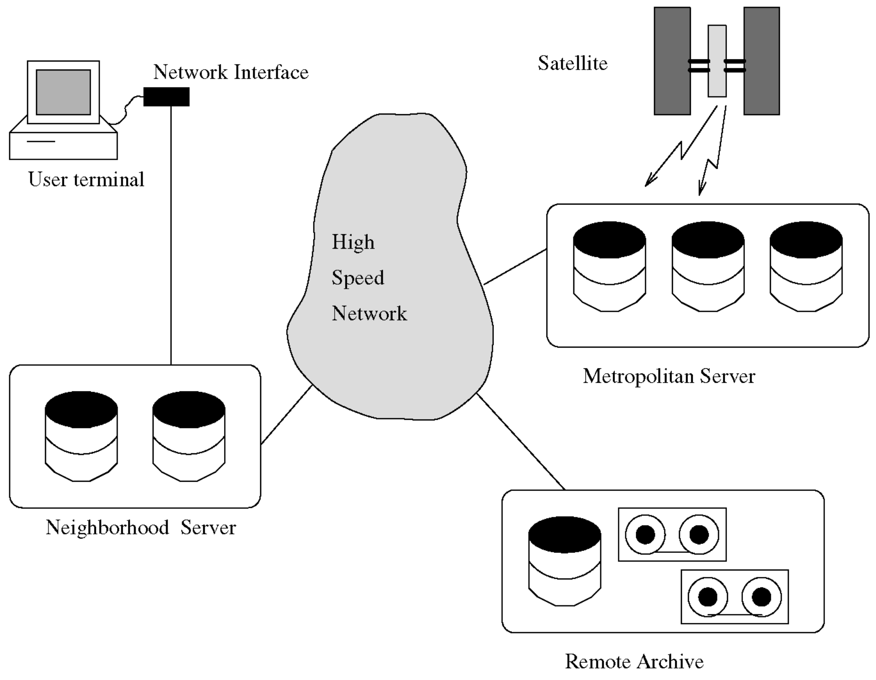


Fig. 1. Block diagram of a hierarchical media-on-demand (MOD) system.

have proposed a disk arm scheduling approach for multimedia data, and characterized the disk-level trade-offs in a multimedia server. Rangan and Vin [5] and Rangan et al. [6] have proposed a model based on constrained block allocation, which is basically noncontiguous disk allocation in which the time taken to retrieve successive stream blocks does not exceed the the playback duration of a stream block. Storing a stream on a single disk restricts the retrieval bandwidth to the data transfer bandwidth of the disk. Ghandeharizadeh and Ramos [7] get around this problem by 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 is similar to this model in using data striping, round-robin distribution of successive stream fragments and contiguous allocation within a given fragment. Vin et al. [8] categorize real time clients into two classes, those that require hard- and soft-performance guarantees, respectively. For the latter class, the worst case assumptions made in admitting new users are relaxed based on the observed server load to increase the number of users that can be supported.

Issues in designing MOD servers are discussed in [9], [10]. Most previous work has concentrated on minimizing disk rotational and seek overheads in retrieving data. It is only now that the issues in dealing with higher level aspects of MOD servers are being addressed. Various striping trade-offs have been studied in [7], [11], [12], [13], [14]. Freedmann and De Witt [15] and Ozden et al. [16] studied efficient memory allocation and utilization techniques to maximize the number of supported users. Freedmann and De Witt [15] also studied cost trade-offs and scalability issues in high-performance MOD servers. Techniques for improving reliability and availability of the storage subsystem are studied in [17]. *Interserver* [18], [10] and *intraserver* [19] caching techniques for tuning server performance based on user access patterns have also been proposed. However, little work has been reported on instrumentation of I/O traffic in a MOD server. We have performed a componentwise instrumentation of the delays in a MOD [20], where we showed that variable delays become performance bottlenecks at high loads.

1.3 Our Research Contributions

A substantial body of research has been directed towards trying to match the disparity between available and required data retrieval bandwidth from secondary storage, for the most part in the form of magnetic disks, in a MOD server. Techniques for balancing the load on the storage devices of a MOD server were developed in [21]. 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. Techniques for doing so are the subject of this paper.

From an application-level perspective, the most important metric for evaluating an on-demand server is the maximum number of streams that it can source simultaneously. From a data storage organization perspective, various striping trade-offs exist for achieving parallelism of data retrieval. After briefly examining these trade-offs, we investigate how *file replication* can improve the performance of one of the striping alternatives with respect to the maximum number of streams that can be supported. Lastly, from an operating systems perspective, it is essential to balance the workload on the subsystems of a server, viz., the processor nodes, the interconnection network, and the storage subsystem.

In this paper, we design and evaluate various policies for assigning stream requests to server nodes, so as to minimize traffic on the interconnection network. Five strategies

1) round-robin (RR),
2) minimum link allocation (MLA),
3) minimum contention allocation (MCA),
4) weighted minimum link allocation (WMLA), and
5) weighted minimum contention allocation (WMCA)

are developed. These policies differ in that some of them attempt to balance the load on the nodes that source data to the outside world (RR), some attempt to minimize load on the interconnection network (MLA, MCA), while the others attempt to do both. We have developed and implemented a logical model for a MOD server [19], [20]. Performance results of the five policies under this model implemented on the Intel Paragon parallel computer are presented. We also consider the subject of file replication to alleviate the workload imbalance on the storage subsystem due to skewed data access patterns. Two schemes—the Parent Group Based Round-Robin Placement (PGBRRP) scheme and the Group Wide Round-Robin Placement (GWRRP) scheme—for storing the replicas of a file are developed. Performance results of the stream assignment strategies when files are replicated are presented.

The rest of this paper is organized as follows: Section 2 presents a brief overview of the server model. In Section 3, we explain the data organization, access and scheduling policies. In Section 4, we develop the five allocation strategies. We present and analyze performance results in Section 5. Two schemes for storing replicated files, and their performance evaluation, are presented in Section 6. In Section 7, we discuss future work, and Section 8 summarizes the paper.

2 Server 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. In this model, we assume that:

- The server is connected to clients by a high-speed wide area network, for example, using ATM switches and a fiber optic network. The wide area network delivers data to clients reliably and at the required bandwidth, consequently, it is eliminated from further discussion in the rest of this paper.
- Clients have *hard* deadlines i.e., they cannot tolerate jitter in delivered data. Although client display stations may have a few megabytes of main memory and tens of megabytes of secondary memory, the storage space is not sufficient to store most multimedia files (which are of the order of a few gigabytes in size) in their entirety. Consequently, the server must retrieve and supply data to clients at almost the same rate as its consumption by clients.
- The data are stored at the server in compressed digital form. The decompression of the data is done at the remote client's *multimedia terminal.*

Fig. 2 shows the data model of the MOD server. It consists of multiple nodes interconnected by a network. Each node is a computer in its own right, with a CPU, RAM and secondary storage. In addition, each node has an interface with the interconnection network. It consists of two types of nodes, **interface (I) nodes** and **storage (S) nodes** connected by a high-speed interconnection network. In addition, there

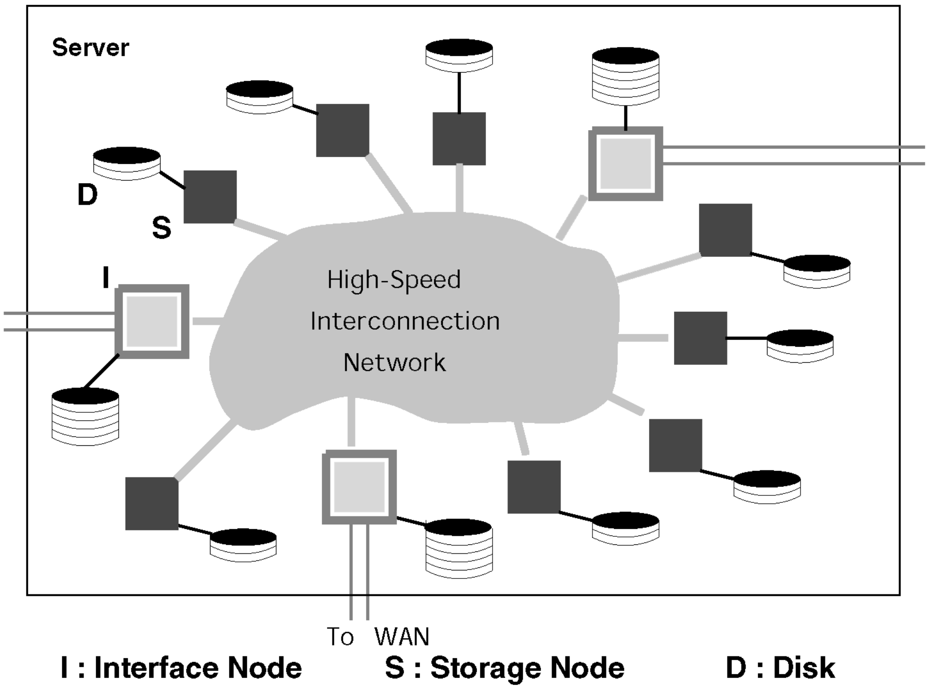
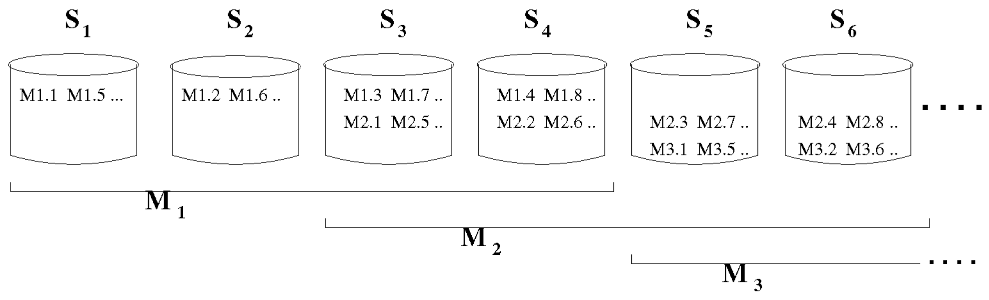


Fig. 2. Logical architecture of server.



