1995

Techniques for Scheduling I/O in a High Performance Multimedia-on-Demand Server

Divyesh Jadav
Syracuse University and CASE Center

Chutimet Srinilta
Syracuse University and CASE Center

Alok Choudhary
Syracuse University and CASE Center

P. B. Berra
Syracuse University

Techniques for Scheduling I/O in a High Performance Multimedia-on-Demand Server *

Divyesh Jadav Chutimet Srinilta Alok Choudhary

P. Bruce Berra

CASE Center and
Department of Electrical and Computer Engineering
Syracuse University
Syracuse, NY 13244

Abstract

One of the key components of a multi-user multimedia-on-demand system is the data server. Digitalization of traditionally analog data such as video and audio, and the feasibility of obtaining network bandwidths above the gigabit-per-second range are two important advances that have made possible the realization, in the near future, of interactive distributed multimedia systems. Secondary-to-main memory I/O technology has not kept pace with advances in networking, main memory and CPU processing power. Consequently, the performance of the server has a direct bearing on the overall performance of such a system.

In this paper we present a high-performance solution to the I/O retrieval problem in a distributed multimedia system. We develop a model for the architecture of a server for such a system. Parallelism of data retrieval is achieved by striping the data across multiple disks. We present the algorithms for server operation when servicing a constant number of streams, as well as the admission control policy for accepting requests for new streams. The performance of any server ultimately depends on the data access patterns. Two modifications of the basic retrieval algorithm are presented to exploit data access patterns in order to improve system throughput and response time. Finally, we present preliminary performance results of these algorithms on the IBM SP1 and Intel Paragon parallel computers.

*This work is supported by Intel Corporation, NSF Young Investigator Award CCR-9357840, and by the New York State Center for Advanced Technology in Computer Applications and Software Engineering (CASE) at Syracuse University. The authors thank the Argonne National Laboratory for providing access to the IBM SP1, and the Caltech CCSF facilities for providing access to the Intel Paragon

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. A *Multimedia Information System* requires the integration of communication, storage and presentation mechanisms for diverse data types including text, images, audio and video, to provide a single unified information system [BCG+92].

The reason why multimedia data processing is difficult is that such data differs markedly from the unimedia data (text) that conventional computers are built to handle [RaV92] :

- Multiple data streams : A multimedia object can consist of text, audio, video and image data. These data types have very different storage space and retrieval rate requirements. The design choices include storing data of the same *type* together, or storing data belonging to the same *object* together. In either case, multimedia data adds a whole new dimension to the mechanisms used to store, retrieve and manipulate the data.

- Real-time retrieval requirements: Video and audio data are characterized by the fact that they must be presented to the user, and hence retrieved and transported, in real-time. In addition, *compound objects* (objects consisting of more than one media type) usually require two or more data types to be synchronized as the object is played out. This further complicates the real-time retrieval requirements.

- Large data size: The size of a typical video or audio object is much larger than that of a typical text object. For example, a two hour movie stored in MPEG-1 [Gal91] format requires over 1 gigabytes of storage. Off-the-shelf PCs and workstations are ill-equipped to handle such storage requirements.

Multimedia information systems have been found to be useful in areas such as education, medicine, entertainment and space research, with new uses being announced day by day. In this paper, we focus on one such application, *video-on-demand* in a distributed environment. This term refers to making it possible for multiple viewers to view video data. A typical scenario would involve a remote user sitting in his/her home to connect through a computer with any video store, browse through the catalog, select a movie, and start viewing it. The viewer can perform the conventional video functions like pausing, fast-forward and rewinding of the movie. The implications of such a system on the technology and the infrastructure needed are tremendous. The storage of even a modest hundred movies requires almost a terabyte of storage capacity in the server. Similarly, gigabyte/sec and terabyte/ sec bandwidth networks are necessary to carry the movies to the consumers. In addition, software is required to translate the object requests into scheduling of the network and server resources to guarantee real-time data delivery.

In the absence of adequate hardware support, past and present interactive digital multimedia systems have been forced to make compromises such as providing single-user instead of multi-user support, small-window displays instead of full-screen display of video and image data, the use of lossy compression techniques and

low audio/video resolution. Recent advances in underlying hardware technologies, however, obviate the need for such compromises. One need only examine the state-of-the-art hardware to verify this. Asynchronous Transfer Mode (ATM) technology is increasingly becoming the candidate of choice for the high-speed networks capable of carrying multimedia data, as it has the requisite speed and the ability to carry voice and other data in a common format that is equally and equitably efficient for both [Lan94]. Compression and decompression of multimedia data can now be done on the fly at low cost, as CPUs are getting smaller and faster, and RISC technology is accentuating this progress. The capacity of secondary storage is approaching gigabytes/disk, while disk sizes and price/byte of storage decrease. Massively parallel processors of gigaflops CPU capacity and and with teraflop storage space are commercially available.

In spite of these technological advances, there is one bottleneck that plagues the realization of such a system : the speed of data transfer from the secondary data storage to main memory. Secondary to main memory data transfer time in the most popular form of secondary storage, magnetic disks, is still governed by the seek and rotational latencies of these devices. These latencies have not decreased commensurately with the advances in other areas of computer hardware. Thus, although the data transfer rates of magnetic disks are high compared to those of other forms of secondary storage (eg. CD-ROMs), stand-alone magnetic disks are inadequate for supporting multiple streams (for example, a 5 megabytes/sec disk array can, at best, support 26 MPEG-1 streams). Multimedia information systems are inherently I/O intensive, and especially so in a distributed environment, it is critical to reduce the ill-effects of this bottleneck.

1.2 Related Work

Researchers have proposed various approaches for the storage and retrieval of mu ltimedia data. Anderson et al. [AOG92] have proposed file system design techniques for providing hard performance guarantees. Reddy and Wyllie [ReW93] have proposed a disk arm scheduling approach for multimedia data. Rangan et al. [RaV92, RVR92] have proposed a model based on constrained block allocation, which is basically non-contiguous disk allocation in which the time taken to retrieve successive stream blocks does not exceed the the playback duration of a stream block. Contiguous allocation of disk blocks for a media stream is desirable, for it amortizes the cost of a single seek and rotational delay over the retrieval of a number of media blocks, thus minimizing the deleterious effects of disk arm movement on media data retrieval. However, contiguous allocation causes fragmentation of disk space if the entire stream is stored on a single disk. Moreover, if a stream is stored on a single disk, the maximum retrieval bandwidth is restricted by the data transfer rate of the disk. Ghandeharizadeh and Ramos [GhR93] get around these problems by striping media data across several disks in a round robin fashion. The effective retrieval bandwidth is then proportional to t he number of disks used. Our model is similar to this model in using data striping, round robin distribution of successive stream fragments and contiguous allocation within a given fragment. Our work differs from previous approaches in that they have not addressed the issue of exploiting data access patterns to maximize the number of simultaneous streams that a multimedia server can source.

1.3 Research Contributions

In this paper, we propose I/O scheduling algorithms for a distributed video-on-demand application. An integrated approach to the storage and retrieval of video data so as to maximize the number of users, while at the same time providing real-time service, is presented. Our model uses parallelism of retrieval to tackle the problem of the low speed of data transfer from secondary-storage to main memory. An algorithm (the *Remote Disk Stream Scheduling (RDSS) algorithm,*) for server operation when sourcing a constant number of media streams, as well as the criteria for accepting new stream requests are presented. We address the problem of buffer management that arises due to the large size of multimedia data. Two modifications of the basic RDSS algorithm, the *Local Disk Stream Scheduling (LDSS)* and the *Local Memory Stream Scheduling (LMSS)* algorithms, are developed that exploit knowledge of data access patterns to improve system throughput and response time. We are in the process of evaluating the performance of these algorithms on the IBM SP1 massively parallel processor, and report preliminary results.

The rest of this paper is organized as follows : Section 2 presents a general overview of our model. In Section 3 we describe the architecture of the server. Section 4 describes the proposed scheduling policies that exploit data access patterns to optimize service time. Admission control algorithms for these policies are put forward in Section 5. We present performance results in Section 6. Section 7 summarizes this paper and outlines our future work.

2 Overview of the Distributed Multimedia System

Figure 1 shows the overall architecture of the system which we consider.

At the heart of the system is a high-performance server optimized for fast I/O. A parallel machine is a good candidate for a server for such a system on account 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. The server is connected to a high-speed wide-area network with ATM switches. The remote clients are computers with tens of megabytes of main memory and hundreds of megabytes of secondary storage.

