An Evaluation of Design Tradeoffs in a High Performance Media-on-Demand Server *

Divyesh Jadav Chutimet Srinilta Alok Choudhary

P. Bruce Berra

Department of Electrical and Computer Engineering
and
CASE Center
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 identify the design parameters that affect the throughput of the server. We have implemented our model on the Intel Paragon parallel computer. We have performed an extensive performance evaluation of how the parameters identified affect the data retrieval efficiency of the server. The results of component-wise instrumentation of the server operation are presented and analyzed. The performance of any server ultimately depends on the data access patterns. Two modifications of the basic retrieval algorithm that exploit data access patterns in order to improve system throughput and response time are presented. Based on our experiments, a dynamic admission control policy that takes server workload into account is proposed.

Keywords

Parallel Input-Output Media-on-Demand Server Striping
Real-time data retrieval Data Access patterns

*This work is supported by Intel Corporation, NSF Young Investigator Award CCR-9357840, and the New York Center for Advanced Technology in Computer Applications and Software Engineering (CASE Center) at Syracuse University. The authors thank 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.

- 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.

Multimedia information systems have been found to be useful in areas such as education, medicine, entertainment and space research. In this paper, we focus on one such application, *media-on-demand* in a distributed environment. This term refers to making it possible for multiple viewers to retrieve multimedia data in real time. We use video data for our purposes. 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 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. For example, 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 directly in hardware. 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 with terabytes of 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. Moreover, the data transfer rates of magnetic disks are low compared to those of other forms of secondary storage. Multimedia information systems are inherently I/O intensive, and it is critical to reduce the ill-effects of this bottleneck. Techniques for doing so are the subject of this paper.

1.2 Related Work

Researchers have proposed various approaches for the storage and retrieval of multimedia data. Anderson et al. [AOG92] have proposed file system design techniques for providing hard performance guarantees. Reddy and Wyllie [ReW93, ReW94] have proposed a disk arm scheduling approach for multimedia data, and characterized the disk-level tradeoffs in a multimedia server. 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 the 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. [RVR92] categorize real time clients into 2 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. Most previous work has concentrated on minimizing rotational and seek overheads in retrieving data. Our approach is to increase the granularity of data retrieved so that the random effects of disk overheads form a smaller fraction of request service time. Moreover, little attention has been paid to the issue of tuning server performance based on user access patterns. [PRP94, LV94] have proposed approaches for *inter-server* information caching in a distributed environment with multiple servers. We have developed techniques for *intra-server* information caching that exploit data access patterns to maximize the number of simultaneous streams that a multimedia server can source.

1.3 Our Research Contributions

In this paper, we propose a model for a server in a distributed video-on-demand application. An integrated approach to the storage and retrieval of video data so as to provide real-time service, is presented. Our model uses parallelism of retrieval to address the problem of the low speed of data transfer from secondary-storage to main memory. Two modifications of the basic retrieval algorithm, the *Local Disk Stream Scheduling (LDSS)* algorithm, and the *Local Memory Stream Scheduling (LMSS)* algorithm are presented.

In order to the analyze the applicability of our model, we have implemented it on the Intel Paragon [Int93] parallel computer. Various parameters affect the server throughput. We performed an extensive performance evaluation of the server operation over a range of parameter values. We studied the effect of varying the buffer space allocated to each stream on the frequency of data retrieval. We studied the effects of varying the stripe factor and the configuration of the server on the intra-server traffic. Based on the results, we identify the important parameters that must be considered in designing a multimedia server, and the various possible tradeoffs.

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 data organization, access and scheduling policies. We present and analyze performance results in Section 5. In section 6, we develop two modifications of the basic retrieval algorithm that improve server throughput by exploiting user access patterns. A dynamic admission control policy is developed in Section 7. Section 8 summarizes the paper.

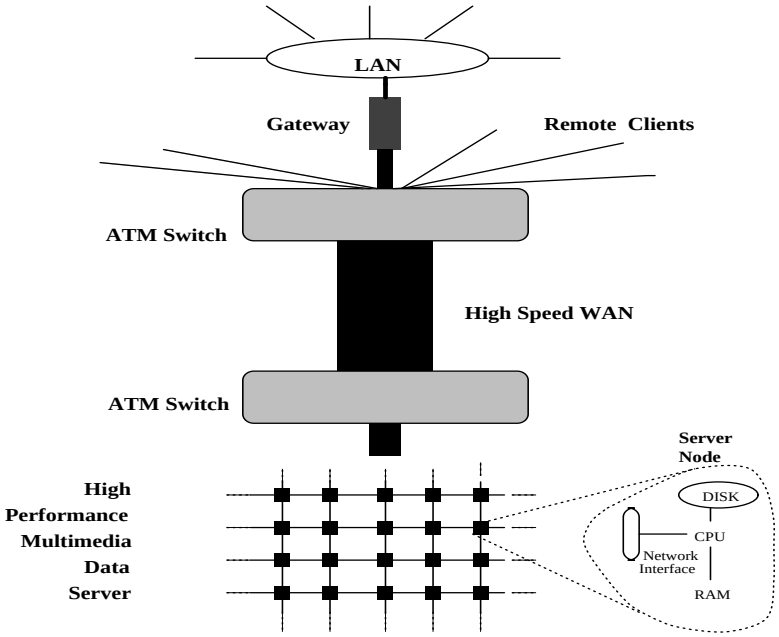

Figure 1: Block diagram of a Distributed Multimedia system

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 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 a high-speed wide-area network, for example, using ATM switches and a fiber optic network. The remote clients are computers with tens of megabytes of main memory and hundreds of megabytes of secondary storage.