S_1
S_2
S_3
S_4
S_5
S_6
M1.1 M1.5 ...
M1.2 M1.6 ..
M1.3 M1.7 .. M2.1 M2.5 ..
M1.4 M1.8 .. M2.2 M2.6 ..
M2.3 M2.7 .. M3.1 M3.5 ..
M2.4 M2.8 .. M3.2 M3.6 ..
M_1
M_2
M_3

Fig. 3. Degree of overlap (*DoO*). This figure shows three objects, *M*1, *M*2, and *M*3 striped across six storage nodes, with a *DoO* of 2.

is a third type of node, called the **object manager (O) node** (not shown in the picture). The object manager receives all incoming requests for media objects. It has knowledge of which **storage** nodes an object resides on and the workload of the **interface** nodes. Based on this knowledge, it *delegates the responsibility* of serving a request to one of the interface nodes. The interface nodes are responsible for scheduling and serving stream requests that have been accepted. Their main function is to request the striped data from the storage nodes, order the packets received from the storage nodes, and send the packets over the high-speed wide area network to the clients. An interface node can also use its local secondary storage to source frequently accessed data objects. Storage nodes actually *store* multimedia data on their secondary storage devices in a striped fashion, and *retrieve* and transmit the data to an interface node when requested to do so.

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. In this architecture, each node has a point-to-point, or direct, connection to some number of other nodes, called *neighboring nodes*. Direct networks have become a popular architecture for constructing parallel computers because they scale well [23] with the number of nodes. Examples of popular direct networks are *n*-dimensional meshes and *k*-ary *n*-cubes; these are popular because their regular topologies simplify routing.

3 Data Access and Scheduling

3.1 Data Organization

As mentioned earlier, the data is compressed and striped across the storage nodes in a round-robin fashion. The number of nodes across which an object[1] 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*. Fig. 3 illustrates these concepts.

1. The term *object* refers to the stored multimedia data, when the data is being retrieved and sourced to the client, it is called a *stream*.

With reference to the figure, object M_1 consists of four (SF = 4) subobjects stored on storage nodes S_1 through S_4. The subobject on node S_1 consists of stripe fragments *M*1.1, *M*1.5 etc.; that on node S_2 consists of stripe fragments *M*1.2, *M*1.6, etc.; that on node S_3 consists of stripe fragments *M*1.3, *M*1.7, etc.; and that on node S_4 consists of stripe fragments *M*1.4, *M*1.8, etc. Note that each storage node may have a single high-performance disk, or an array of slower, but cheaper disks. The point to note is that a storage node represents a *virtual disk* to an interface node. Since the stripe fragments on any given storage node's disk are not consecutive fragments, it is not necessary to store them contiguously. Disk scheduling algorithms to optimize retrieval from the disk surface have been proposed [3], [24], [15], and can be used in our model. We are concerned with harnessing the parallelism provided by striped storage and balancing the load across the server subsystems.

An important factor that affects retrieval time is the *placement* of each stream's media data relative to that of other streams, i.e., the manner in which the data is partitioned across multiple disks has a critical effect on the retrieval time seen by any one stream; this is so because some or all of the data of other streams that are being served may overlap with the data of the observed stream on the storage nodes. This overlap results in queueing delays for the observed stream's retrievals from the storage nodes. For understanding the data partitioning strategy used we define a term called the **degree of overlap (DoO)**. This is a positive integer, $0 \le DoO \le SF$ (*SF* is the stripe factor) and denotes the distance between the *i*th subobject of object *j* and the *i*th subobjects of object $j + 1$, in terms of the number of storage nodes. The concept of *DoO* is illustrated in Fig. 3.

Another factor that affects retrieval time is the stripe factor. We differentiate between *wide striping*, in which a media object is striped across all the storage nodes of a server, and *narrow striping*, in which an object is striped across a fraction of the total storage nodes in a server. Each approach has its advantages and disadvantages. In the latter case, a striping group containing a frequently accessed object can become a bottleneck for the server in the absence of object replication. For example, in a video-on-demand case, it is but natural that some videos will be more frequently accessed than others: Newly released videos are likely to be more frequently accessed than older videos. Wide striping avoids the formation of such bottlenecks by striping an object across all the storage nodes, which has the effect of balancing the load across all storage nodes even for skewed access patterns. However, wide striping

TABLE 1
THE PARAMETERS USED IN THIS PAPER

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	-
DoO	Degree of Overlap	-

also has some drawbacks. In wide striping, there is only one striping group; this complicates system reconfiguration. Since each storage node has some data for all objects stored in the server, most of the chunks of all the objects may have to be moved. This may lead to an undesirable reconfiguration load on *all* storage nodes. In contrast, in narrow striping, only the storage nodes in the striping group where the object is stored incur the penalty. Secondly, the larger the size of a striping group (i.e., the wider the striping), the lower is the reliability, and also the availability, of the entire system [17], [14] in the event of a single disk failure.

3.2 Parameters Used and Scheduling Constraints

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 secs. 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

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

where δ_{rg} 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.

Thus, if the playout of an *I* node buffer is started at time *t*, then the *latest* time by which the requests for the next set of stripe fragments must be issued to the *S* nodes is:

$$t_{max} = t + \delta_S - \delta_{io}\,. \tag{3}$$

Note that (2) uses average seek and rotational latencies for disk accesses. Since these latencies are variable, there will be boundary conditions when the time to retrieve P_S bytes is much more (less) than the average value. However, the effect of this deviation from the average value on the overall service time depends on the relative magnitudes of the other components of the service time. Our approach is based on the fact that when the granularity of data read from disk is large, the effect of random disk seek and rotational overheads is reduced. While it is true that doing so increases buffering requirements, contemporary computers have large main memories, and using such machines as server nodes is well worth the gain obtained in making disk service time more predictable. Of course, if some clients require strict performance guarantees, then one can categorize users into those requiring hard and soft deadlines, and use the maximum values of the disk overheads for admitting users of the latter kind.

4 STREAM ASSIGNMENT POLICIES

The motivation for these policies is that in a given server configuration, only a finite number of nodes would be connected to the high-speed wide area network; these nodes are the interface nodes. 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[2] on the secondary storage subsystem [12]. For example, a movie on demand server with a secondary storage capacity of 100 gigabytes would be able to store about 100 MPEG-1 encoded

2. The assumption is that the penalty of retrieving data from archival storage is too high for the high-bandwidth requirements of multimedia data.

movies, but have a catalog of 400 titles to maximize clientele. However, the number of movies stored on secondary storage would be a slowly changing set, changing every two or three days or maybe weekly. Consequently, over a short period of time (say one day) the data distribution on secondary storage would be more or less static. Given the fact that the position of the I nodes and the storage pattern of data on S nodes is fixed.[3] 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. We now present and evaluate five schemes that differ in the workload they impose on the interconnection network when assigning requests to interface nodes.

The communication time over the network is the sum of two factors:

1) the network latency in the absence of blocking, and
2) the blocking time due to link contention in the interconnection network i.e.,

$$\delta_{nw_{P_S}} = \delta_{nw_{comm}} + \delta_{nw_{bl}}. \tag{4}$$

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 queuing delay at the S nodes. If

$$\delta_{S_Q}$$

denotes this queueing delay, (2) requires to be modified to:

$$\delta'_{io} = \delta_{rg} + \delta_{seek} + \delta_{rot} + \delta_{tr_{P_S}} + \left(\delta_{nw_{comm}} + \delta_{nw_{bl}}\right) + \delta_{S_Q}\ . \tag{5}$$

We have studied the effect of the variable delays

$$\delta_{nw_{bl}}$$

and

$$\delta_{S_Q}$$

on total retrieval time at various workloads [20]. While most admission control policies for accepting stream requests in a MOD server have concentrated on storage device parameters such as seek and rotational latencies (which are bounded), a dynamic admission control policy was proposed in [20] that takes these variable (and potentially unbounded) delays into account. At heavy workloads, these blocking delays become the limiting factors on stream sourcing 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 simultaneously.

3. Redistribution of data on the fly is expensive in terms of operating system overhead on account of disk I/O and the large size of typical multimedia objects.

4.1 Round-Robin (RR) Assignment Policy

This is the simplest policy and will be used as the baseline policy for comparison purposes. 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 $(k + 1) \bmod n$. The greatest merit of this policy is that it is simple, requiring $O(1)$ time to execute. It also balances the workload in terms of number of streams served per I node equally among the I nodes: The maximum difference among the number of streams that any two I nodes server at a given time is at most 1. However, this policy does not balance or minimize the load imposed on the interconnection network.

4.2 Minimum Link Assignment (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

$$l_{I_iS_j}$$

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:

$$CA_{MLA}(I_i) = \sum_{j=1}^{SF} l_{I_iS_j}\ . \tag{6}$$

This policy will assign a request to interface node p, where

$$p = i : (\min(CA_{MLA}(I_i)) \qquad i = 1, 2, ..., n) \tag{7}$$

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. Since (6) has to be evaluated for all n I nodes in the server, the worst case execution time of this algorithm is $O(n * SF * d)$ where d is the maximum time to determine the number of links between any two nodes in the server.

This policy tries to minimize the number of streams using a given link. However, it does not take into account the pre-existing load on a link. Nor does it balance the total stream load among all the interface nodes.