The data is stored at the server and transmitted in compressed digital form. As the multimedia industry evolves, standards are being enacted. For instance, the MPEG-1 standard is suitable for digital video upto a data rate of 1.5 Mbits/sec [Gal91], while MPEG-2 is a digital video standard being finalized for supporting applications such as HDTV requiring higher bandwidths of 15 Mbits/sec and beyond. The decompression of the data is done at the remote client's *multimedia terminal*, which is an intelligent computer with hardware such as a microphone, digital video camera, high-resolution graphics display, stereo speakers and a sophisticated cable decoder. The cable decoder is the interface to the high-speed wide-area network. It has tens of kilobytes of buffer space and compression and decompression hardware built into it [Per94]. This is a typical example of how the digitalization and integration being brought about by multimedia concepts is blurring the classical boundaries between the computer, communication and consumer electronics industries [Aok94].

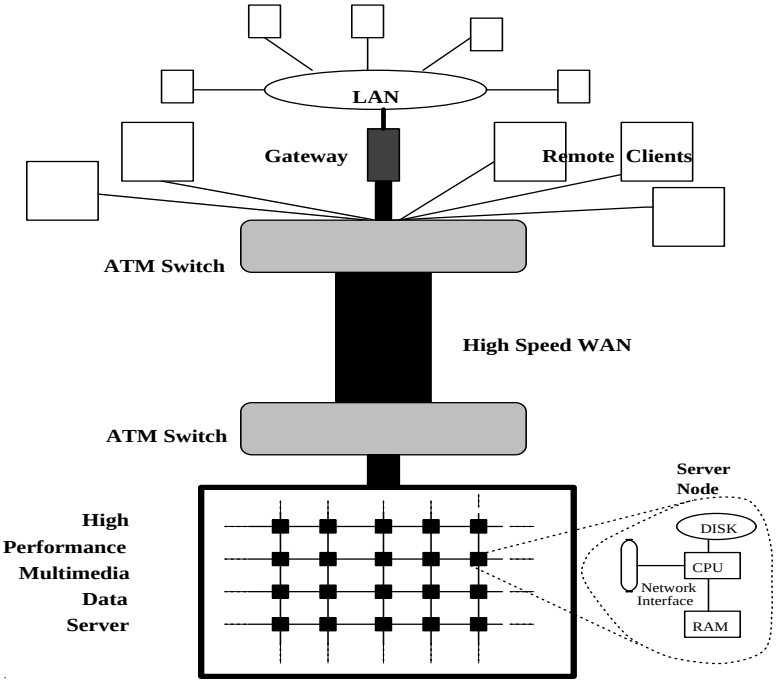

Figure 1: Block diagram of a Distributed Multimedia system

3 The High Performance Multimedia Server

3.1 Why use a Parallel Computer for the Server ?

The goal of a server for the type of application described above is to maximize the number of simultaneous real-time streams that can be sourced to clients. As explained above, the advent of multimedia applications strains the resources of a uniprocessor computer system for even a single-user mode of operation. When the server has to handle multiple requests from multiple users simultaneously, it is clear that the server must be considerably more powerful than a PC or workstation-type system. At the very least, the server should have terabytes of secondary storage, gigabytes of main memory, and a high-speed wide-area network. The server may also be required to perform fast decompression (eg. for supervisory and diagnostic purposes) and compression of multimedia data. Hence it should have good floating-point and scalar arithmetic performance. In order to satisfy all these requirements, we propose that the server be one of a class of parallel machines. Specifically, the architecture is based on the interconnection of tens to hundreds of commodity microprocessor-based nodes, which provides scalable high performance over a range of system configurations. This is the class of parallel machines that is helping in commercializing parallel processing technology [Zor92, Khe94].

At the same time, it must be noted that most parallel computers available till recently have concentrated on minimizing the time required to handle workloads similar to those found in the scientific computing domain. Hence, the emphasis was laid on performing fast arithmetic and efficient handling of vector operands. On the other hand, multimedia-type applications require fast data retrieval and real-time guarantees. I/O

constitutes a severe bottleneck in contemporary parallel computers and is the topic of vigorous research currently. [RoC94] present a comprehensive survey of the problems in high-performance I/O. Secondly, parallel computers have traditionally been expensive on account of their high-end nature and the comparatively small user community as compared to that of PCs. The advent of multimedia applications has brought the esoteric parallel machines in direct competition with volume-produced PCs and workstations. This is borne by the fact that vendors are building multimedia servers based on both MPP and PC technology. For instance, companies like Oracle and Silicon Graphics advocate powerful and expensive parallel processing technology to build multimedia servers; while companies like Microsoft, Intel and Compaq claim to achieve equivalent functionality at a lower cost by building servers by interconnecting the same chips used in PCs. [HPC94] An example of the latter approach is Microsoft's Tiger file system, which uses a high-speed communication fabric to interconnect Intel Pentium-processor based nodes.

We propose a *logical* model for a continuous media server, which is independent of the architectural implementation. The same model can be implemented on a parallel machine or a collection of PCs/workstations interconnected by high-speed links. In this paper, we have used the parallel computer approach to validate our work. We present our results for the Intel Paragon and the IBM SP1.

Accordingly, the architecture of the server is that of a parallel computer with a high-capacity magnetic disk(s) per node, with the nodes being connected by a high-speed interconnection network. This is the so-called *shared-nothing* architectural model (Fig. 1) [Sto86]. The reason for this nomenclature is that 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. Consequently, a node can operate independently of other nodes or two or more nodes can co-operate to solve the same problem in parallel. This model allows one to stripe the multimedia data across the magnetic disks of the server. This allows its retrieval to proceed in parallel, thus helping the server to satisfy real-time requirements. In addition, the shrinking size and cost of RAM makes it possible to have hundreds of megabytes of main memory per node; memory capacity of this range is an advantage for buffering multimedia data during secondary-memory storage and retrieval. Secondly, the increasing acceptance of the shared-nothing approach in a number of commercial and research database systems suggests that it will be the architecture of choice for future generations of at least commercial high-performance database machines [DeG92], if not for all large scale parallel computers.

3.2 Logical Model of the Server

Figure 2 shows a block diagram of the logical view of the proposed server.

The physical server nodes are divided into three classes based on functionality : **Object Manager** A, **Interface** I, and **Server** S nodes. In the figure, dotted lines indicate control traffic, while the solid lines indicate data traffic. (Note that the connections shown are just software (conceptual) connections and not physical links). In a typical request-response scenario, the object manager node would receive a request for an object, M. The server node(s) on which the object resides would be identified by the object manager. If the resource requirements of the request are consistent with the system load at that time, then the request

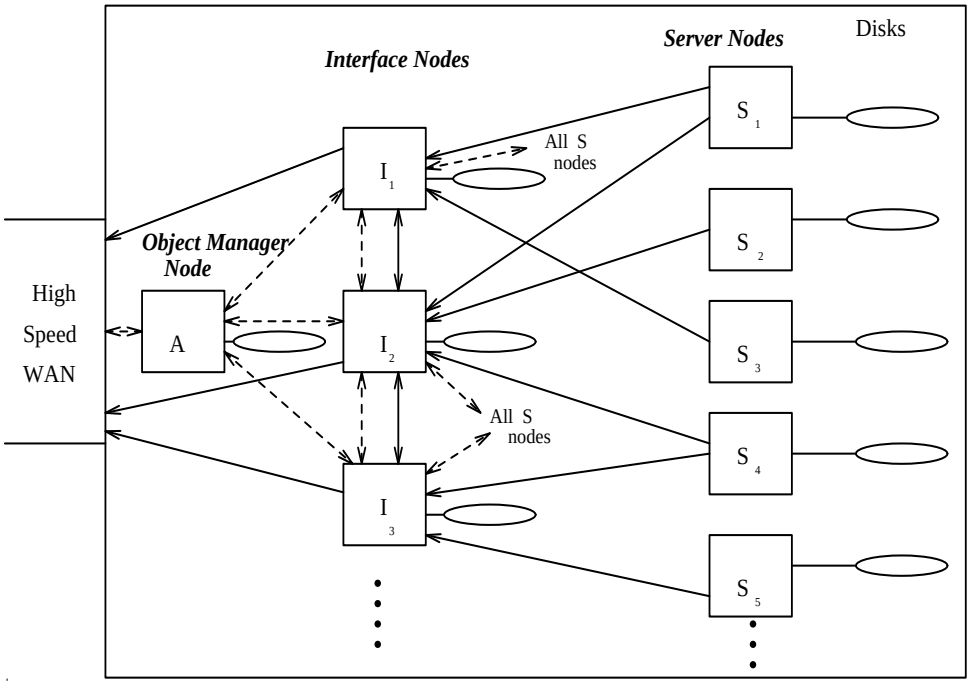

Figure 2: Logical Model of the Server. Example communication patterns are shown: dark lines indicate data, dotted lines indicate control information

is accepted. (This is explained further in section 5). An interface node to serve the stream is chosen by the object manager, and the interface node then takes over the authority and responsibility of serving the stream. To that end, it retrieves the stream fragments from the server nodes and transmits them at the required rate to the client. The three types of nodes are explained in greater detail below :

1. **The Object Manager node** is at the top of the server's control hierarchy. The Object Manager receives all incoming requests for media objects. It has knowledge of which Server 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 Object Manager node also *logs data request patterns*, and uses this information to optimize server response time and throughput. This is explained in 4.2.