2.1 Assumptions regarding the Data

We assume that the data are stored at the server in compressed digital form. As the multimedia industry evolves, standards are being enacted. For instance, the MPEG-1 standard is suitable for a digital video 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. We assume the MPEG-1 standard for the purpose of this paper. 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]. Such intelligent terminals are an 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].

3 The High Performance Multimedia Server

3.1 Architecture

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 and gigabytes of main memory. The server may also be required to perform fast compression of multimedia data. Hence it should have good floating-point and scalar arithmetic performance. A parallel computer with multiple independent nodes interconnected by a high-speed interconnection network is a good candidate for these requirements. Complementary views have been expressed to this effect in the context of high performance relational database systems [Sto86, DeG92].

At the same time, it must be noted that most parallel computers available until 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 currently the topic of vigorous research. A comprehensive survey of the problems in high-performance I/O appears in[RoC94]. 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 conventional parallel processors as well as PC technology. For instance, companies like Oracle and Silicon Graphics advocate the use of powerful parallel computers to build multimedia servers; while companies like Microsoft, Intel and Compaq claim to achieve equivalent functionality at a lower cost by building servers through interconnecting the bulk-produced 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 MPP-like machine or a collection of PCs/workstations interconnected by high-speed links. In this paper, we have used the MPP approach to validate our work. We present our results for the Intel Paragon.

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. 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 cooperate 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.

3.2 Logical Model of the Server

Figure 2 shows a block diagram of the logical view of the proposed server. In the figure, node I_1 is serving a stream whose data is stored on nodes S_1 and S_3, node I_2 is serving a stream whose data is stored on nodes S_1, S_2 and S_4, and node I_3 is serving a stream whose data is stored on nodes S_4 and S_5.

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. 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 is accepted. An interface node to serve the stream is chosen by the object manager, and the interface node then takes over the authority and authority 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

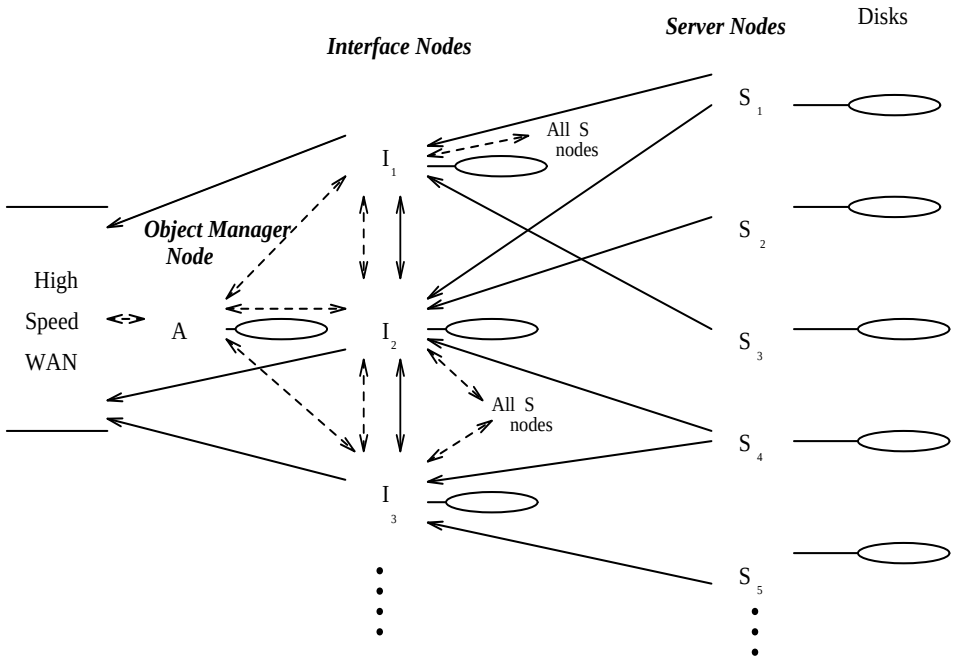

Figure 2: Logical Model of the Server. Example communication patterns are shown: dark lines indicate data, dotted lines indicate control information

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. Before accepting a request, the Object Manager communicates with the selected Interface nodes to ensure that the new request, if accepted, can be successfully served, while at the same time ensuring that existing requests continue to be served at the required rate [1].

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[DK+92],

[1] We have developed algorithms for performing these functions. However, they are not the topic of 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
S	Stripe factor	-

Table 1: The parameters used in this paper

or a number of independent disks for greater I/O throughput.

4 Data Access and Scheduling

4.1 Parameters Used and Scheduling Constraints

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:

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

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

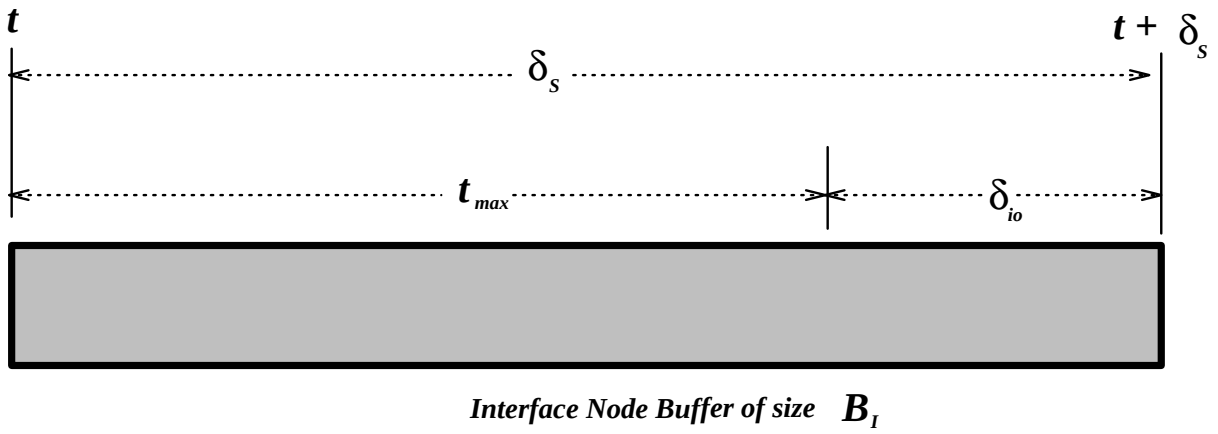

Figure 3: Time relationships of Interface-Server node traffic

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 :