4.3 Minimum Contention Assignment (MCA) Policy

In this policy, state information is maintained about the usage of each link by all objects being retrieved. Specifically, whenever a new request 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

$$C_{I_iS_j}[k]$$

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:

$$CA_{MCA}(I_i) = \sum_{j=1}^{SF} \sum_{k=1}^{l_{I_iS_j}} l_{I_iS_j}[k]\ . \tag{8}$$

This policy will assign a request to interface node p, where

$$p = i : (\min(CA_{MCA}(I_i)) \qquad i = 1, 2, ..., n) . \tag{9}$$

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 $\left(k = 1 \text{ to } l_{I_p S_j}\right)$
        $C_{I_p S_j}[k] = C_{I_p S_j}[k] + ld$          (10)
```

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 (P_S) and the playback rate of the stream R_{pl}.

The basic premise behind this policy is that the load should be distributed evenly over the interconnection network. Otherwise, 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. This policy is more computationally expensive than the other two, as it requires accessing the load data structure for each link. Its worst case running time is $O(n * SF * maxl)$, where $maxl$ is the maximum number of links between two nodes in the network.

Note that when a stream relinquishes resources due to termination or pause, network load must also be updated. This is the opposite of (10), i.e.,

```
for ($j = 1$ to $SF$)
    for $\left(k = 1 \text{ to } l_{I_p S_j}\right)$
        $C_{I_p S_j}[k] = C_{I_p S_j}[k] - ld$ .          (11)
```

4.4 Weighted Minimum Link Assignment (WMLA) and Weighted Minimum Contention Assignment (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 is:

$$CA_{WMLA}(I_i)' = \alpha * M_{I_i} + \beta * CA_{MLA}(I_i) \tag{12}$$

and under WMCA is:

$$CA_{WMCA}(I_i)' = \alpha * M_{I_i} + \beta * CA_{MCA}(I_i) \tag{13}$$

where α and β are fractions that sum to 1, and $CA_{MLA}(I_i)$ and $CA_{MCA}(I_i)$ are given by (6) and (8), respectively. However, in each of (12) and (13), the two quantities being weighted and combined can have different magnitudes. Hence, it is necessary to normalize

$$M_{I_i}$$

and $CA_{xxx}(I_i)$ prior to multiplication by α and β, respectively. This is done as follows. We normalize these terms so that the value of each of them lies between 0 and 1, inclusive. Let M be the *total* number of streams being served. Then,

$$CA_{WMLA}(I_i) = \alpha * \frac{M_{I_i}}{M} + \beta * \frac{CA_{MLA}(I_i)}{l_{max}} \tag{14}$$

where l_{max} is the *maximum* number of links between a given node and SF other nodes with which it has to communicate. The normalized cost for WMCA is:

$$CA_{WMCA}(I_i) = \alpha * \frac{M_{I_i}}{M} + \beta * \frac{CA_{MCA}(I_i)}{ld_{agg}} \tag{15}$$

where ld_{agg} is the *aggregate* traffic load on the interconnection network; i.e., the sum of the load on each link of the network. This value can be easily maintained (one addition operation) whenever (10) is invoked.

The criterion for selecting a candidate I node is similar to that for the respective unweighted cases [(7) and (9), 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 α = 1, β = 0 is equivalent to RR, while WML(C)A with α = 0, β = 1 is equivalent to ML(C)A.

5 Performance Evaluation

We have implemented our logical server model on the Intel Paragon parallel computer. The Intel Paragon [27] is a mesh-based architecture with Intel i860XP microprocessors. Interprocessor communication is done using *wormhole routing* [23]. The most important metric of an interconnect for multimedia data is its communication latency, which is the sum of three factors: start-up latency, network latency, and blocking time. The first two are static features for a given system in that the sum of their values represents the latency of packets sent in the absence of other network traffic and transient system activities. Blocking time includes all possible delays encountered during the lifetime of a packet, such as those due to link contention. An important reason for the growing popularity of wormhole routing as a switching technique in interconnection networks is that when it is used, the network latency is almost independent of the path length when there is no link contention and the packet size is large. Thus, minimizing link contention ensures that the deleterious effects of blocking time are kept in check, which, as explained above, is crucial to providing real-time communication guarantees. By its very nature, wormhole routing is highly susceptible to deadlock conditions. Various routing algorithms have been proposed and used to provide deadlock-free wormhole routing. We use *deterministic XY routing* in which packets are first sent along the X direction, and then along the Y dimension. We benchmarked the interconnection network of the Paragon for determining its bandwidth. In the absence of any other traffic, a round trip send-receive achieved an average link bandwidth of 17.6 Megabytes/sec.

The data access pattern is assumed to follow a Zipfian distribution (Fig. 4) with parameter 0.271 [26]. The total number of objects in the server is assumed to be 100. Due to storage space and availability of real-world data limitations, the disk access part was simulated. The disk access time was simulated by elapsing the system timer on each storage node. Disk retrieval was simulated by assuming that the stripe fragments are stored on the disk using a random placement model [22]. We have assumed gigabytes of disk space per node, and a disk data transfer rate of 10 Mbytes/sec, with two disks per storage node. Currently available magnetic disks have data transfer rates of a few megabytes per second. In general, for higher data transfer rate and rotational speed of the disk, the higher the disk cost. Thus, it might be better to have an array of cheaper but slower disks than a single fast disk. For example, one could use an array of four disks to achieve a 10 Mbytes/sec data transfer rate. In practice, the exact type and configuration of disks to use is an implementation decision. We used a playback rate (R_{pl}) equal to the MPEG-1 rate of 1.5 Mbits/sec. Table 2 shows the values of the parameters defined in Table 1 that we used for our simulation. It should be noted that except for the simulation of the disk access, the rest of the server operations were implemented including the scheduler and data transfer over the interconnection network.

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 secs. There is an initialization transient until all interface nodes have started sourcing streams. When this occurs, the software begins gathering performance data. 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. Disk retrieval was simulated by assuming that the stripe fragments are stored on the disk using a random placement model [22]. For the purposes of this paper, we assume a *DoO* of 0, nine stripe groups and a stripe factor of 4.[4] We used a 8 × 6 mesh resulting in a total of 48 nodes used in the experiment. Fig. 5 shows the distribution of I nodes and S nodes. Node 20 is the object manager (O), nodes marked 'I' are interface nodes, and the other nodes are storage nodes.[5] The value of the parameter *ld* for the MCA policy (Subsection 4.3) 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 during startup. We carried out experiments for all five assignment policies. The same data distribution and request pattern were used for each experiment, in order to permit comparison of the results. The most important metric for comparison is the maximum number of streams that can be supported. We have used deterministic (as opposed to statistical) deadlines, i.e., deadlines are *hard* and *clients cannot tolerate missed packets*. The other metrics we used were the degree to which

4. Numerous trade-offs are possible with respect to the data partitioning strategy, which are well reported in [7], [11]. However, these are not the subject of this paper.

5. We have performed experiments for other distributions of S and I nodes. The results in those cases were similar to the ones presented below, and have not been included in this paper on account of space limitations.

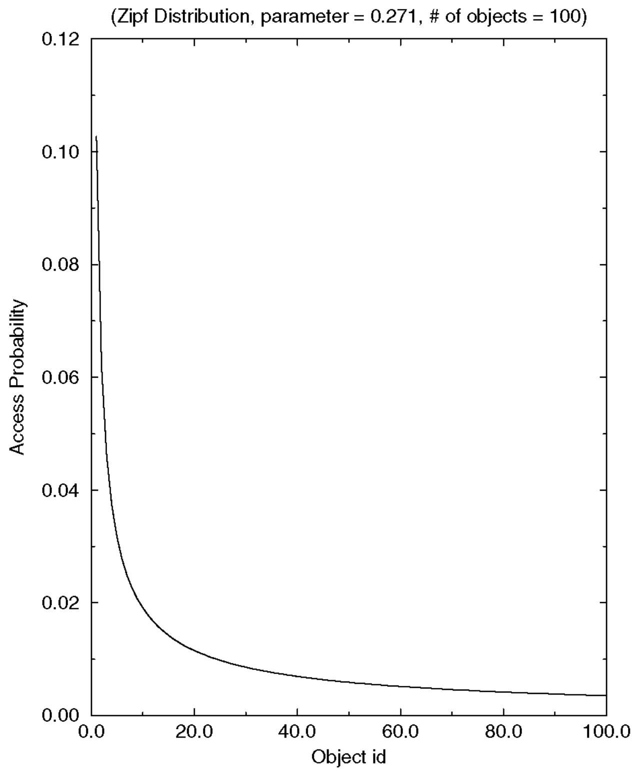


Fig. 4. Zipfian distribution.

TABLE 2
THE PARAMETER VALUES USED FOR THE EXPERIMENTS

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
Degree of overlap DoO	0
Stripe Factor (S)	4
Num. of Interface nodes	11
Num. of Storage nodes	36
Num. of media objects	100
Evaluation machine	Intel Paragon

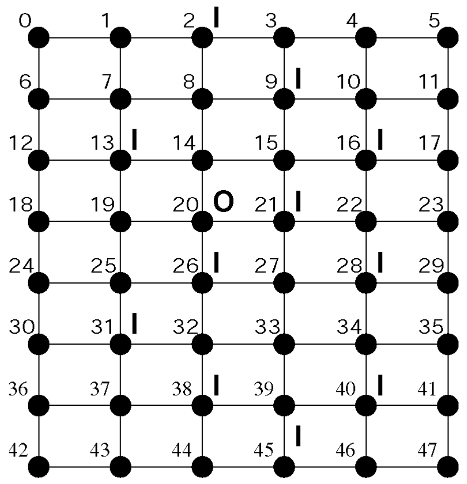


Fig. 5. Distribution of I, S, and O nodes in the evaluation configuration.

load is balanced across the interface nodes in terms of number of requests, and the frequency distribution of packet blocking time.

5.1 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 number of total streams served by the server for each policy except for RR. Fig. 6 illustrates these values for each I node in Fig. 5 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 reason for the different stream loads used for RR and the other four policies is as follows: We are primarily interested in server performance under stress conditions (i.e., high-stream loads). Except for the RR policy, each of the other four policies supported at least 630 streams, but the maximum number of streams supported by RR was only 565. (Note that I node 1 in the graph corresponds to node 2 in the mesh, I node 2 corresponds to node 9, etc.) We first compare the RR, MLA, and 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 shows this value for the five policies. 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.