2. **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 server nodes, order the packets received from the server nodes, and send the packets over the high-speed wide area network to the clients. *Efficient buffer management* algorithms are vital towards achieving these functions. An interface node can also use its local secondary storage to source frequently accessed data objects.

3. **Server Nodes** actually *store* multimedia data on their secondary storage in a striped fashion, and *retrieve* and transmit the data to an interface node when requested to do so. It is to be noted that the disk-per-node assumption is not literal : a node can have a disk-array instead for greater I/O throughput.

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
S	Stripe factor	-

Table 1: The parameters used in this paper

Given a n-node machine, interesting tradeoffs are possible with respect to partitioning the machine into node types. Since it is the interface nodes that actually source the client streams, it is desirable that their number be large, so that the total streaming capacity of the server is high. (it must be noted here that the number of interface nodes cannot be arbitrary : the server architecture and the number of ports provided by the switch interface between the server and the WAN impose an upper bound on the number of interface nodes). On the other hand, since it is the S nodes that actually store the media data, it is desirable that their number be large also, so that more objects can be stored, or the same number of different objects plus some replicas can be stored. These tradeoffs can be characterized in terms of the ratio of S nodes to I nodes. It is shown in [JCB95] that a low S to I ratio results in higher average total retrieval time compared to a high S to I ratio. Given a fixed total number of nodes and a certain ratio of S nodes to I nodes, the designer can increase the ratio so that more storage space is available. Although the total number of streams that the server can source will decrease, the designer can afford to choose disks with lower performance so that the same quality of service can be guaranteed to clients at a lower net server cost.

4 Scheduling Algorithms

4.1 Parameters Used and Scheduling Constraints

We assume that the interprocessor connection network of the server and the wide-area network have the necessary bandwidth to support multimedia data rates and multiple clients. As mentioned earlier, the data is compressed and striped across the server nodes in a round-robin fashion. The number of nodes across which an object is striped is called the *stripe factor*. Since the stripe fragments on any given server 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 [ReW93], and can be used in our model. We are concerned with harnessing the parallelism provided by striped storage and investigating the buffering policies for the data. 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:

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

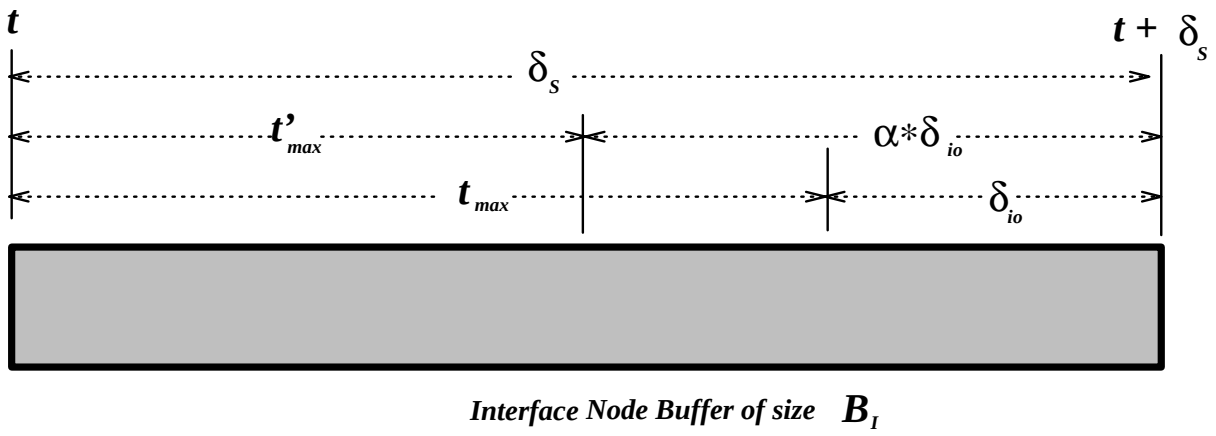

Figure 3: Time relationships of Interface-Server node traffic

Once the requested 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.

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 :

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

In order to ensure that the worst-case is not encountered, and thus to guarantee that a packet deadline is not missed, we introduce a *slack factor*, α, such that t_{max} is reduced to :

$$t'_{max} = t + \delta_S - \alpha * \delta_{io}, \qquad \alpha > 1 \tag{4}$$

Figure 3 shows these relationships. The factor α essentially overlaps playout of an I node buffer with filling it for the next round of packets. This is required since the S node packets need not arrive in order, and also to provide a cushion against delays, such as those due to interconnection network and disk traffic. We can have a similar slack-factor with respect to sending stream packets to the client. The value of the slack factor depends on factors like quality-of-service requirements, burstiness of the traffic and system utilization, among others. The computation of the slack factor is beyond the scope of this paper due to space limitations.

4.2 Exploiting Data Access Patterns

It is natural that certain objects in a database are accessed more frequently than other objects. For example, in this particular application, it is highly likely that the demand for newly released movies will be higher than that for older movies. Similarly, requests for movies will be more frequent during evenings and nights than during daytime, and more frequent on weekends than during weekdays. We now present three different algorithms that address this issue. The first algorithm does not take frequency of data access into account, while the next two exploit this feature to reduce the response time to new requests.

4.2.1 Remote Disk Stream Scheduling Algorithm (RDSS)

In this algorithm, each video stream is scheduled by explicitly retrieving stripe fragments from the S nodes. In this approach the I/O scheduler takes no advantage of the possibility that the same multimedia object is being used by multiple users simultaneously. Consequently, when many objects have this reference pattern, this policy will create excess interconnection-network and disk traffic. However, it is the simplest to implement.

4.2.2 Local Disk Stream Scheduling Algorithm (LDSS)

This algorithm and the next one depend on being able to detect that some objects are being accessed more frequently than others. This function can be performed by the object manager node (node A in figure 2). Since all new requests for streams come to this node, it can log the object access patterns over a specified *time window,* Δ_t. If any object is accessed at a rate above a threshold, Th_{pop}, then that object is classified as a *popular object*.

Having identified an object as being popular, when the next request for that object comes in, the stripe fragments are retrieved from the S nodes in the usual way. However, in addition to sending packets of size P_I to the client, the stripe fragments retrieved from the S nodes are written to the *local disk at the corresponding* I *node*. Thus, when the next request for the object comes in, the object can be streamed from the local disk of the I node. This has the benefit of reducing interconnection-network and (S node) disk traffic, and also improving the overall response time of the system. Note that the overhead of storing the stripe fragments on local disk is marginal, since disk writes are non-blocking and can proceed in the background. It is beneficial to use a disk array at the I nodes to compensate for the loss of parallelism in retrieval due to using this algorithm.

4.2.3 (Local) Memory Stream Scheduling Algorithm (LMSS)

This algorithm goes a step further in reducing system response time for popular objects. In this case, a popular object is stored on the I node backing store as in the LDSS scheme. In addition, the first few packets of the object are stored in the *main memory of the* I *node*, so that when a request comes in, it can be served immediately once it has been accepted.

In both the LDSS and LMSS schemes, it is also necessary to keep track of when the frequency of access of a object falls below the threshold separating popular object and other objects. In that case, the disk space

occupied by that object at the I node can be used to store another popular object.