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

This is illustrated in figure 3 . Note that equation 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 processors have large main memories, and using such processors 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 as in[VG+94], and use the maximum values of the disk overheads for admitting users of the latter kind.

5 Performance Evaluation

We have implemented our logical server model on the Intel Paragon parallel computer. The Intel Paragon [Hwa93] is a mesh-based architecture with Intel i860XP microprocessors. Interprocessor communication is done using wormhole routing [NiM93, MTR94]. Due to storage space and availability of real-world data limitations, the disk access part was simulated. We have assumed gi-

Description	*Value*
Required playback rate (R_{pl})	1.5 Mbits/sec
Size of packets sent by an I node (P_I)	64 Kbytes
Minimum disk seek time	1 msec
Time for one rotation	10.1 ms
Average rotational latency	8.03 ms
Evaluation machine	56 node Intel Paragon

Table 2: The parameter values used for the simulation

gabytes of disk space per node, and a disk data transfer rate of 10 Mbytes/sec. Currently available magnetic disks have data transfer rates of a few Mbytes/sec. 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 4 disks to achieve the 10 Mbytes/sec data transfer rate we have assumed above. 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 disk access time was simulated by elapsing the system timer on each server node. The playback time for each stream varied between 4 and 5 minutes, depending on the time of arrival of the request for that stream.

The data retrieval process as explained above, whereby an interface node serving a stream retrieves stripe fragments from the server nodes on which the data is stored in each service round, is called the *Remote Disk Stream Scheduling (RDSS)* algorithm. In the worst case, each user request is for a different data stream; which requires use of the RDSS scheduling technique. Consequently, this is the technique for which the experiments were performed. Two modifications of the RDSS scheme, which exploit user access patterns are explained, and their performance is compared with that of the RDSS scheme in section 6.

The parameters that were varied were the stripe factor (S), the request size to the server nodes (P_S), the buffer size at the interface nodes (B_I), and the ratio of the number of server nodes to the number of interface nodes. Disk retrieval was simulated by assuming that the stripe fragments are stored on the disk using a random placement model [KJ+84]. The load on the server was varied by increasing the number of streams that could be supported *per interface node*, incrementally in units of 5 streams. Readings were taken for 5 to 50 streams per interface node.

We measured the components of stream retrieval for the server. For any stream that is being served, the process of retrieving a set of stripe fragments from the S nodes is made up of a number of

activities. The various time components are, in order, time for the fetch request from the interface node to reach the server node, time that the request has to wait at the server node while requests with earlier deadlines are served, the actual service time to retrieve the data from the disk, and finally, the time for the retrieved data to reach the requesting interface node.

The communication time over the network is the sum of two factors - the network latency in the absence of blocking, and the blocking time due to link contention in the interconnection network. For a given message size and interconnection network, the former is fixed; while the latter depends on the network traffic. The network blocking time was dynamically recorded as follows : for each request sent by an I node to a S node, the time delay between the issuing of the data request and the arrival of the packets from the S node was measured. The sum of all the other delays was subtracted from this round trip request delay to give the network blocking time. Thus, the network blocking time includes the buffering and copying overheads associated with messages, when multiple messages contend for the network (the network communication time in the absence of blocking includes the buffering and copying overhead for one message under ideal conditions).

We present below the detailed experimental results. For each experiment, we have plotted two graphs : the *average* of the delay components over all packets of all streams, and the *maximum* of the delay components over all packets of all streams. The reasons for using these measures are explained in the following subsections along with the performance results.

5.1 Buffering vs. Frequency of Retrieval Tradeoff

In the first set of experiments, we fixed the stripe factor and varied the size of the packets requested from the server nodes, P_S. Due to the periodic nature of media retrieval, the server services multiple clients by proceeding in *service rounds*. During each service round, the server retrieves a sequence of media blocks for each client stream. With reference to our model, a service round corresponds to the retrieval of S media fragments by an interface node, for each stream being served by it, (S is the stripe factor) from the server nodes at which the stream concerned is stored. The retrieved data of each stream is stored in its stream buffer at the interface node serving the stream. The interface node then periodically sends out chunks of media data from the stream buffer to the client so that the playback rate requirements are satisfied. Given this mode of operation, varying the value of P_S, keeping the stripe factor fixed involves a tradeoff between stream buffer requirements at the interface node, and the frequency of issuing fetches to the server nodes. For larger values of P_S, the the data retrieved from the S nodes lasts longer; consequently, longer is the duration of a service round at the I node. Hence, the frequency of media fragment retrieval is lower. On the other hand, larger the value of P_S, larger is the buffering requirement at an I node for each stream.

For this experiment, we used a server configuration of 6 I nodes and 35 S nodes. The graphs in figure 4 show the *average* component delays as a function of number of streams per I node, for P_S equal to 40, 80, 120 and 160 kilobytes respectively, for a stripe factor of 4. The components depicted are, from bottom to top,

1. Network communication time in the absence of blocking, N_C.
2. Disk data transfer time, D_R.
3. Disk seek time, D_S.
4. Disk rotation time, D_R.
5. Network blocking time, N_Q.
6. S node queueing delay, S_Q.