Now, consider the WMLA and WMCA policies. The graphs in Fig. 6 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 less 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 for this metric.

5.2 Comparison of Network Blocking Time

We now compare the performance of the policies with respect to the network blocking time. With reference to (5), 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 runtime. Given a disk and a value of P_S,

$$\delta_{tr_{P_S}}$$

can be computed.

$$\delta_{S_Q}$$

is given by

$$\delta_{S_Q} = \Delta_t - \left(\delta_{seek} + \delta_{rot} + \delta_{tr_{P_S}}\right),$$

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 (5),

$$\delta_{io'},$$

is measurable at runtime. Hence, the only unknowns in (5) are δ_{rq} and

$$\delta_{nw_{bl}},$$

from which the latter can be approximated.[6] Fig. 7 shows the distribution of packet network blocking time,

$$\delta_{nw_{bl}}$$

for the five policies.

Bins of size 10 msec each were used to count the distribution of network blocking time for each packet. The horizontal axis depicts these bins i.e., 0-10 msec, 11-20 msec, 21-30 msec, etc. 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 (5) be bounded and of minimal value. Hence, the higher the cumulative percentage of packet blocking times falling in the leftmost bins, the better is the performance of the policy. Keeping this in mind, it can be seen that the performance of the policies with respect to this metric (in ascending order) is RR, MCA, WMCA, WMLA, and MLA. Note that the frequency distribution of blocking times for the last four are not very different from each other. This would suggest that the performance of the (W)MLA policy would be similar to that of the (W)MCA policy in terms of the maximum number of streams supported. However, this is not the case, as shown below.

6. The time for the request from the I node to reach the S node, δ_{rq}, can be neglected. This is a packet of a few bytes in length; moreover, it is in the opposite direction of the dataflow. Hence, its value is very small compared to the other terms.

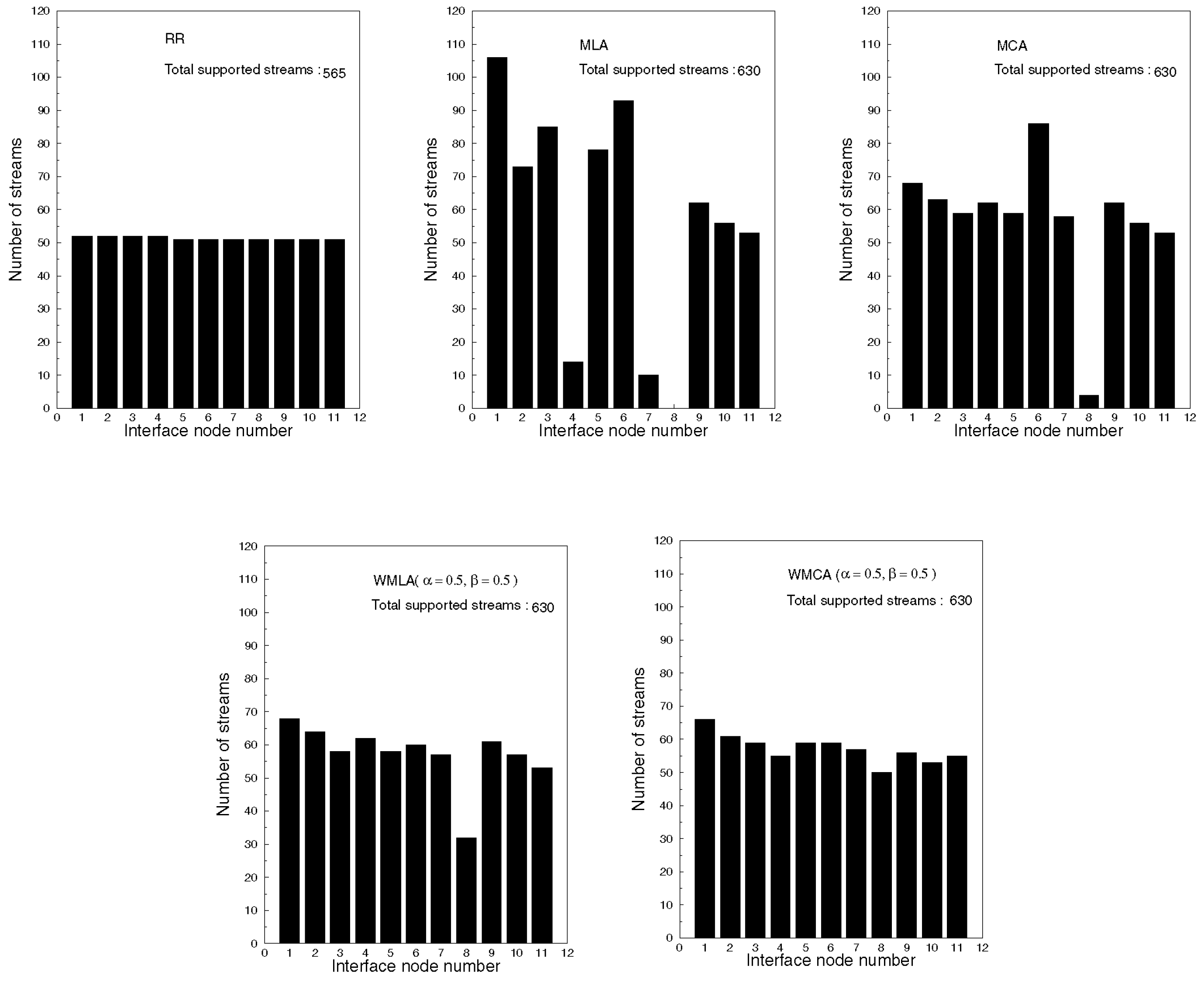


Fig. 6. Comparison of request assignment of RR, MLA, MCA, WMLA, and WMCA policies.

TABLE 3
STANDARD DEVIATION OF STREAM LOAD PER I NODE

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 ($\alpha = 0.5, \beta = 0.5$)	57.27	8.85
WMCA ($\alpha = 0.5, \beta = 0.5$)	57.27	4.07

5.3 Comparison of Stream Sourcing Capacity

We now compare the policies with respect to the more important metric of stream sourcing capacity. Table 4 shows the maximum number of streams that were supported by each policy, without missing any packet deadlines, 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 percent and 14.5 percent, respectively. 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 α = 0.5 gave the highest throughput of 757 streams

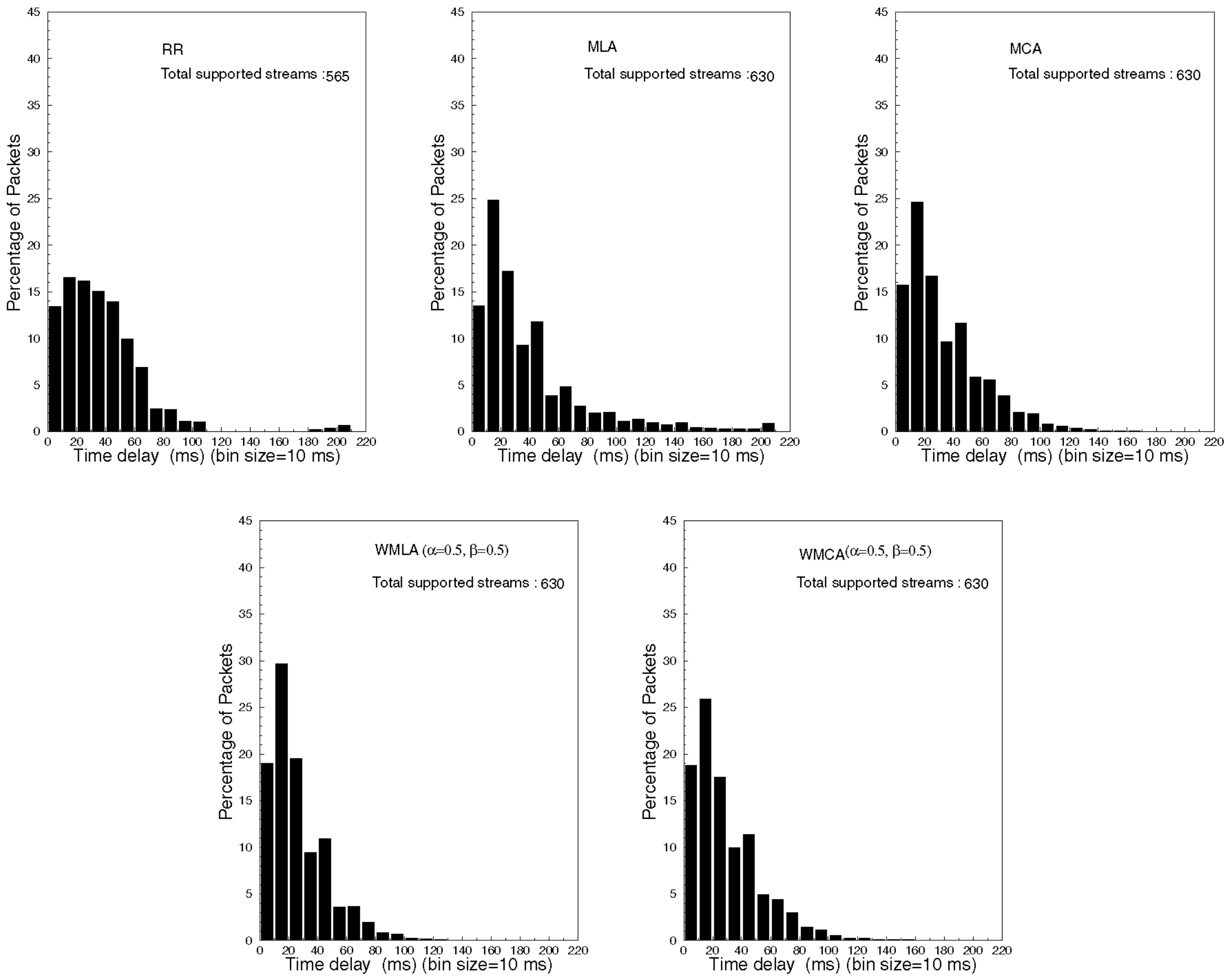


Fig. 7. Frequency distribution of packet network blocking time for RR, MLA, MCA, WMLA, and WMCA policies.

TABLE 4
THE MAXIMUM NUMBER OF STREAMS SUPPORTED FOR EACH EXPERIMENT

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 %

among the five cases shown, corresponding to a 34.0 percent 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 in the maximum number of supported streams.

5.4 Overhead of (W)MLA and (W)MCA Policies

An advantage of the RR policy is that its use incurs almost zero overhead. Execution of the (W)MLA and (W)MCA policies incurs overhead due to computations and comparisons. For the highest workload supported by WMLA (736 streams), the *total* (i.e., total for all streams) time overhead was 9.3 secs, while for WMCA (757 streams), it was 29.7 secs.[7] The overhead for either of (W)MLA or (W)MCA *per stream* is of the order of a few millisecs, which is a negligible fraction of the duration of a two-hour movie. Moreover, the algorithm needs to be executed only at the

7. The difference between overhead of MLA and WMLA, and MCA and WMCA, was negligible (of the order of a few millisecs). This is so because MLA and MCA differ from WMLA and WMCA by only 2 floating point multiplications and an addition.

time of accepting a request (and possibly when a paused stream is restarted. The (W)MCA algorithm also needs to be run when a stream pauses or departs). Thus not only are these policies effective in increasing server throughput, they are also efficient in terms of overhead incurred.

6 EFFECT OF REPLICATION

6.1 Motivation

In the experiments described thus far, the subject of object replication has not been addressed. We now develop two schemes for replicating frequently accessed files, and present the performance of these two schemes.

As noted in Table 2, the database we used for the experiments consisted of 100 files, with each file stored in one of nine striping groups. Each file is assumed to have an object id, i. Similarly, each striping group also has a group id, k. The object ids are positive integers starting from 1 (Fig. 4), and so are the group ids. If N is the number of striping groups, a file with id i is stored in group k, where

$$k = (i - 1) \bmod (N) + 1 \tag{16}$$

Thus, files with object ids 1, 10, 19 etc. will be stored in group 1, files with object ids 2, 11, 20, etc. will be stored in group 2, and so on. In the absence of file replication and in the presence of a skewed data access pattern, a striping group containing a frequently accessed file can become a hot-spot for the storage subsystem and cause the performance of the server to degrade. One way to avoid such hot-spots from developing is to replicate frequently accessed files in *different* striping groups, so that accesses are spread over multiple striping groups instead of being localized to one group. Object replication is expensive in terms of storage space (due to the typically large size of multimedia files). Moreover, when files are replicated, the issue arises as to which replica of a file is to be chosen for serving a request for the file. This may require multiple executions of the the assignment policies developed in this paper, which may be computationally expensive. We develop two heuristic replication schemes that not only avoid multiple executions of the algorithms to select an interface node for serving a request, but also improve server performance, in terms of the number of streams supported.

Two important design decisions that must be made when replication is considered are:

1) *Which* files should be replicated, and
2) *How many* replicas should be made of a file?

Observe that in a skewed access distribution such as in Fig. 4 (which is a distribution observed in practice by [26]), a few files account for a majority of the accesses. For the particular case in the figure, 10 percent of the files account for nearly 40 percent of all accesses. Assuming that the decision about which files to replicate has been made, we propose that the number of replicas of each file, and the placement of each replica, can be determined using the following heuristics:

1) If a is the access probability, expressed as a percentage, of a file i which is to be replicated, then form $(a - 1)$ replicas of file i, rounded to the nearest integer, so that there can exist at most $\lceil a \rceil$ copies of the file. For example, if the access probability of a file is 6.8 percent, then six replicas of the file will be formed, which, together with the original, will result in seven copies of the file in existence.
2) The maximum number of copies of a file should not exceed N, the number of striping groups.
3) Each copy of a file must be placed in a unique striping group.

We now discuss two alternative schemes for choosing the striping group in which to store a copy of a replicated file. We define the *parent group* of a file to be the group id of the file as determined by its object id by (16). The original (first) copy of a replicated file is stored in its parent group. The two schemes presented below give the striping group of a replica of a file with respect to its parent group.

6.2 Scheme #1: Parent Group-Based Round-Robin Placement (PGBRRP)

In this scheme, if k_0^i is the parent group for a file i, then the jth replica of file i will be stored in group k_j^i, given by

$$k_j^i = \left(k_0^i + j\right) \bmod (N) + 1, \qquad 1 \le j < n_i \tag{17}$$

where N is the number of striping groups and n_i is the total number of copies of file $i (n_i \le N)$. In other words, in this scheme, the replicas of a file are stored in a round-robin fashion among the striping groups, starting from the parent group of the file.

6.3 Scheme #2: Group Wide Round-Robin Placement (GWRRP)

In this scheme, the original copy of each file is placed in its parent group as determined by (16). The replicated versions are placed round-robin a mong the striping groups as follows. Let r be the number of files that are chosen for replication, let M be the total number of replicas (i.e., excluding original copies) of all files, N be the number of striping groups, and n_i be the total number of copies of file $i (n_i \le N)$. The pseudocode for storing the jth replica of file i, denoted by k_j^i, follows:

```
Rptr = 0;
for (Rcount = 0; Rcount < M; Rcount ++)
    for (i = 1; i <= r; i ++)
        for (j = 1; j < n_i; j ++){
                if ($k_0^i \ne Rptr \bmod (N) + 1$)
                $k_j^i$ = Rptr mod (N) + 1;
            else
                $k_j^i$ = (Rptr + 1) mod (N) + 1;
            Rptr = Rptr + 1;
        }
```

Fig. 8 shows the way in which five objects having six, four, three, two, and one replica, respectively, would be stored among a configuration of nine striping groups under the two placement schemes. Intuitively, one would expect the GWRRP scheme to better balance load among the striping groups than the PGBRRP scheme.

Object id	Group id
1	1 2 3 4 5 6 7
29	2 3 4 5 6
3	3 4 5 6
40	4 5 6
5	5 6

PGBRRP

Object id	Group id
1	1 2 3 4 5 6 7
29	2 8 9 1 2
3	3 4 5 6
40	4 7 8
5	5 9

GWRRP

Fig. 8. Placement of five objects and six, four, three, two, and one replicas of each object, respectively, among nine striping groups for PGBRRP and GWRRP schemes.

6.4 Operation of Dynamic Allocation Policies with Replication

When a file is replicated, one way of choosing an interface node to source a request for the file is to execute the assignment policies of Section 4 for each replica of the file. However, this can be computationally expensive. For instance, if a file has 10 copies stored, then the policies would have to be executed 10 times for each request for the file, one for each stored copy of the file. One way to get around this problem is to use a simple round-robin rule to choose the copy of a replicated file that will be used to serve the file request. This design choice has the advantage of negligible ($O(1)$) computational overhead in choosing a copy of a replicated file to be used in serving a request for the file, and also has the added advantage of balancing workload among the copies of a replicated file. This is the approach we use in this paper. Another approach is described in Section 7. We now present the performance of the stream allocation policies for the two replication schemes.

6.5 Comparison of I-Node Load Balancing Ability

As noted in Subsection 6.1, in a skewed data access pattern, such as the Zipfian distribution considered in this paper, a few files account for a large number of files. Accordingly, we chose to replicate 10 percent of the files in the database, which amounts to 10 files. Two sets of experiments were conducted, one for each replication scheme described in Subsections 6.2 and 6.3. The files were replicated according to the two schemes, and the values of the other parameters were the same as in Section 5.

Fig. 9 shows the distribution of requests among the interface nodes for the MLA, MCA, WMLA, and WMCA policies for three cases: no replication and with replication using the two schemes. Performance of RR has not been shown because it will be similar to the case of no replication. Except otherwise noted, the graphs are for a total server load of 630 streams (MLA with replication scheme #2 could support a maximum of only 611 streams). Table 5 shows the average number of streams served by each I node, and the standard deviation from this average for an I node. The results for replication are similar to the case without any replication, viz., (W)MCA performs better than (W)MLA, WMLA performs better than MLA, and WMCA performs better than MCA, the metric being load balancing ability among the I nodes.

6.6 Comparison of Striping Group Load Balancing Ability

While load balancing performance among the I nodes in the presence of replication was not too different from that without replication, consider now the metric of load balancing with respect to the striping groups.

Fig. 10 shows the number of streams served by each of the nine striping groups for the three cases of no replication, and replication using schemes 1 and 2, for three total server stream loads: 630, 730, and 800 streams. Observe from the figure that the variation among group load seems to be the largest for the case with no replication. This is corroborated by Table 6, which shows the group-wide minimum and maximum number of streams, average number of streams per group served by the storage subsystem, and the standard deviation from the average, for the three cases. From the column for the standard deviation we observe that for a given replication policy, the load imbalance among the striping groups increases as net server stream load increases. More importantly, at a given stream load, either replication scheme results in *at least* a 50 percent reduction in the standard deviation from the average number of streams per striping group, as compared to the case with no replication. Among the two replication schemes, the PGBRRP scheme consistently outperformed the GWRRP scheme in terms of striping group load balancing ability.

6.7 Comparison of Stream Sourcing Capacity

We now consider the most important metric from an application-level point of view: the maximum number of streams supported. Table 7 shows the number of streams that were supported by each assignment policy for each replication case, together with the percentage improvement over the case of *RR allocation of stream requests to I nodes and no data replication*. For the WMLA and WMCA policies, results are shown for α values of 0.25, 0.5, and 0.75, respectively. In the case of the RR policy, replication has a derogatory effect on maximum streams that can be supported; this number was lower for both replication schemes as compared to the case with no replication. Since RR gives worst performance among the five allocation policies, this observation is not discussed further. For the MLA and MCA policies, replication has a negligible impact on the maximum number of streams that can be supported. This is to be expected, since the dominating influence on

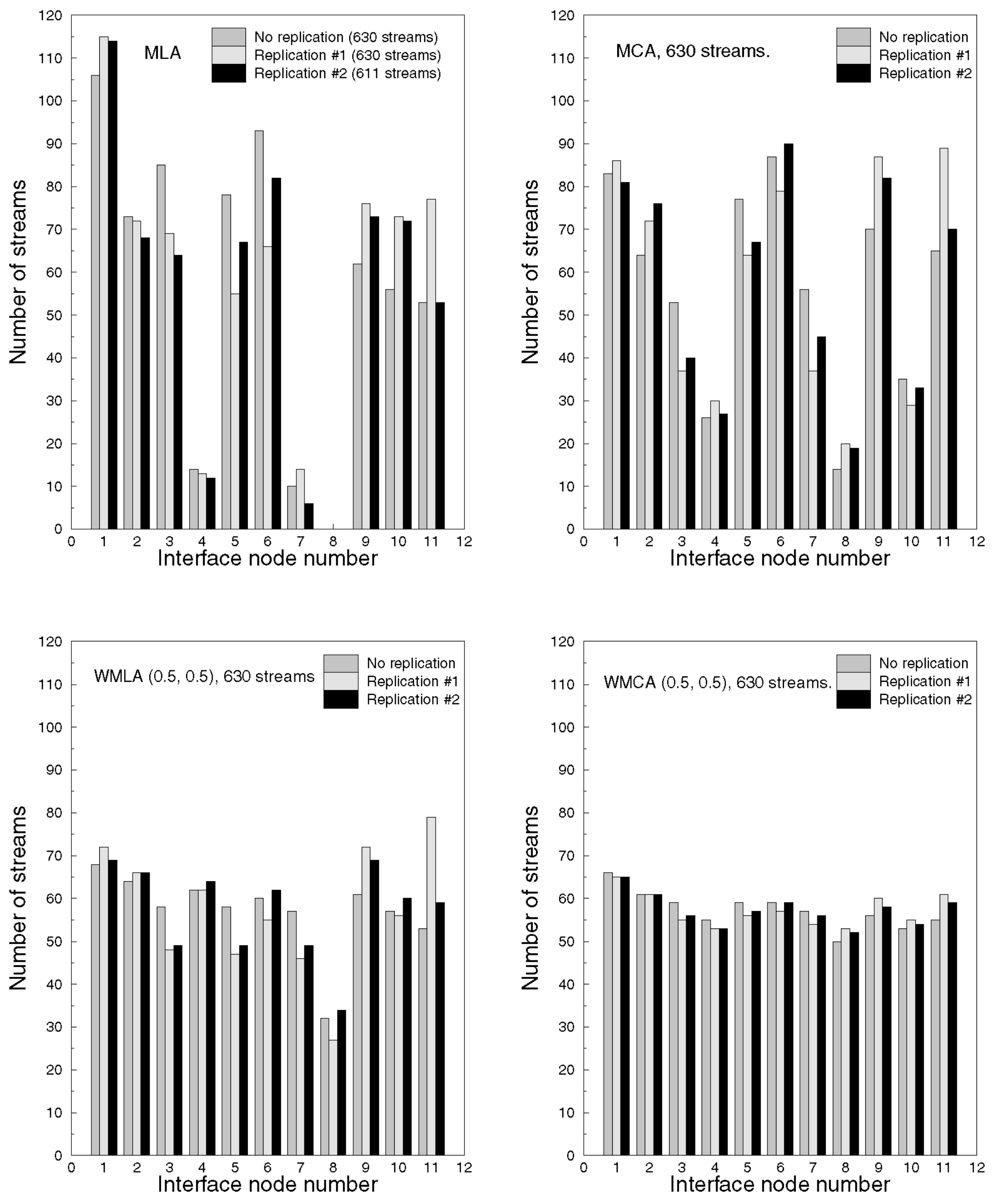


Fig. 9. Comparison of request assignment of MLA, MCA, WMLA, and WMCA policies for the two replication policies.

TABLE 5
STANDARD DEVIATION OF STREAM LOAD PER I NODE

Policy	**No Replication**		**Replication Scheme#1**		**Replication Scheme#2**	
	Avg. streams per I node	*Standard Deviation*	*Avg. streams per I node*	*Standard Deviation*	*Avg. streams per I node*	*Standard Deviation*