5 Admission Control Policies

We define the admission control policies for new stream requests in this section. A new request can be accepted only if an I node *and* each of the S nodes across which the stream is striped can sustain the extra load due to the new stream, while still guaranteeing undisturbed service to the existing streams that each is serving at that point of time. An additional consideration is that the node interconnection network has a fixed bandwidth in the absence of link contention. The traffic on the network should be scheduled in such a manner as to achieve the maximum throughput and to minimize performance degradation due to link contention. The criteria for a S node and I node are explained first for the $RDSS$ algorithm, and then extended to the other two algorithms. This is followed by an approach for admission control which takes into account scheduling communication on the interconnection network.

5.1 Criterion for a S node

In steady-state, a given S-node will be servicing some number of client streams. T_f is the period at which an I node requests a S node for stripe fragments. Each S node maintains the *minimum* period amongst all the streams it is serving (this corresponds to the maximum rate at which the S node will have to retrieve stream fragments). We denote this parameter by $T_{f_{min}}$. This value constitutes an *upper bound* on the overhead that a S node can incur in between two consecutive transmissions of that stream. The overhead arises due to processing requests from I nodes for fragments of the streams being serviced by that S node, retrieving the requested data from disk(s), and sending it to the requesting I node. Hence, if the new request is to be accepted, the overhead due to it, when added to the existing S node overhead, must not exceed the upper bound.

The average time to retrieve a stripe fragment from a S node is given by :

$$t_{P_S} = \delta_{avg_{seek}} + \delta_{avg_{rot}} + \delta_{tr_{P_S}} \tag{5}$$

where the terms on the right-hand side are as defined in equation 2. Then, given a request for a stream M, it can be accepted if, and only if, $\forall S_i$ that will serve the stream,

$$T'_{f_{min_i}} > \sum_{j=0}^{m_i} (t_{P_{S_i}})_j + t_{P_{S_M}} \tag{6}$$

where m_i is the number of streams that S_i is currently servicing, and $(t_{P_{S_i}})_j$ is the value of t_{P_S} for the jth stream being served by the S_i. $T'_{f_{min_i}}$ denotes the minimum fetch period among the m_i streams that S_i is currently servicing *and* the requested stream i.e.

$$T'_{f_{min_i}} = \min(T_{f_{min}}, T_{f_M}) \tag{7}$$

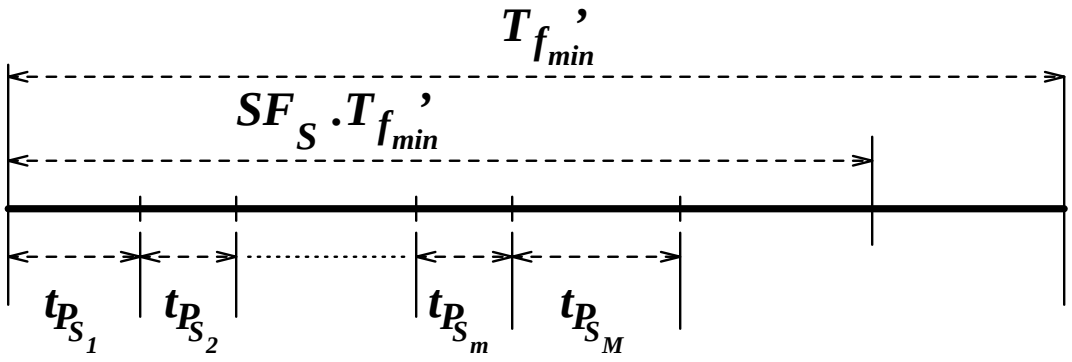

Figure 4: Admission control criterion at a S node

This criterion is illustrated in Fig. 4.

In order to ensure that the next set of packets reaches the I node before the current data in its buffer has been consumed, we must ensure that the boundary condition is not reached. Accordingly, we introduce a *S node Safety Factor, SF_S,* by modifying equation 6 to :

$$SF_S * T'_{f_{min_i}} > (\sum_{j=0}^{m_i} (t_{P_{S_i}})_j) + t_{P_{S_M}}, \qquad 0 < SF_S < 1 \tag{8}$$

The value of this factor is a function of the disk latencies, the granularity of transfer, and the number of streams that the server node is currently servicing.

5.2 Criteria for an I node

In this case two conditions must be satisfied. Firstly, there must be sufficient buffer space at the I node to satisfy buffering requirements of the new stream. Secondly, as in the case of a S node, the overhead due to the new stream, when added to the existing overhead at the I node, must not exceed the maximum allowable value (imposed by the stream that has the highest playback rate among the streams being sourced by the I node). These criteria are explained below :

- If an I node is serving n streams, and $B_{I_{tot}}$ is the total buffer space at the interface node, then in order to start serving a new stream request, M, there should be sufficient buffer space for the new stream :

$$B_{I_{tot}} > \sum_{j=0}^{n} B_{I_j} + B_{I_M} \tag{9}$$

- If to_{I_j} denotes the time overhead for composing and extracting control and data packets for stream j at the I node, then the sum of the overheads for active streams and the overhead of the new stream, M, should be less than the minimum period of transmitting stream packets to remote clients, i.e.,

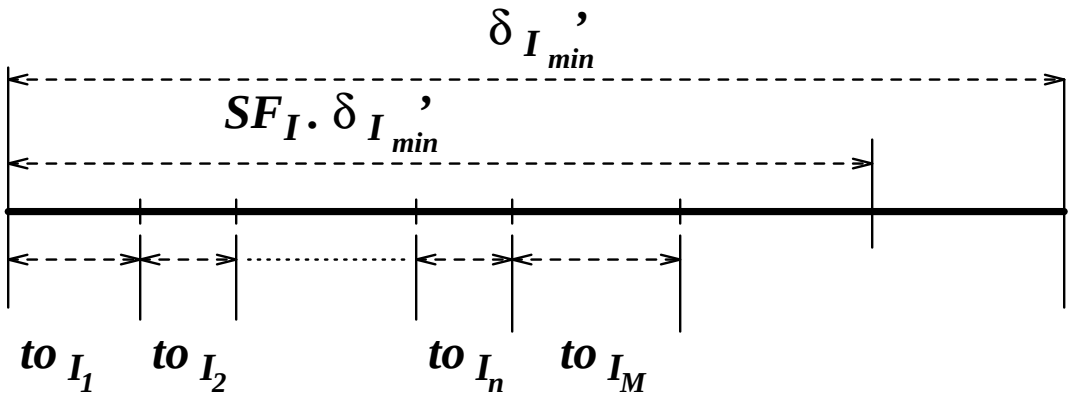

Figure 5: Admission control criterion at a I node

$$\delta'_{I_{min}} > \sum_{j=0}^{n} to_{I_j} + to_{I_M} \tag{10}$$

where, as in the case of $T'_{f_{min_i}}$,

$$\delta'_{I_{min}} = \min(\delta_{I_{min}}, \delta_{I_M}) \tag{11}$$

As in the case of a S node, to ensure that deadlines are not missed, we make the condition of equation 10 more conservative by introducing a *I node Safety Factor, SF_I,* by modifying equation 10 to :

$$SF_I * \delta'_{I_{min}} > \sum_{j=0}^{n} to_{I_j} + to_{I_M}, \qquad 0 < SF_I < 1. \tag{12}$$

The value of SF_I is a function of the number of streams that the interface node is handling and the overhead due to buffering and interconnection network transport. We are in the process of identifying and quantifying this dependence. This criterion is illustrated in Fig. 5.

5.3 Admission Control for the LDSS and LMSS Algorithms

In both these schemes, the conditions for admission control at a S node are the same as in the RDSS scheme, while the I node conditions are more complex. In both these schemes, an I node also functions as a S node for the popular object resident on its disk(s). Hence, intuitively, the conditions for accepting a request are a combination of the conditions for an I node and a S node. Moreover, when a new request comes in at a given I node, the node may or may not be home to a popular object. If the I node is not home to a popular object, then the conditions to be met in order to accept the request are identical to the RDSS case. We explain the case when it is sourcing some number of streams of a popular object; the case of migrating an object which has been detected to be a popular object to the I node is a special case, as explained below.

We derive below the conditions for the case where a given I node is home to only one popular object; they can be extended to the case when the I node is home to multiple popular objects.