The total service time for a request to a S node for a stripe fragment is the sum of these six components. This is denoted by R_{tot}.

Note that N_C depends only on the value of P_S and the network bandwidth. We benchmarked the Paragon over a number of message sizes in the range of interest to us, and obtained a nearly constant bandwidth of 13.8 Mbytes/sec. The results are not presented here due to space limitations. D_T depends only on the value of P_S. Given a value of P_S, these two components are fixed. The measured disk seek and rotational times (D_S and D_R) were averaged for all the requests. Since media blocks are assumed to be uniformly distributed over the disk surface, these two components vary very little in the graphs (the variation is too small to be noticed in the graphs. Moreover, in many cases depicted, disk seek and rotational latencies are insignificant compared to the other delays). The S node queueing delay, S_Q, is a measure of the time that a request has to wait at a S node before it can obtain service.

We note from the graphs that the total delay in retrieving media blocks, R_{tot}, increases as the workload on the server increases, and that this behavior occurs at all four values of P_S. We also see that, at a given server load, as the granularity of P_S is increased, the total time to retrieve media blocks increases. The reason for this is that N_C and D_T are directly proportional to the value of P_S. Lastly, we observe that among the four values of P_S, for $P_S = 40$ kB, S_Q dominates at higher workloads, while for larger values of P_S, it is the network blocking time that dominates at higher workloads. The former behavior is due to the fact that given a particular S node which stores fragments for a stream being served, for a small value P_S, the frequency of fragment requests observed by the S node for that stream is greater than that for larger values of P_S. Thus, over an interval of time (greater than a service round), the *number of disk requests* for a stream is greater

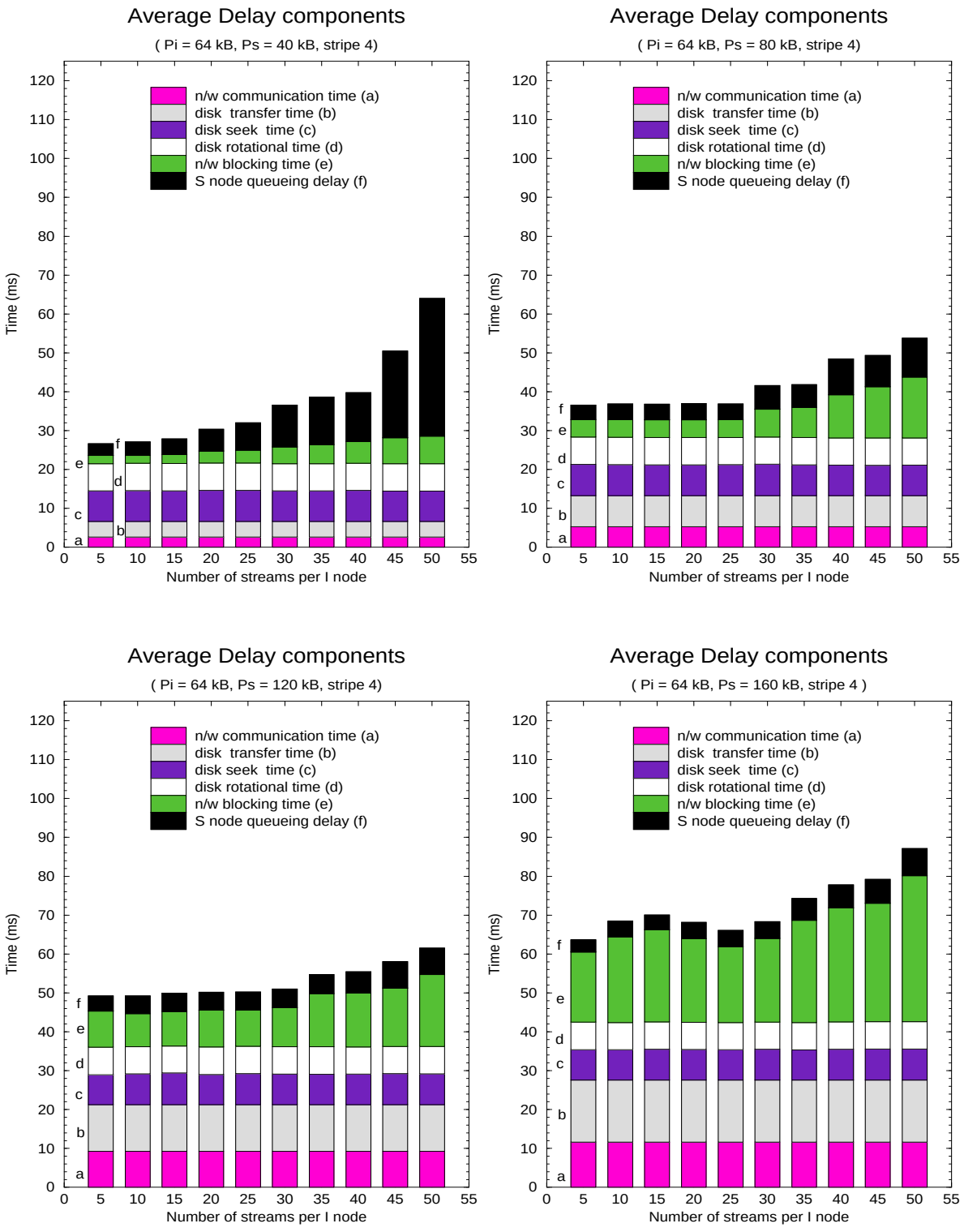

Figure 4: Effect of varying frequency of fetching on average delays. The graphs show the average server delays as a function of the number of streams supported per interface node for 6 I nodes, 35 S nodes, stripe factor of 4. The 4 graphs are for server request sizes of 40 kB, 80 kB, 120 kB and 160 kB respectively.

for a smaller value of P_S. Since each disk access incurs seek and rotational overheads, and their effect is more pronounced for smaller retrieval sizes, the disk utilization is higher for small values of P_S. On the other hand, for large values of P_S (like 160 kB) the frequency of disk accesses is lower, but the network blocking effect dominates due to large message sizes.