RR	51.36	0.48	50.73	0.45	49.55	0.5
MLA	57.27	33.75	57.27	32.85	55.55	33.7
MCA	57.27	22.49	57.27	25.59	57.27	23.88
WMLA ($\alpha = \beta = 0.5$)	57.27	8.85	57.27	14.29	57.27	10.34
WMCA ($\alpha = \beta = 0.5$)	57.27	4.07	57.27	3.74	57.27	3.77

supportable streams for these policies is the wide disparity in the number of requests assigned to each interface node. However, for the WMLA and WMCA policies, with different nonzero weights, the maximum number of streams that can be supported with replication is more than that without file replication. Moreover, among the two replication schemes, the second scheme consistently outperforms the first scheme for a given policy.

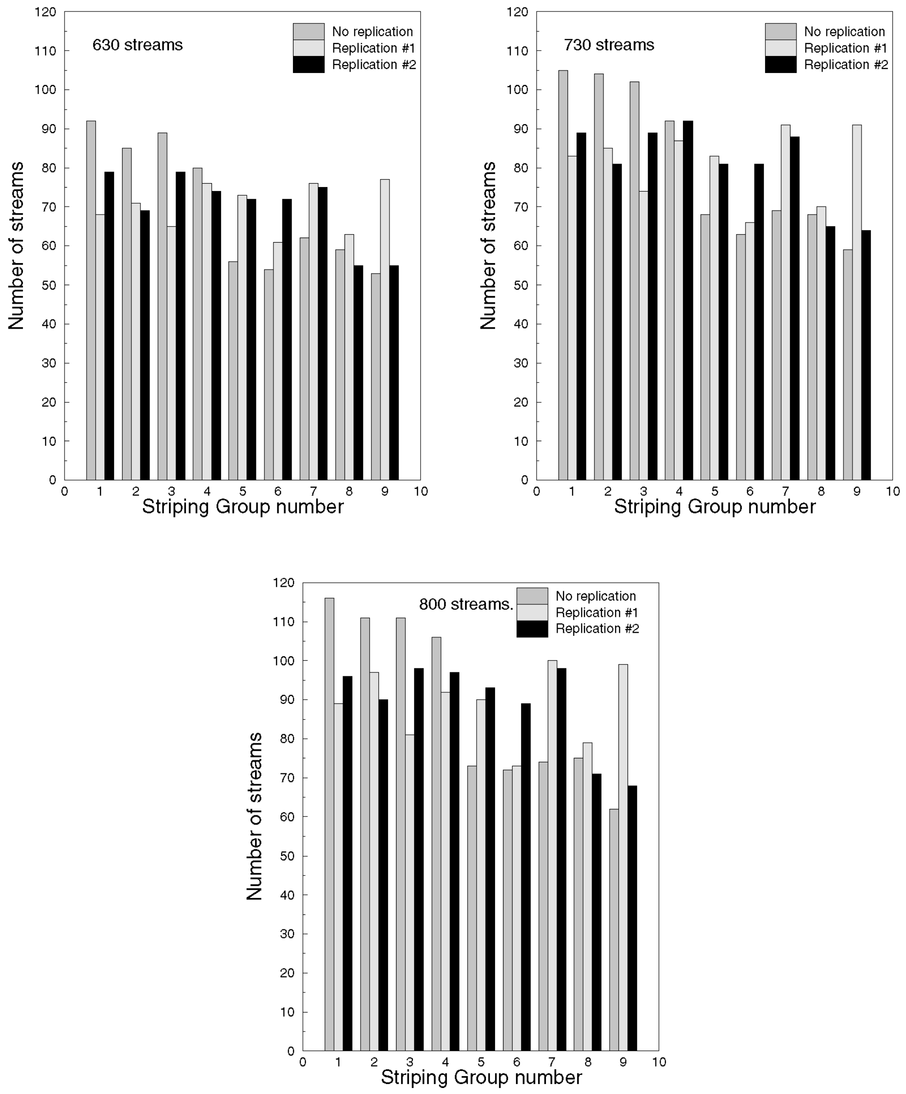


Fig. 10. Comparison of stream load on each striping group for no replication and the two replication schemes, at total stream loads of 630, 730, and 800 streams, respectively.

TABLE 6
STANDARD DEVIATION OF STREAM LOAD PER STRIPING GROUP

Total Streams served	Replication Scheme	Striping Group statistics (# of streams)			
		Min.	*Max.*	*Avg./group*	*Standard Deviation*
630	*No replication*	53	92	70.0	15.26
	Scheme #1	61	77	70.0	5.68
	Scheme #2	55	79	70.0	8.57
730	*No replication*	59	105	81.11	18.12
	Scheme #1	66	91	81.11	8.53
	Scheme #2	64	92	81.11	9.67
800	*No replication*	62	116	88.89	20.22
	Scheme #1	73	100	88.89	8.91
	Scheme #2	68	98	88.89	10.83

7 DISCUSSION AND FUTURE WORK

In this paper, we were primarily interested in determining the maximum number of streams that could be supported for each of the five policies proposed. Hence, the experiments were of the nature of *stress tests*, i.e., the workload offered to the server (in terms of stream requests) was incrementally increased until some packet delivery deadline was missed. In an actual media server, requests arriving at the server would be subject to an *admission control policy,* which would determine whether the request could be accepted or not, based on existing workload. Refer to Fig. 11, which shows the history of a client session at the server. The session consists of four stages. In the first stage, the client request would either accepted or denied depending on the result of executing the admission control policy.

TABLE 7
THE MAXIMUM NUMBER OF STREAMS SUPPORTED FOR THE REQUEST ASSIGNMENT POLICIES FOR NO REPLICATION AND THE TWO REPLICATION SCHEMES

Policy		No Replication		Replication Scheme#1		Replication Scheme#2	
Name	*Weight* $\alpha(1-\beta)$	*Max. # of streams*	*improvement over RR*	*Max. # of streams*	*improvement over RR*	*Max. # of streams*	*improvement over RR*
RR	1.00	565	-	558	-	545	-
MLA	0.00	633	12.0 %	637	12.7 %	611	8.1 %
MCA	0.00	647	14.5 %	653	15.6 %	649	14.9 %
WMLA	0.25	659	16.6 %	679	20.2 %	689	21.9 %
WMCA	0.25	677	19.8 %	701	24.1 %	727	28.6 %
WMLA	0.50	736	30.3 %	756	33.8 %	778	37.7 %
WMCA	0.50	757	34.0 %	781	38.2 %	803	42.1 %
WMLA	0.75	764	35.2 %	798	41.2 %	822	45.5 %
WMCA	0.75	797	41.1 %	805	42.5 %	823	45.7 %

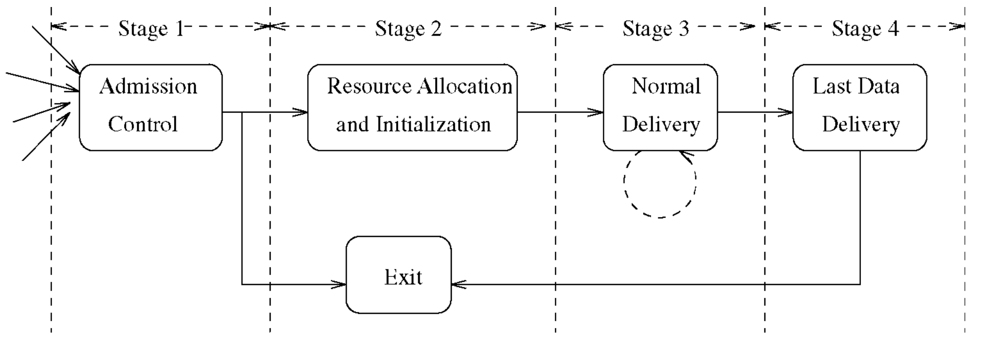


Fig. 11. History of a client session at the server.

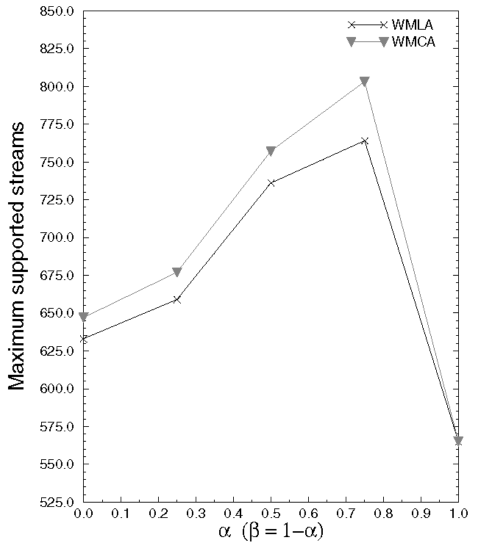
If the request is accepted, resources are allocated and initialized in the next stage. In the third stage, data is delivered to the client as per the contract negotiated. In the last stage. the session terminates and resources are freed. The allocation policies proposed in this paper could be executed in stage 1 or stage 2 of Fig. 11. Since the policies allocate streams to I nodes, it is clear why they could be implemented as part of the stage 2 software. The (W)MCA policies maintain state of interconnect workload, and hence, could be integrated as part of an admission control policy. We are developing admission control policies which can make use of these policies.

When files are replicated, the issue of which file to use for an incoming request for the file arises. As explained in Subsection 6.4, one way to handle this would be to execute the allocation policies for each copy of the requested file. However, since each policy is executed for each I node (except RR), this could be expensive. In this paper, we used the low-overhead heuristic of distributing requests for file accesses to replicated files round-robin among the replicas. Although this balances the workload among the *copies* of a replicated file, it does not attempt to balance workload among the *striping groups*. A policy that would do this can be devised: State information is maintained dynamically about the number of requests that each striping group serves. When a request for a replicated file is received, it is assigned to that copy of the file that is stored in the striping group serving the minimum number of streams among the groups which store copies of the file. We are in the process of implementing this policy.

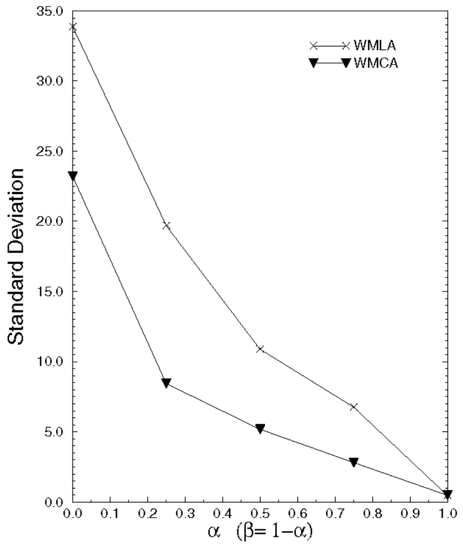
Lastly, we intend to extend our work to different hardware environments, such as the IBM SP/2 parallel computer and networks of workstations connected by high-speed links.

8 CONCLUSIONS

In this paper, we developed five policies for assigning requests to the interface nodes in a high-performance multimedia server. The performance of these policies under identical background conditions was compared. 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 to the load imposed by other nodes. This explains why WMCA gave the best throughput. Fig. 12 shows the effect of varying the weight values on the maximum number of supported streams for the five policies. For the parameters and data access pattern considered in this paper, α = 0.75, β = 0.25 gave the best performance for both WMLA and WMCA policies. The optimum values to assign to the weights is an implementation-dependent problem that depends on the network topology, routing strategy and the maximum streams that an I node can source. However, irrespective of these implementation details, changing the ratio values of α and β in one direction makes the assignment



(a)



(b)

Fig. 12. Effect of changing weight values on performance of WMLA and WMCA policies: (a) graph is for maximum streams supported, while (b) graph is for standard deviation of stream load per I node.