Consider first the LDSS algorithm. Suppose that a given I node is serving k streams of the popular object when a request for a stream M comes in. The new request can be for a stream of either the popular object or another object. Depending on that, one of two conditions must be satisfied. With respect to equation 8, $T'_{f_{min_i}}$ is just $T_{f_{pop}}$, the value of T_F for the popular object. Let the the safety factor be denoted by SF_{IS}. Consider an interval $T_{f_{pop}}$. In the worst case, between successive fetches from disk for that stream, k disk fetches will have to be performed for the streams of the popular object. In addition, suppose l packets of the stream corresponding to $\delta_{I_{min}}$ have to be sourced in the interval $T_{f_{pop}}$. Then, if the new request is for a stream of the popular object, then we must have

$$SF_{IS} * T_{pop} > (k+1) * t_{P_{s_{pop}}} + l * (SF_I * \delta_{I_{min}}), \tag{13}$$

while if the request for a stream for another object, we must have

$$SF_{IS} * T_{pop} > (k) * t_{P_{s_{pop}}} + l' * (SF_I * \delta'_{I_{min}}), \tag{14}$$

where l' reflects the change in l (likely to be) caused by the introduction of $\delta'_{I_{min}}$ (as defined in equation 11) instead of $\delta_{I_{min}}$. Note that putting $k = 0$ in equation 13 gives the condition for making the I node as the new home of an object that has been detected to be popular object.

In addition to requiring that one of equations 13 or 14 (as applicable) hold, the I node should also have sufficient buffer space for the new stream, so that equation 9 must hold.

In terms of main memory requirements and disk usage, the only difference between the LDSS and LMSS algorithms is that in the latter case the amount of buffer space available at a given I node for allocating to a new stream is likely to be much less than that in the former case, on account of the fact that part of the popular object is stored "permanently" in main memory. Thus the conditions for accepting a new request in the LMSS scheme are identical to those for doing so in the LDSS scheme, but availability of sufficient buffer space (as embodied by equation 9) is likely to be the constraint, rather than equations 13 or 14.

5.4 Effect of the Interconnection Network on Admission Control

The derivation of admission control criteria for the interconnection network is highly dependent on network-specific factors like topology, routing, and the switching technique used. We present below an approach for a mesh-connected computer which uses *wormhole routing* to switch data from the input channels to the output channels of the network routers. An example of such an architecture is the Intel Paragon.

In wormhole routing, a packet is divided into a number of *flits* (flow control digits) prior to transmission. A header flit carries the route and the remaining flits follow in a pipeline fashion. A comprehensive survey of wormhole routing techniques is given in [NiM93, Int93]. 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

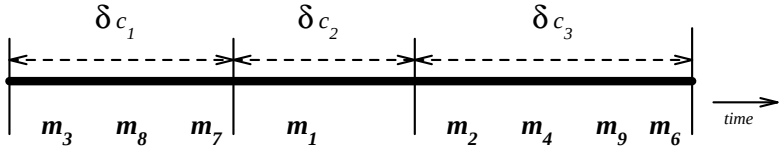

Figure 6: An example of the communication scheduling window, δ_c. Figure shows 3 consecutive windows and the streams to be scheduled in each (m_i).

values represents the latency of packets sent in the absence of network traffic and transient system activities. Blocking time includes all possible delays encountered during the lifetime of a packet, such as those due to channel contention. In order to provide a guaranteed data arrival rate at the interface nodes, this is the crucial component that must be checked for in the admission control for the network.

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. Therefore, in order to exploit this feature in a multimedia server, prior to admitting a new stream request, the server must ensure that accepting the request does not produce high link contention. This, in turn, 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.

The approach we use to schedule multiple streams over the network is that of virtual channels, in which a single physical channel is time-multiplexed among several virtual ones. Doing so guarantees the availability of a guaranteed minimum bandwidth to each virtual channel so long as the number of virtual channels sharing the same physical channel is bounded.

The communication scheduler keeps track of the streams that require data from the S nodes during a period of time called the *communication scheduling window, δ_c*. For instance, figure 6 shows the streams whose data needs to be scheduled to be retrieved from the S nodes during a certain span of three windows.

Corresponding to a δ_c, a matrix known as the *stream connectivity matrix (SCM)* of size n x k is maintained, where n is the number of source nodes and k is the number of destination nodes for network data. Clearly, n equals the total number of server nodes and k equals the number of interface nodes in the server configuration. Figure 7a shows the SCM for δ_{c_1}, where s_i represents the ith source node, and d_i represents the ith destination node.

In other words, the SCM stores which S nodes need to communicate with which I nodes during the communication scheduling window. In dimensional XY routing, given a s_i and d_j, the path traversed by packets is completely determined. Consequently, given the SCM for a time window, it is easy to identify the links that will carry the data during the time window. This information is computed and stored in a vector called the *link utilization matrix (LUM)*, which has an entry for each link in the mesh. Figure 7b shows an example LUM, where the value of an element represents the usage count of the corresponding link, as

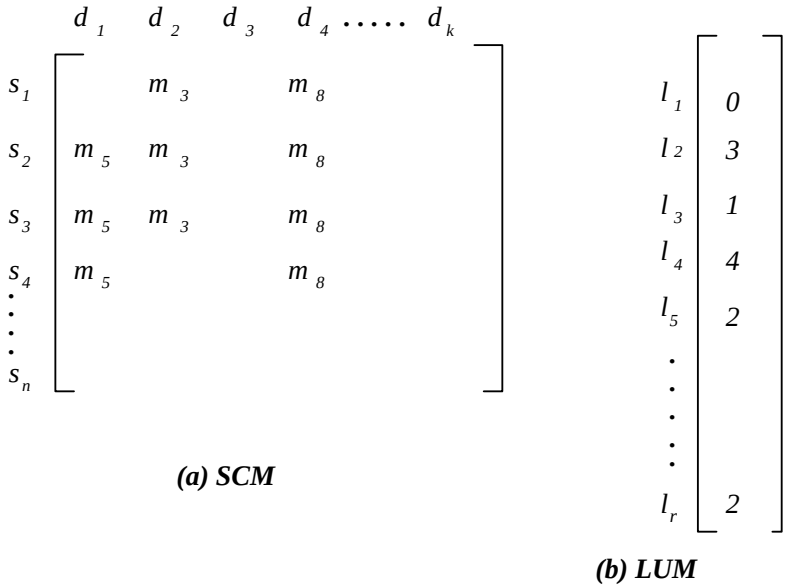

Figure 7: (a) The stream connectivity matrix (SCM) for δ_{c_1}. (b) An example link utilization matrix (LUM).

explained below. (r is the total number of links in the mesh).

We now explain how the the SCM and LUM can be used for admission control of new stream requests. Since the bandwidth of a physical channel is fixed, there is a limit on the number of virtual channels that can simultaneously share a physical channel if each virtual channel is to be guaranteed a minimum bandwidth. The number of streams contending for use of a physical channel during a δ_c is maintained by the LUM. Each stream that uses link i increases the value of LUM(i) by a fixed amount. Given an interconnect, the actual value depends on the packet size (P_S) and bandwidth required by the stream. In the simplest case, we can assume that all streams have the same playback rate and packet size, so that each stream using link i increases the value of LUM(i) by one. Since the maximum bandwidth of a given interconnect is known, it can be translated to a *link threshold*, l_{th}. Accordingly, given the SCM and LUM for a δ_c, a new stream request can be accepted *only if accepting the request leaves the LUM in a safe state, i.e. LUM(i)* $\leq l_{th}, \forall i$.

The operation of this scheme is an iterative process, whereby at the beginning of each δ_{c_i}, the LUM is computed from the SCM. If there is a pending request for a new stream, the links it needs to use if it is scheduled during the given δ_c, say l_1, l_2, ... l_p, are computed from the source and destination nodes for the request. If

$$LUM(i) + 1 \leq l_{th}, \qquad \forall i = l_1, l_2, ...l_p \tag{15}$$

then the new request can be accepted and scheduled during the given δ_c while still providing the reserved bandwidth for the existing streams. If the request is accepted, then the SCM and LUM for δ_{c_i} are updated; if the request is not accepted, then the same procedure is repeated for $\delta_{c_{i+1}}$. If the request cannot be accepted in any of the scheduling windows, then the server cannot accept the new request due to interconnection network saturation. The client is turned away and must try again after some time.