Figure 5 shows the *maximum* component delays for the same parameter values as in figure 4. Note that given a value of P_S, the network communication time and the disk transfer time is the same in both the average case and in the maximum case. The motivation for studying maximum component delays is that different clients may have different *quality of service (QOS)* requirements, whereby some clients may be willing to bear occasional loss or long delays of packets, while others may have hard deadlines which cannot be missed. In such a situation, in order to provide hard real-time guarantees for the latter class of clients, it is necessary to design the server so as to minimize instantaneous large delays, and also to minimize the variation in media data retrieval time due to variations in server workloads. Accordingly, depending on the client mix, a design that gives slowly varying maximum retrieval delays at different workloads but higher average retrieval delays may be preferred over a design that provides lower average retrieval delays but may cause wide variations in maximum delays over the range of the anticipated workloads.

In this case, at a stream load of 50, P_S = 40 kB experienced the largest total retrieval delay (R_{tot}), while P_S = 160 kB experienced the smallest total retrieval delay. This is in contrast to the trend in figure 4, where R_{tot} increases as the value of P_S increases at a stream load of 50. One reason for this is that in the average case (figure 4), the network and S node blocking delays (N_Q and S_Q respectively) are comparable in magnitude to the fixed delays (N_C and D_T respectively) at a given value of P_S. Hence the variation of R_{tot} due to variation of P_S is affected by changes in both N_Q and S_Q, as well as by changes in N_C and D_T. On the other hand, in the maximum delay case of figure 5, changes in N_Q and S_Q dominate R_{tot}, causing changes in N_C and D_T to have little or no effect on R_{tot} as the value of P_S is varied. Moreover, at low P_S, S_Q increases faster than it does at high P_S as the workload is increased. Consequently, a low value of P_S gives larger maximum delays than a high value of P_S at high stream loads. Note however, that at low stream loads (of less than 30 streams), as in the average case, smaller values of P_S resulted in lower maximum delay than larger values of P_S at a given stream load. Both these observations can be explained from the summary graphs of figure 6, which show the variation of maximum delays due to network blocking and server queueing with change in the value of P_S, at both low and high stream loads (15 and 50 streams per I node respectively).

With reference to figure 6, observe that at low stream loads, larger the value of P_S, larger is the value of N_Q. On the other hand, at high stream loads, the network blocking effect is more or less constant as P_S varies; this is so because the network gets saturated. Consider now the queueing