criterion tend to RR (α = 1, β = 0), while changing the ratio in the opposite direction will make it tend to ML(C)A (α = 0, β = 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.

We also addressed the issue of file replication in this paper. Two policies for placement of the replicas among the striping group, PGBRRP and GWRRP, were developed. Fig. 13 and Fig. 14 show the performance of the WMLA and WMCA polcies, respectively, for the three cases of no replication, replication using PGBRRP and replication using GWRRP schemes. Consider first Fig. 13. Except for the case of MLA (α = 0) and RR (α = 1), file replication results in improving the maximum number of streams that can be supported. Also, replica placement scheme #2 outperforms scheme #1 for these cases. This is also the general trend with respect to the WMCA class of policies (Fig. 14). Between PGBRRP and GWRRP, while the standard deviation of workload on the striping groups is slightly smaller for the former than that of the latter (Table 6), the standard deviation of stream load among the I nodes is smaller for the latter than that due to the former. On account of these reasons, the number of streams supported using GWRRP is more than that supported using PGBRRP for a given allocation policy. Finally, in all three variations of replication, the superiority of WMLA over MLA and WMCA over MCA, and WMCA over WMLA is maintained.

ACKNOWLEDGMENTS

This work was supported by the Intel Corporation, the National Science Foundation through Young Investigator Award No. CCR-9357840, and the New York State Center for Advanced Technology in Computer Applications and Software Engineering (the CASE Center) at Syracuse University. The authors thank the California Institute of Technology CCSF facilities for providing access to the Intel Paragon.

REFERENCES

[1] D. Anderson, Y. Osawa, and R. Govindan, “A File System for Continuous Media,” *ACM Trans. Computer Systems*, vol. 10, no. 4, pp. 311-337, Nov. 1992.
[2] D. Le Gall, “MPEG: A Video Compression Standard for Multimedia Applications,” *Comm. ACM*, pp. 46-58, Apr. 1991.
[3] A. Reddy and J. Wyllie, “Disk-Scheduling in A Multimedia I/O System,” *Proc. First ACM Int’l Conf. Multimedia*, p. 225, Aug. 1993.
[4] A. Reddy and J. Wyllie, “I/O Issues in a Multimedia System,” *Computer*, vol. 27, no. 3, pp. 69-74, Mar. 1994.
[5] P.V. Rangan and H. Vin, “Efficient Storage Techniques for Digital Continuous Multimedia,” *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 6, Aug. 1993.
[6] P.V. Rangan, H. Vin, and S. Ramanathan, “Designing an On-Demand Multimedia Service,” *IEEE Comm.*, vol. 30, no. 7, July 1992.
[7] S. Ghandeharizadeh and L. Ramos, “Continuous Retrieval of Multimedia Data Using Parallelism,” *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 4, Aug. 1993.
[8] H. Vin, A. Goyal et al., “An Observation-Based Admission Control Algorithm for Multimedia Servers,” *Proc. Int’l Conf. Multimedia Systems and Computing*, pp. 234-243, May 1994.
[9] D. Jadav and A. Choudhary, “Design Issues in High Performance Media-On-Demand Servers,” *IEEE Parallel and Distributed Technology Systems and Applications*, Summer 1995.
[10] T. Little and D. Venkatesh, “Prospects for Interactive Video-On-Demand,” *IEEE Multimedia*, vol. 1, no. 3, pp. 14-24, Fall 1994.
[11] S. Ghandeharizadeh and C. Shahabi, “Management of Physical Replicas in Parallel Multimedia Information Systems,” *Proc. Foundations of Data Organization and Algorithms (FODO) Conf.*, Oct. 1993.
[12] S. Ghandeharizadeh and C. Shahabi, “On Multimedia Repositories, Personal Computers, and Hierarchical Storage Systems,” *Proc. Second ACM Int’l Conf. Multimedia*, pp. 407-416, Oct. 1994.
[13] S. Berson, S. Ghandeharizadeh, R.R. Muntz, and X. Ju, “Staggered Striping in Multimedia Systems,” *Proc. ACM Int’l Conf. Management of Data*, pp. 79-90, May 1994.
[14] A. Dan, M. Kienzle, and D. Sitaram, “A Dynamic Policy of Segment Replication for Load Balancing in Video-On-Demand Servers,” *ACM Multimedia Systems J.*, pp. 93-103, vol. 3, no. 3, 1995.

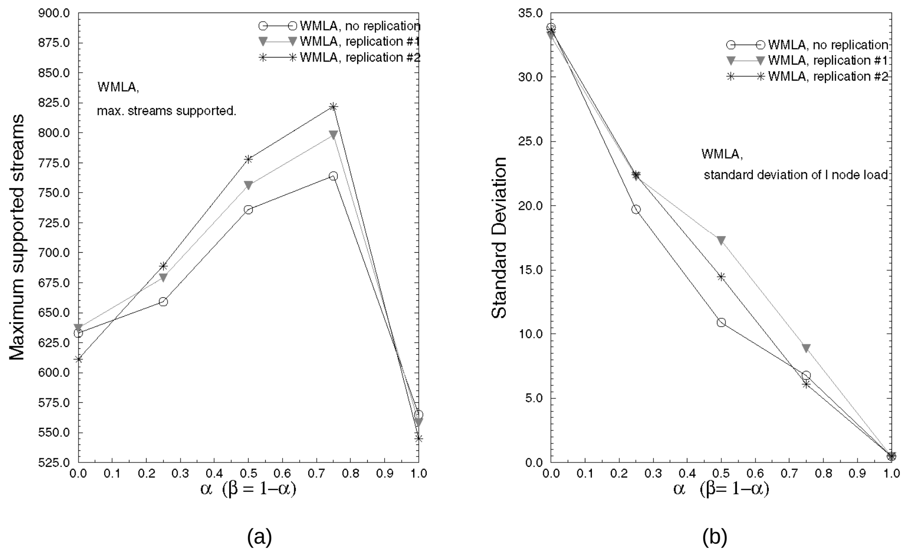


Fig. 13. Effect of changing weight values on performance of WMLA policies for no replication and the two replication schemes: (a) graph is for maximum streams supported, while (b) graph is for standard deviation of stream load per I node.

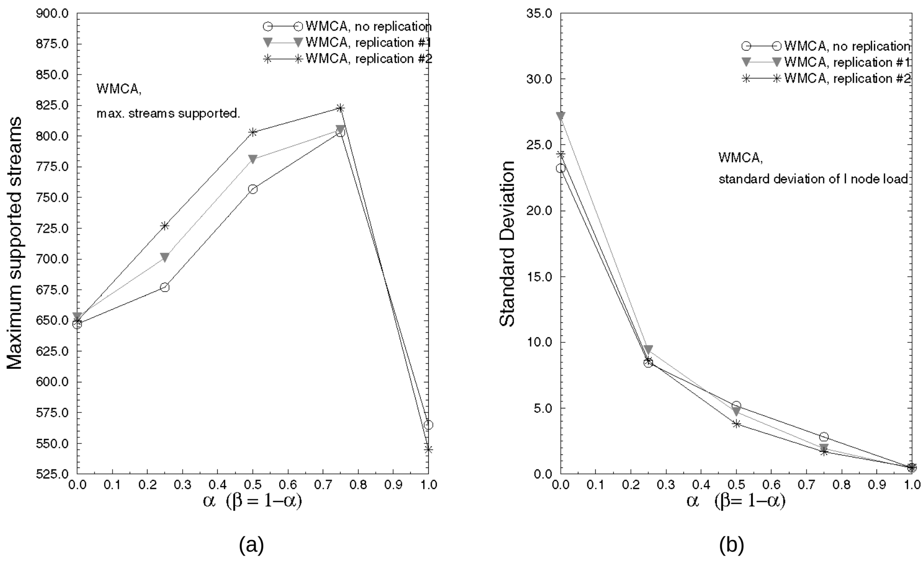


Fig. 14. Effect of changing weight values on performance of WMCA policies for no replication and the two replication schemes: (a) graph is for maximum streams supported, while (b) graph is for standard deviation of stream load per I node.

[15] C.S. Freedman and D.J. DeWitt, “The SPIFFI Scalable Video-On-Demand System,” *Proc. ACM Int’l Conf. Management of Data*, pp. 352-363, May 1995.

[16] B. Ozden, R. Rastogi, and A. Silberschatz, “Demand Paging for Video-On-Demand Servers,” *Proc. Second IEEE Int’l Conf. Multimedia Computing and Systems*, pp. 264-272, May 1995.

[17] S. Berson, L. Golubchik, and R.R. Muntz, “Fault Tolerant Design of Multimedia Servers,” *Proc. ACM Int’l Conf. Management of Data*, pp. 364-375, May 1995.

[18] C. Papidimitriou, S. Ramanathan, and P.V. Rangan, “Information Caching for Delivery of Personalized Video Programs on Home Entertainment Channels,” *Proc. Int’l Conf. Multimedia Systems and Computing*, pp. 214-223, May 1994.

[19] D. Jadav, C. Srinilta, A. Choudhary, and P.B. Berra, “Design and Evaluation of Data Access Strategies in a High Performance Multimedia-On-Demand Server,” *Proc. Second IEEE Int’l Conf. Multimedia Computing and Systems*, pp. 286-291, May 1995.

[20] D. Jadav, A. Choudhary, P.B. Berra, and C. Srinilta, “An Evaluation of Design Trade-Offs in a High Performance Media-On-Demand Server,” CASE Center Technical Report No. 9502, CASE Center, Syracuse Univ., Feb. 1995.

[21] A. Dan and D. Sitaram, “An Online Video Placement Policy Based on Bandwidth to Space Ratio,” *Proc. ACM Int’l Conf. Management of Data*, pp. 376-385, May 1995.

[22] M. McKusick, W. Joy, S. Leffler, and R. Fabry, “A Fast File System for Unix,” *ACM Trans. Computer Systems*, vol. 2, no. 3, Aug. 1984.

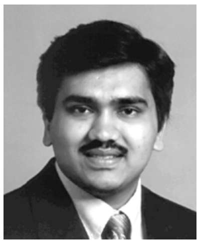
[23] L. Ni and P. McKinley, “A Survey of Wormhole Techniques in Direct Networks,” *Computer*, vol. 26, no. 2, pp. 62-76, Feb. 1993.