Figure 8 shows an example of the admission control algorithm. Figure(a) shows an example mesh

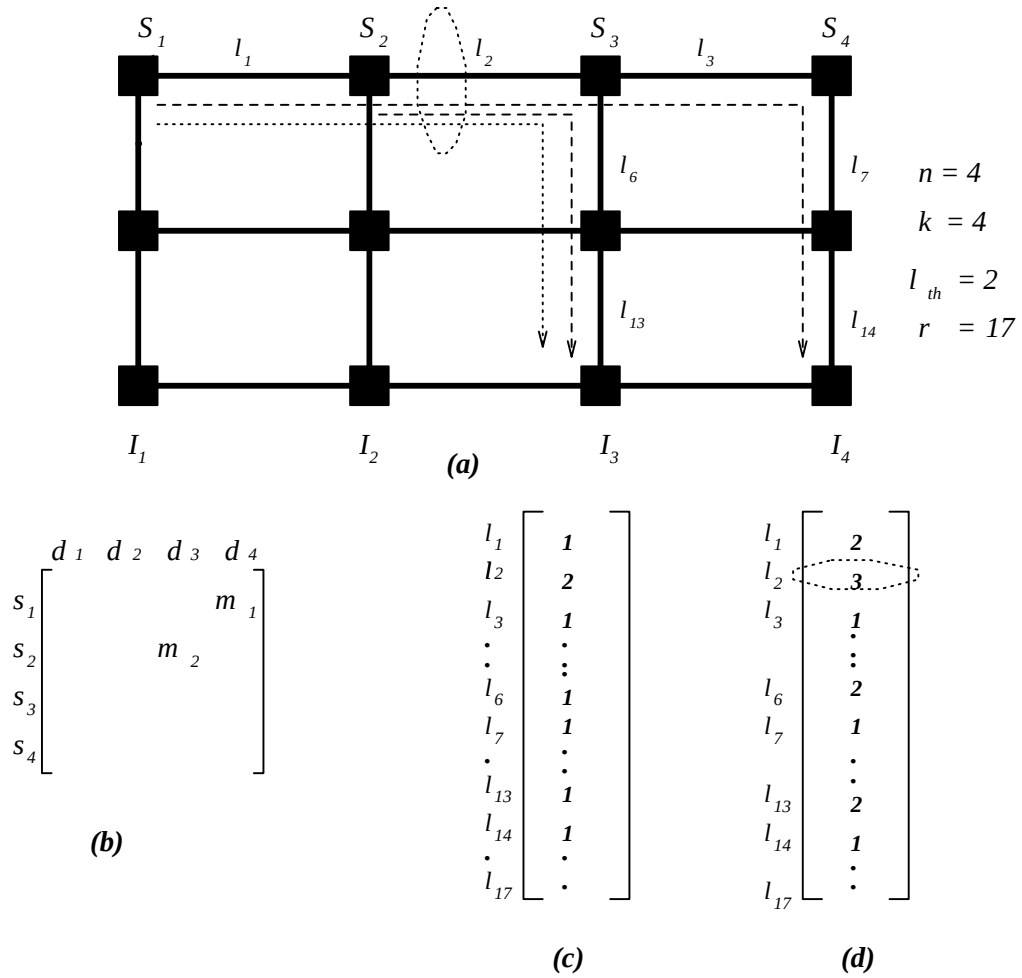

Figure 8: Example of the admission control policy. (a) Example configuration. (Dashed lines represent existing communication pattern, dotted lines show pattern required by the pending request) (b) SCM for the δ_c. (c) LUM corresponding to the SCM. (d) Result of applying the admission control policy to the LUM.

configuration with 4 S nodes and 4 I nodes (thus $n = k = 4$). In a certain δ_{c_i}, node S_1 needs to communicate with node I_4, and node S_2, with node I_3. Figure 8b shows the corresponding SCM, and figure 8c shows the LUM for the SCM. Assume that $l_{th} = 2$ for this case. Thus, link l_2 is already saturated. If a request requiring S_1 to communicate with I_3 is pending, the admission control policy tries to see if the request can be scheduled in the current δ_c. Figure 8d shows the result of applying equation 15 to the LUM. As shown in the figure, the LUM(l_2) exceeds l_{th}, and consequently, the request cannot be scheduled in the δ_c under consideration.

Before closing this subsection, we mention some implementation issues. The communication scheduler that executes the admission control algorithm needs centralized information regarding stream scheduling. Hence, with reference to the logical model, it is best implemented as part of the object manager node. Secondly, the size of a communication scheduling window is a design choice that depends on many factors like packet size, playback rate and server work load. In the simple case of a single playback rate and uniform packet size, a lower bound would be the time to transfer P_S bytes over the interconnection network, while an upper bound is the duration of a service round (the time to cycle through replenishing the interface node buffer of all streams being served).

Lastly, note that the analysis for admission control has been performed with respect to the data packets only i.e. the traffic due to the control packets has been neglected. This can be justified as follows : The size of the control packets is very small (few bytes) compared to the size of the data packets (tens/hundreds of kilobytes). Moreover, since we use virtual channels, some bandwidth can be reserved for the control packets; the bandwidth required will be small. Lastly, with reference to the retrieval process, most of the control messages travel in the direction opposite to that travelled by the data messages. Assuming bidirectional links, the small control messages do not cause too much of traffic interference.

6 Results

We have evaluated the performance of the three scheduling algorithms. We present preliminary results for two popular parallel machines, the IBM SP1 and the Intel Paragon below.

The IBM 9076 SP1 uses RISC processor technology. The compute nodes are interconnected by a high-performance switch. A 128 node machine has been installed at Argonne National Laboratories [Gro93] that has 128 Mbytes main memory per node. The notable feature of this machine is that the nodes can be used in isolation, as stand-alone workstations, or in unison as a parallel machine. Three communication modes are available : IP, EUI and EUIH. The first mode is useful when using the machine as a collection of interconnected workstations running NFS. The second and third modes are for parallel configurations, with EUIH being a faster mode than EUI. We used EUIH for our experiments.

The Intel Paragon [Hwa93, Int93] is a mesh-based architecture with Intel i860XP microprocessors. There are two types of nodes : compute nodes and I/O nodes, but their number and hardware configuration is user controlled. Each node is connected to a mesh-routing chip that connects to the interconnection network. A node is connected to its neighbours in the north, south, east and west directions through the mesh routing

Symbol	*Description*	*Value*
R_{pl}	Required playback rate	1.5 Mbits/sec
P_I	Size of packets sent by an I node	64 Kbytes
B_I	Buffer size at an I node	1 Mbyte
P_S	Size of packets sent by a S node	128 Kbytes

Table 2: The parameter values used for the experiments

chip. Interprocessor communication is done using (XY) wormhole routing. Any node can communicate with any other node in software.

The disk access part was simulated on account of the following reasons. The machines used were the 128 node SP1 at Argonne National Laboratories and a 56 node Paragon at Caltech. These are research machines that are shared by users all over the world. Hence, it was not possible to get the sufficient storage space for real data. Moreover, these machines do not have the required I/O configuration i.e. a disk array per node. We have assumed gigabytes of disk space per node, and a disk data transfer rate of 10 Mbytes/sec. 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 experiments. [1]. The database size used was 500 objects. A slack factor of 1.4 was sufficient to guarantee that no deadlines were missed. The total run time of each experiment was 5 minutes. Consequently, the playback time for each stream varied between 4 and 5 minutes, depending on the time of arrival of the request for that stream.

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, $0 \leq DoO \leq S$ (S is the stripe factor) and denotes the distance between the ith stripe fragment of object j and the ith stripe fragment of object $j+1$, in terms of the number of storage nodes. The concept of DoO is illustrated in figure 9.

Note that numerous tradeoffs are possible with respect to the data partitioning strategy, which are well reported in [GhR93, GhS93]. We are in the process of investigating such tradeoffs in our model. However, these are not the subject of this paper. Without loss of generality, then, for the purposes of this paper, we assume a DoO of 2 for all the experiments.

6.1 Performance of the RDSS, LDSS and LMSS algorithms

We noted the performance of the algorithms for a server configuration of 6 interface nodes and 24 server nodes, and a strip factor of 4. The composition of the requests was varied as follows : starting from requests

[1] Note that a whole set of design tradeoffs exists with respect to the size of B_I and P_S. We have addressed this issue in [JCB95]. However, that is not the subject of this paper. Without loss of generality, we assume the sizes mentioned in the table for all the experiments.