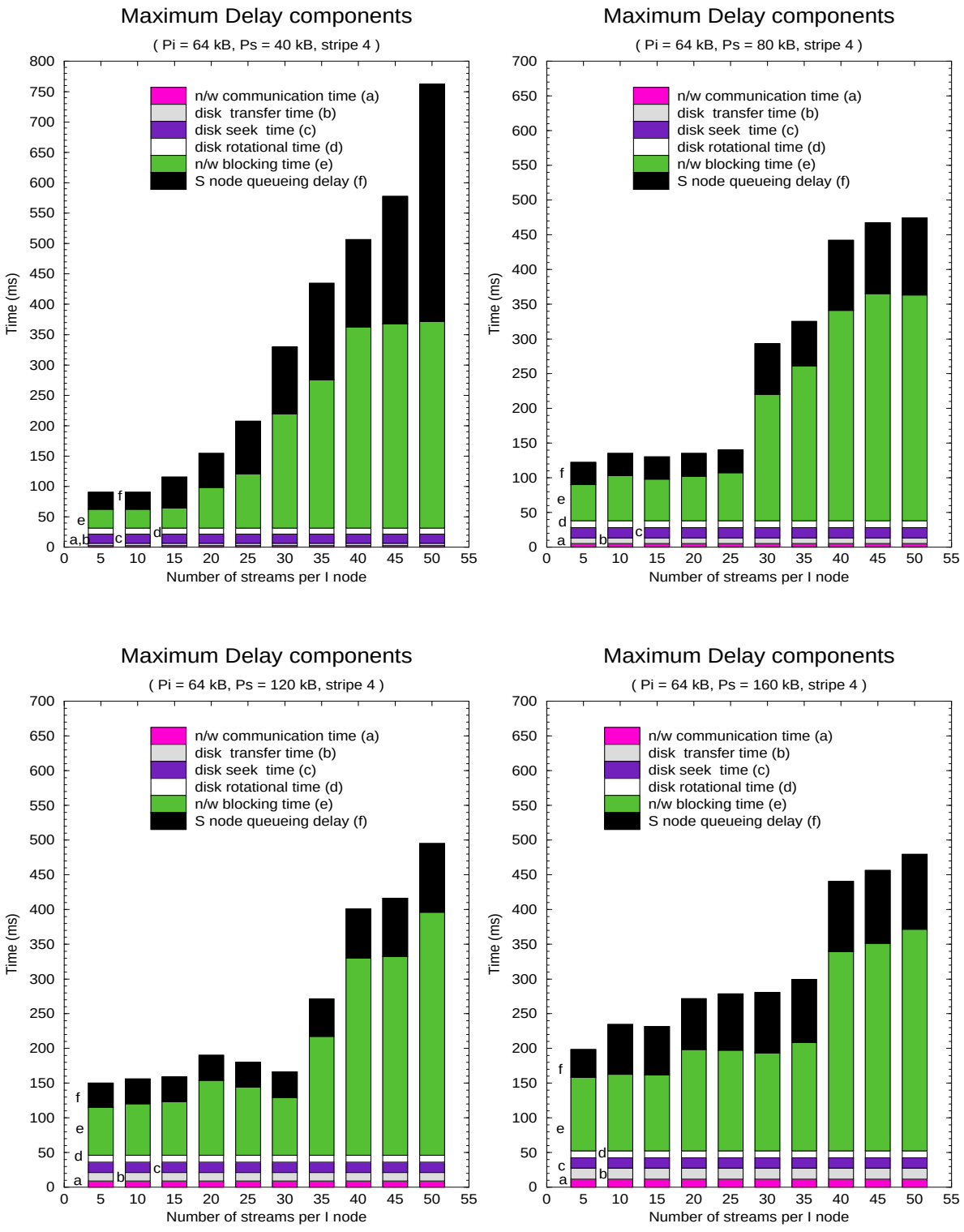

Figure 5: Effect of varying frequency of fetching on maximum delays. The graphs show the maximum server delays as a function of the number of streams supported per interface node for 6 I nodes, 35 S nodes, stripe factor of 4. The 4 graphs are for server request sizes of 40 kB, 80 kB, 120 kB and 160 kB respectively.

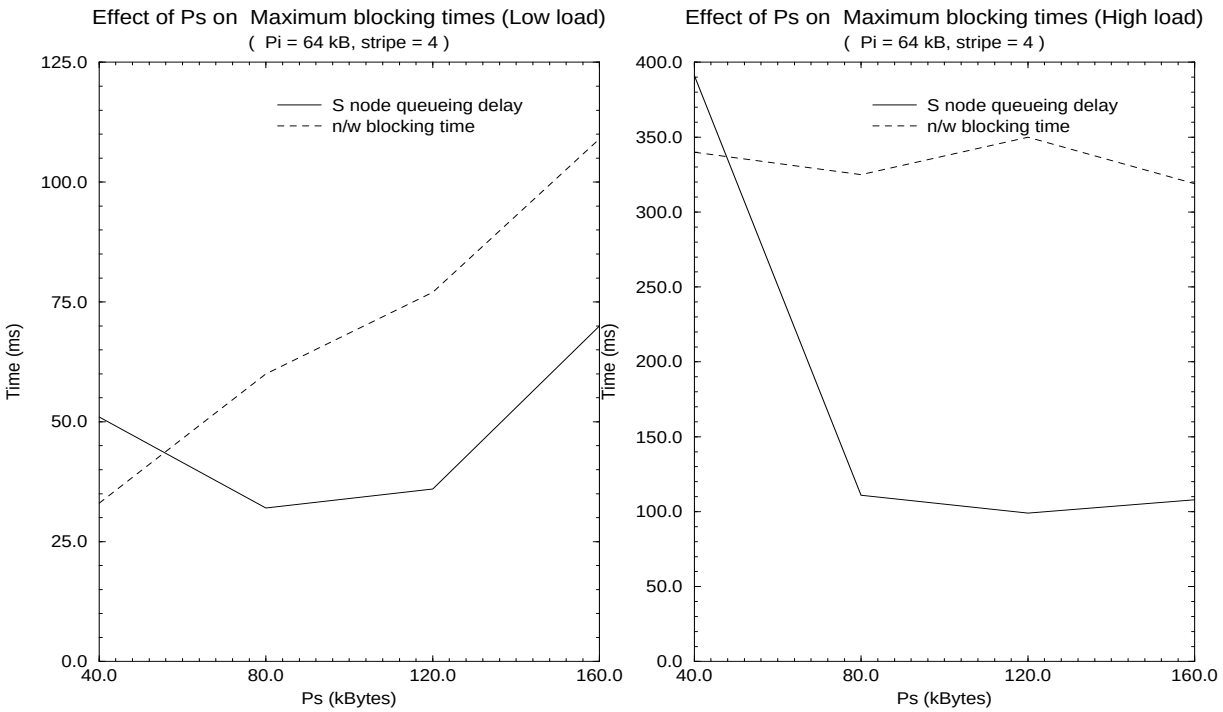

Figure 6: Summary of Effect of varying frequency of fetching on maximum S node queueing and network blocking delays at low and high loads

delay at the servers. At low stream load, the queueing delay for $P_S = 40$ kB is lower than for $P_S = 160$ kB, while that for $P_S = 80$ kB and 120 kB is lower than that for $P_S = 40$ kB. For low P_S and low stream load, the disk is idle most of the time, and hence the delay seen by a service request is low. At high P_S and low stream load also the disk is idle most of the time; however since the transfer time increases, the queueing delay increases. On the other hand, at low P_S and high load, the disk is highly utilized; moreover, the frequency of service requests is high. Since the disks spend a lot of the time seeking and rotating, S_Q for $P_S = 40$ kB is very high. Increasing the value of P_S at high load results in greater S_Q than in the case of high P_S and low load, but the value of S_Q is lower than it is in the case of low P_S and high load. This is so because the frequency of disk fetches is lower at high P_S, hence disk utilization is lower. In conclusion, we note from figure 5, that for $P_S = 80$ kB, the sum of the dominant component delays in the maximum case is close to minimum for *both* low and high stream loads. This suggests that, for the range of values of stripe fragments studied, $P_S = 80$ kB is a good operating point.