[24] P. S. Yu, M.-S. Chen, and D.D. Kandlur, “Design and Analysis of A Grouped Sweeping Scheme for Multimedia Storage Management,” *Proc. Third Int’l Workshop Network and Operating System Support for Digital Audio and Video*, pp. 44-55, Nov. 1992.

[25] D.A. Patterson, G. Gibson, and R.H. Katz, “A Case for Redundant Arrays of Inexpensive Disks,” *Proc. ACM Int’l Conf. Management of Data*, pp. 109-116, 1988.

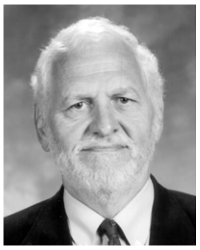
[26] A. Dan, D. Sitaram, and P. Shahabuddin, “Scheduling Policies for An On-Demand Video Server with Batching,” *Proc. ACM Multimedia*, pp. 15-23, 1994.

[27] Intel Corp., *Paragon OSF/1 User’s Guide*, Intel Supercomputer Systems Div., Feb. 1993.



Divyesh Jadav received his BE degree in computer engineering from the Victoria Jubilee Technical Institute, Bombay University, India, in 1992; and his PhD and MS degrees in computer engineering in 1997 from the Department of Electrical and Computer Engineering at Syracuse University. He has been with the IBM Almaden Research Center in San Jose, California, since 1997. His research interests include high-performance storage systems, clustered and fault-tolerant servers, and multimedia and web servers and databases. He is a member of the ACM, the IEEE and the IEEE Computer Society.

Alok N. Choudhary received his BE degree (with honors) from the Birla Institute of Technology and Science, Pilani, India, in 1982; his MS degree from the University of Massachusetts, Amherst, in 1986; and his PhD degree from the University of Illinois at Urbana-Champaign in electrical and computer engineering, in 1989. He has been an associate professor in the Electrical and Computer Engineering Department at Northwestern University since September 1996. From 1993 to 1996, he was an associate professor in the Electrical and Computer Engineering Department at Syracuse University; and from 1989 to 1993, he was an assistant professor in the same department. He has spent time visiting various industries—including IBM and Intel—as a research scientist. His main research interests are in high-performance computing and communication systems and their applications in many domains, including multimedia systems, information processing, and scientific computing. In particular, his interests lie in the design and evaluation of architectures and software systems (from system software such as runtime systems, compilers, and programming languages to applications), high-performance servers, high-performance databases, and input-output. He has published more than 100 papers in various journals and conference proceedings in the above areas. He has also written a book and several book chapters on these topics. He received the National Science Foundation's Young Investigator Award in 1993 (for 1993-1999). He has received an IEEE Engineering Foundation Award, an IBM Faculty Development Award, and an Intel Research Council Award. His past and present research has been sponsored by DARPA, NSF, NASA, AFOSR, ONR, DOE, Intel, IBM, and TI. He served as conference co-chair for the International Conference on Parallel Processing, and is currently chair of the International Workshop on I/O Systems in Parallel and Distributed Systems. He is an editor of the *Journal of Parallel and Distributed Computing* and has served as a guest editor for *Computer* and *IEEE Parallel and Distributed Technology*. He serves (or has served) on the program committee of many international conferences in architectures, parallel computing, multimedia systems, performance evaluation, distributed computing, etc. He serves in the High-Performance Fortran Forum, a forum of academia, industry and government laboratories working on standardizing programming languages for portable programming on parallel computers. He is a member of the IEEE, the IEEE Computer Society, and the ACM.



P. Bruce Berra received his BS and MS degrees from the University of Michigan, and his PhD degree from Purdue University. He is currently director of the Information Technology Research Institute in the College of Engineering and Computer Science at Wright State University in Ohio and regents professor of information technology in the Department of Computer Science and Engineering at the same institute. He is also an emeritus and research professor of Electrical Engineering and Computer Science at Syracuse University. Prior to July 1, 1996, he was director of the New York State Center for Advanced Technology in Computer Applications and Software Engineering (the CASE Center), a professor of electrical and computer engineering, and a faculty member in computer and information science, all at Syracuse University. He previously served as chair of industrial engineering and operations research at Syracuse, and has taught at the University of Michigan's Dearborn campus, Boston University, and Purdue University. His industrial experience includes periods of service with Hughes, Bendix, and IBM. He is currently president of PBB Systems, a knowledge and database consulting firm. He has served on the IEEE Computer Society Board of Governors, as editor-in-chief of IEEE Computer Society Press, on the IEEE Press Editorial Board, and as a member of the Computer Society's Distinguished Visitors Program. He has served as general chair and program chair of the International Conference on Data Engineering. He has served as an editor of *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Software Engineering*, and *IEEE Transactions on Computers*, and as chair of the IEEE Transactions Advisory Committee. He is one of the three co-founders of *IEEE Transactions on Knowledge and Data Engineering*. He pursues research interests in multimedia information systems, parallel processing for very large data and knowledge bases, and optical database machines. He was presented with the L.C. Smith College of Engineering and Computer Science Award for Excellence in Scholarship for his contributions to knowledge in his field. In 1994 and 1995, he was a finalist in the Supporter of Entrepreneurship category of the Entrepreneur of the Year Institute Awards. He is a fellow of the IEEE and a member of the IEEE Computer Society.