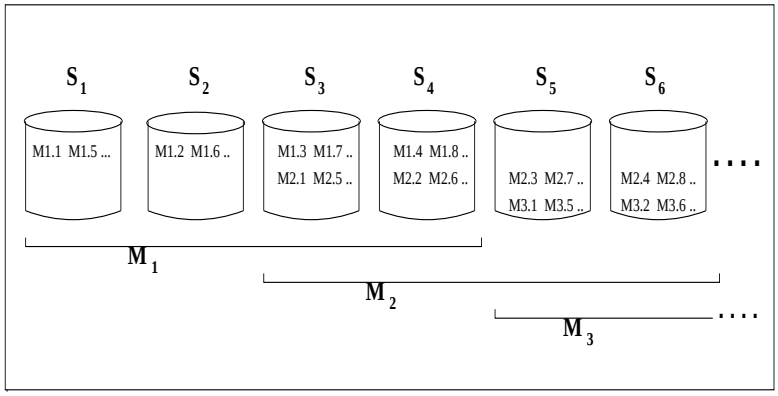

Figure 9: Degree of overlap (DoO). Figure shows 3 objects, M_1, M_2 and M_3 striped across 6 storage nodes, with a DoO of 2

for unique media objects (uniform frequency of access), the percentage of requests for the same object was successively increased. Figure 10 shows the maximum number of streams that could be simultaneously supported using each policy on the SP1.

We observe that for a low percentage of requests for the same object, the $RDSS$ algorithm outperforms the other two algorithms. This is so because in the latter two cases we allocate a dedicated I node for the popular object. For a low percentage of requests for the popular object, the dedicated node is underutilized : it sources less streams than its full capacity, while a normal I node in its place could have sourced the maximum number of streams that such a node can source. With increasing amounts of requests for the same object, however, the $LDSS$ and $LMSS$ algorithms outperform the $RDSS$ algorithm as they reduce the load on the server nodes caused by frequently accessing the same object. Between the $LDSS$ and $LMSS$ algorithms, the latter clearly outperforms the former for different values of the percentage of requests for a popular object. Lastly, the performance of the $RDSS$ algorithm deteriorates rapidly as the percentage of requests for the popular object is increased, due to the corresponding increase in the load of the S nodes on which the popular object is stored.

We ported our code to the Intel Paragon and repeated the same experiment as above. Figure 11 shows the results we obtained. The effect of varying the number of requests for the same object on the maximum number of streams that can be supported is similar as above. One difference is that the number of streams that can be supported was higher for the Paragon than for the SP1, for all three algorithms. The most important reason for this is the difference in the interconnection network bandwidth. For the SP1, we attained the maximum bandwidth of 8.5 Mbytes/sec reported in [Gro93]. Although the maximum link bandwidth of the Paragon is 200 Mbytes/sec [Int93], this is the theoretical value. Software overheads prevent this value from being attained. We measured it as 13.5 Mbytes/sec. However, this is still better than that of the SP1, which accounts for the better performance.

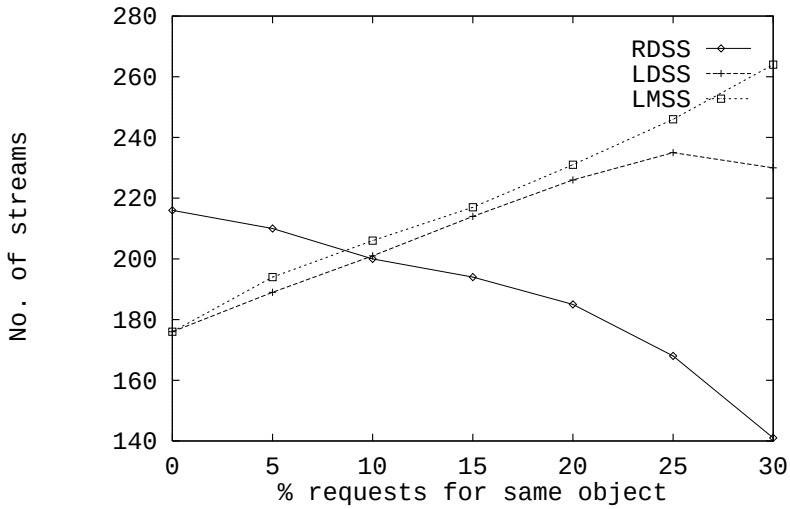

Figure 10: Maximum number of streams that can be supported for each algorithm for 6 I nodes and 24 S nodes, for varying number of requests for the same object, on the IBM SP1

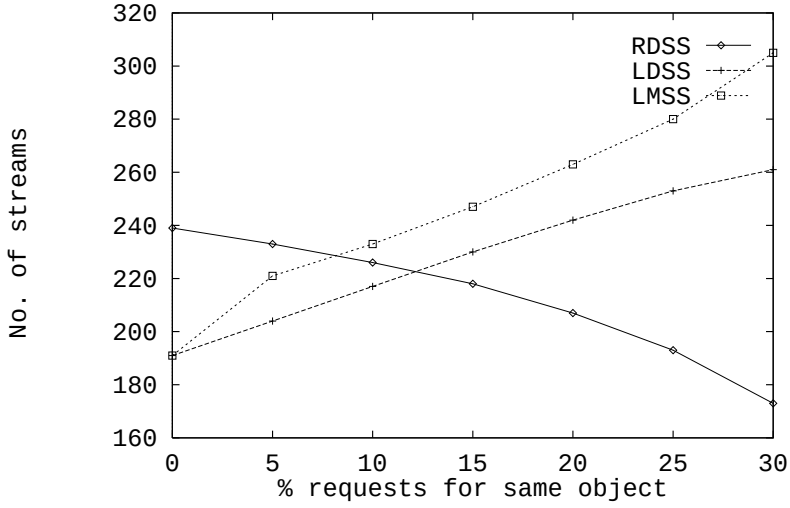

Figure 11: Maximum number of streams that can be supported for each algorithm for 6 I nodes and 24 S nodes, for varying number of requests for the same object, on the Intel Paragon

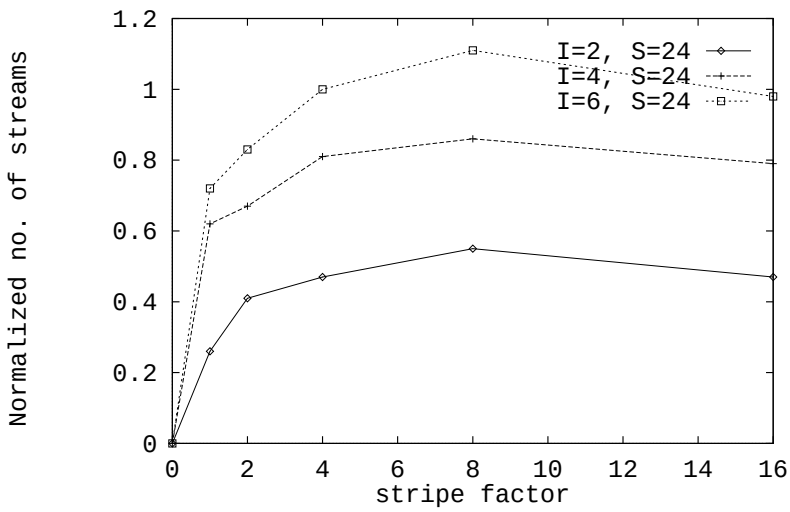

Figure 12: Normalized number of streams that can be supported for each algorithm for various stripe factors and number of Interface nodes, on the IBM SP1 (normalized with respect to the point (0,216) for the curve corresponding to the RDSS algorithm in figure 6).

6.2 Effect of varying the stripe factor (S)

In another experiment, we investigated the results of varying the stripe factor on the number of streams that can be supported. In this experiment, the buffer size at the interface node was $2 * P_s * S$ (table 2). The value of S was varied. All other values were the same as in table 2. The results are shown in figure 12 for the SP1, where the number of streams supported have been normalized with respect to the point (0,216) for the curve for the RDSS algorithm in figure 10.

The number of streams that can be supported for a given number of interface and server nodes increases as the stripe factor is increased. This is on account of the fact that increasing the stripe factor increases the amount of data retrieved per stream by the interface nodes from the server nodes. Consequently, the frequency of fetching from the I nodes is reduced. There is a corresponding decrease in retrieval overhead at the I nodes, which translates into a gain of it being able to support more streams. However, the stripe cannot be increased indefinitely; at higher stripe factors, the performance degrades due to the greater volume of traffic on the server's interconnection network. Another point to be noted from the graph is that a fixed stripe factor, increasing the number of interface nodes increases the number of supportable streams. This supports the use of a MPP for the server, since the designer has at his disposal multiple nodes, and these can be easily partitioned between interface and server nodes in such a way as to maximize the use of the server's resources.