5.2 Effect of Stripe factor

In the second set of experiments, we fixed the total amount of data retrieved per service round for each stream, and varied the stripe factor. Thus, the value of P_S varies so that the total amount of data retrieved is constant. In other words, the size of the stream buffer at the interface nodes (B_I) is fixed for each stream. The same configuration of 6 I nodes and 35 S nodes was used. As

before, we collected the average and the maximum component delays as a function of the number of streams served per interface node. Figure 7 shows the results for a stream buffer size of 640 kilobytes and stripe factors of 2, 4, 6 and 8. This corresponds to P_S values of 320, 160, 106 and 80 kilobytes respectively.

In this case we observe that in general, smaller the stripe factor, larger is the total delay for a given number of streams. This is due to the fact that low stripe factor implies large S node request size, which in turn implies larger disk transfer and network communication time. An increase in these times increases R_{tot}. Also, observe that for the low stripe factor of 2, the network blocking time in general is higher than the network blocking time for higher stripe factors. This can be attributed to the large message size (320 kB) for this stripe factor as compared to the message sizes for higher stripe factors.

Figure 8 shows the maximum component delays for the same configuration as above.

In this case, we see that the values of network blocking time (N_Q) and S node queueing delay (S_Q) are much greater than those of the other delays. Hence, the variations in the other delays due to changes in stripe factor are negligible in comparison to the variations in the blocking delays. For a stripe factor of 2, N_Q is more or less constant over the range of stream loads. This can be attributed to the large message size of 320 kB which causes extensive blocking irrespective of load patterns. On the other hand, for a stripe factor of 8, N_Q is low for stream loads below 35 streams, and then increases rapidly. With stripe factor 8, the messages are smaller (80 kB) as compared to the message size for stripe factor 2 (320 kB). Although the *number of messages* in an interval of time equal to a service round is greater for higher stripe factors, their *smaller size* results in smaller maximum delays. However, at higher stream loads, the performance penalty due to the increased number of messages outweighs the advantages of smaller message size. This is so because each message requires scheduling, processing and buffering (copying) overhead. At high loads, these overheads for the larger number of messages cause higher maximum network delays.

The variation of the maximum network and S node queueing delays as a function of stripe factor is depicted in figure 9.

The graph on the left is for a relatively low stream load per interface node (15 streams,) while the graph on the right is for a higher load (50 streams). In both cases, we observe that N_Q is greater than S_Q. S_Q varies little over the range of stripe factors. The reason for this behavior is that the total amount of data retrieved in a service round for a stream is constant, hence it lasts for approximately the same time at the interface node during playout. Consequently, the frequency of requests to the S nodes for any given stream is the same in all four cases. Higher values of stripe factor give a slightly greater S_Q; this is so because the number of S nodes associated with a stream increases, thus increasing the scheduling and processing delay of media retrieval. As explained