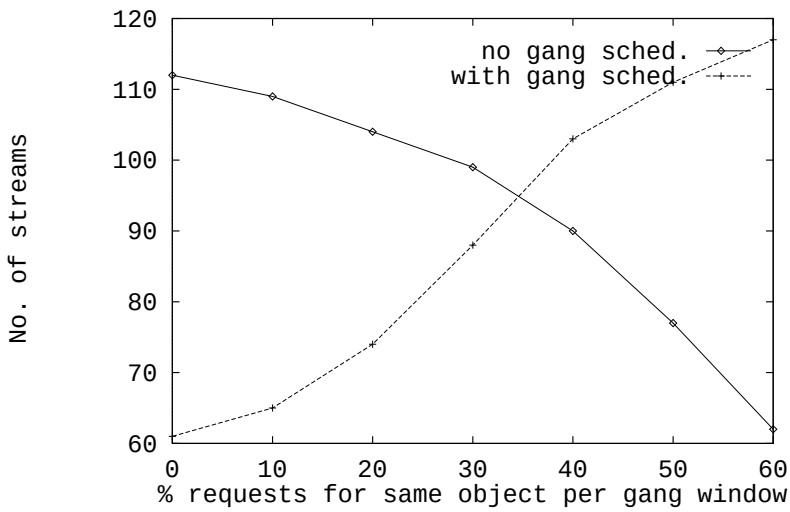

Figure 13: Number of streams that can be supported for RDSS algorithm for stripe factor of 5, 2 I nodes and 6 S nodes, for varying number of requests for the same object per gang window (Paragon).

6.3 Gang Scheduling

The $LDSS$ and $LMSS$ algorithms exploit the fact that some objects are more popular than others, and thus are requested more frequently. This fact is used to maximize the number of supportable streams of such objects by dedicating nodes to service requests for them.

In the first set of experiments, the servicing of a request is started as soon as the request has been admitted. The performance of all three algorithms can be improved by accumulating requests over an interval of time, and avoiding multiple fetches for requests received for the same object during that interval of time. We call this method *gang scheduling*. For instance, if during a *gang window* of 5 minutes, 10 requests are received for a certain object, then the server can start retrieving only one stream at the end of the gang window and source 10 client streams from the one stream. Clearly, this requires that all the 10 requests will have to wait till the end of the gang window before service can start. One stream can be used to serve multiple clients by means of the multicast [Bou92] facility.

For evaluating gang scheduling, we used a configuration of 2 I nodes and 6 S nodes, and a stripe factor of 5. We used a gang window of 1.5 seconds and 30 requests per gang window. Of course, in practice a longer window would be used. Without loss of generality, we use the window mentioned for the run time of 5 minutes. The values of the other parameters are the same as in table 2. Figure 13 shows the effect of varying the percentage of requests for the same object per gang window on the maximum number of streams that can be supported on the Paragon for the RDSS algorithm.

Gang scheduling involves an extra overhead of accumulating requests over the gang window and searching through the accumulated requests to identify repeated requests. Hence we observe from the figure that RDSS with gang scheduling is inferior to pure RDSS for low number of repeated requests per gang window. However,

as the percentage of requests for the same object per gang window increases, RDSS with gang scheduling identifies the request pattern and outperforms pure RDSS.

In effect, this method delays the servicing of some admitted requests in order to minimize the load on the server. Hence there is a tradeoff between the response time for clients and reduction in server workload. Consequently, the size of the gang window is a crucial parameter in making use of gang scheduling. An approach similar to gang scheduling is treated at length in [DSS94], where it is also shown that the nature of customer waiting time tolerance leads to scheduling tradeoffs.

7 Conclusions and Future Work

In this paper we have presented an I/O model for a server in a distributed multimedia system. Three algorithms that exploit knowledge of data access patterns were developed to maximize the number of streams that the server can source simultaneously. Admission control policies for the three algorithms were presented. Preliminary experiments show that the $LMSS$ algorithm outperforms the $LDSS$ algorithm, which in turn outperforms the $RDSS$ algorithm when an appreciable percentage of stream requests are for the same media object. We have shown the effect of varying the stripe factor on the number of streams that can be supported. Increasing the number of interface nodes translated into the ability to support a greater number of streams. We showed the utility of gang scheduling in further improving the server performance. In gang scheduling, a single stream between interface and server nodes is used to serve multiple clients. One problem with this approach is that if one of the clients interrupts the stream, say for pausing or fast forward, then that client will fall out of phase with the single stream being retrieved. Hence the server should be able to dynamically establish a fresh server-interface stream for the interrupting client. We are developing solutions to this problem so that the delay seen by the interrupting client is minimum. We are also developing algorithms for selecting an interface node for serving as the home for a popular object, and for combining object replication with knowledge of data access patterns to maximize the number of simultaneously supportable streams, with guaranteed playback rates.

References

[BCG+92] P. B. Berra, C.-Y. Chen, A. Ghafoor and T. Little. Issues in networking and data management of distributed multimedia systems. In *proceedings of the First International Symposium on High Performance Distributed Computing*, September 1992.

[RaV92] 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.

[Lan94] J. Lane. ATM knits voice, data on any net. *IEEE Spectrum*, February 1994.

[Gal91] D. Le Gall. MPEG: a video compression standard for multimedia applications. *Communications of the ACM*, April 1991.

[Per94] T. Perry. Technology 1994: Consumer Electronics. *IEEE Spectrum*, January 1994, pg. 30

[Aok94] T. Aoki. Digitalization and integration portend a change in life-style. *IEEE Spectrum*, January 1994, pg. 34.

[Zor92] G. Zorpette. Teraflops galore. *IEEE Spectrum*, September 1992.

[Khe94] G. Khermouch. Technology 1994 : Large Computers. *IEEE Spectrum* , January 1994.

[Hwa93] K. Hwang. Multiprocessors and multicomputers. *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* , McGraw Hill, 1993.

[NiM93] L. Ni and P. McKinley. A survey of wormhole techniques in direct networks. *IEEE Computer*, February 1993.

[Gro93] W. Gropp, editor. Early experiences with the IBM SP1 and the high performance switch. *Tech. Rpt. ANL-93/41, Argonne National Laboratory*, November 1993.

[RoC94] J. Rosario and A. Choudhary. High-performance I/O for parallel computers: problems and prospects. *IEEE Computer*, March 1994.

[HPC94] *HPCwire (electronic magazine)*, article number 4097, May 1994.

[Sto86] M. Stonebraker. The case for shared-nothing. *Database Engineering*, Vol. 9, No. 1, 1986.

[DeG92] D. DeWitt and J. Gray. Parallel database systems: the future of high-performance database systems. *Communications of the ACM*, Vol. 35, No. 6, June 1992.

[ReW93] A. Reddy and J. Wyllie. Disk-scheduling in a multimedia I/O system. *Proceedings of the 1st ACM Intl. Conference on Multimedia*, August 1993, pg. 225.

[RVR92] P. Rangan, H. Vin and S. Ramanathan. Designing an on-demand multimedia service. *IEEE Communications*, Vol 30, No. 7, July 1992.

[GhR93] 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.

[AOG92] D. Anderson, Y. Osawa and R. Govindan. A file system for continuous media. *ACM Trans. on Computer Systems*, Vol. 10, No. 4, November 1992.

[MTR94] P. McKinley, Y. Tsai and D. Robinson. A survey of collective communication in wormhole-routed massively parallel computers. *Technical Report MSU-CPS-94-35, Dept. of Computer Science, Michigan State University*, June 1994.

[GhS93] S. Ghandeharizadeh and C. Shahabi. Management of physical replicas in parallel multimedia information systems. *Proc. of the 1993 Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

[JCB95] Divyesh Jadav, Chutimet Srinilta, Alok Choudhary and P. Bruce Berra. An evaluation of design tradeoffs in a high performance media-on-demand Server, submitted to *ACM Multimedia Systems Journal*, January 1995. (Also available as *CASE Center Research Report*, CASE Center at Syracuse University, February 1995).

[Int93] Intel Corporation. *Paragon OSF/1 User's Guide*, Intel Supercomputer Systems Division, February 1993.

[Bou92] J. Le Boudec. The asynchronous transfer mode : a tutorial. *Computer Networks and ISDN Systems*, Vol. 22, 1992.

[DSS94] A. Dan, D. Sitaram, et al. Scheduling policies for an on-demand server with batching. *Proc. of the 2nd ACM Intl. Conf. on Multimedia*, October 1994.