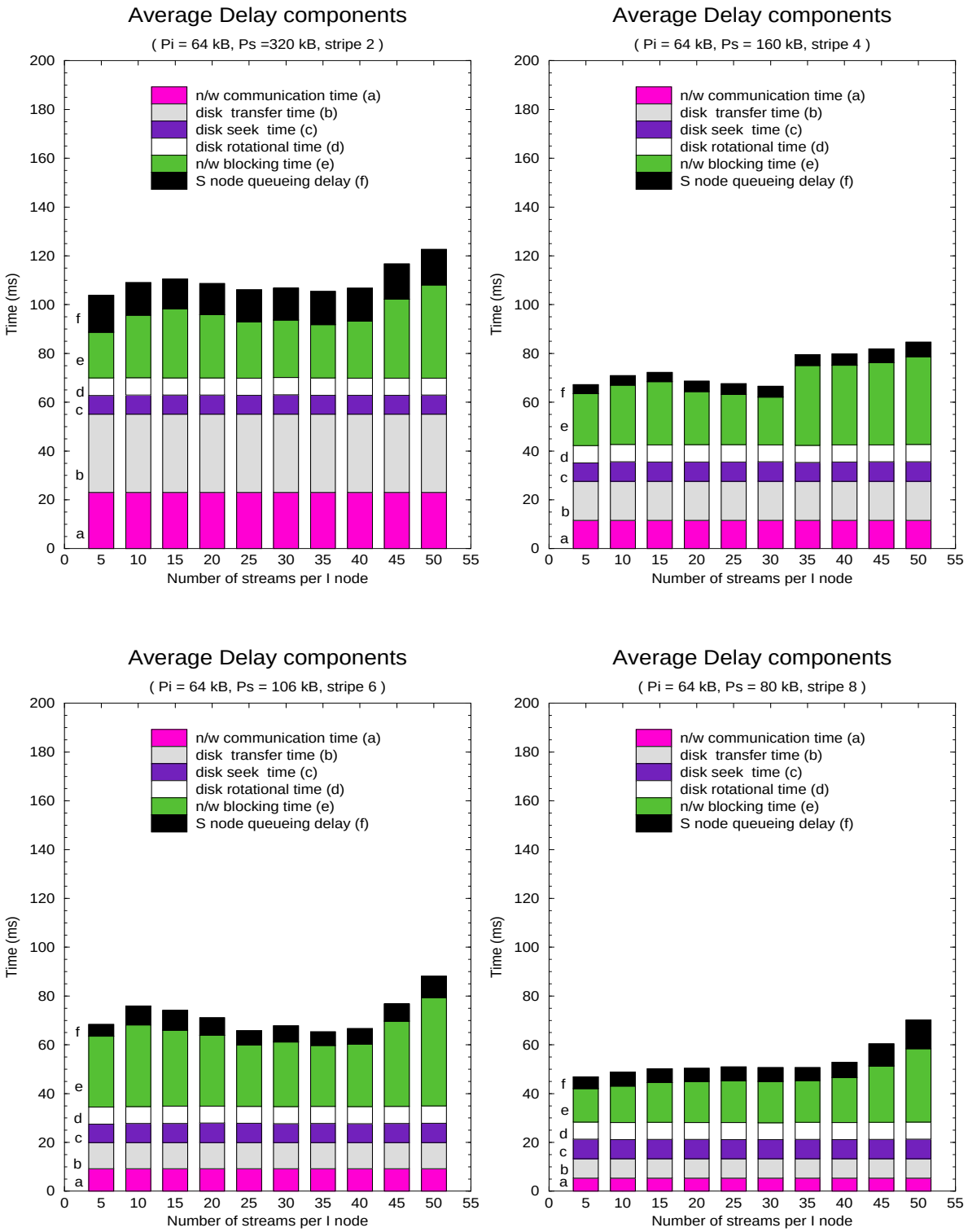

Figure 7: Effect of stripe factor on average delays. The graphs show the average server delays as a function of the number of streams supported per interface node for 6 I nodes, 35 S nodes, and stripe factors of 2, 4, 6 and 8. The corresponding values of P_S are 320 kB, 160 kB, 106 kB and 80 kB respectively.

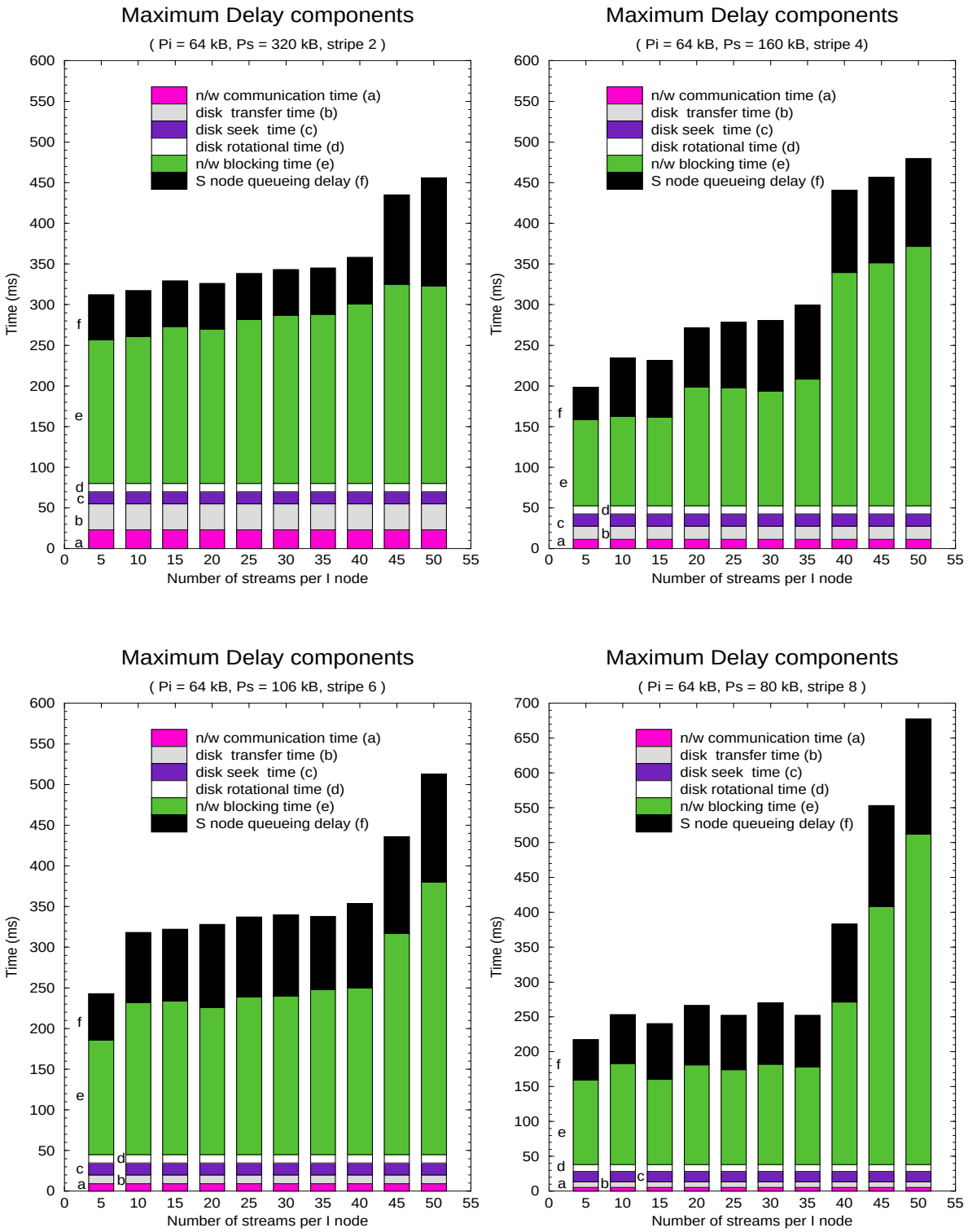

Figure 8: Effect of stripe factor on maximum delays. The graphs show the maximum server delays as a function of the number of streams supported per interface node for 6 I nodes, 35 S nodes, and stripe factors of 2, 4, 6 and 8. The corresponding values of P_S are 320 kB, 160 kB, 106 kB and 80 kB respectively.

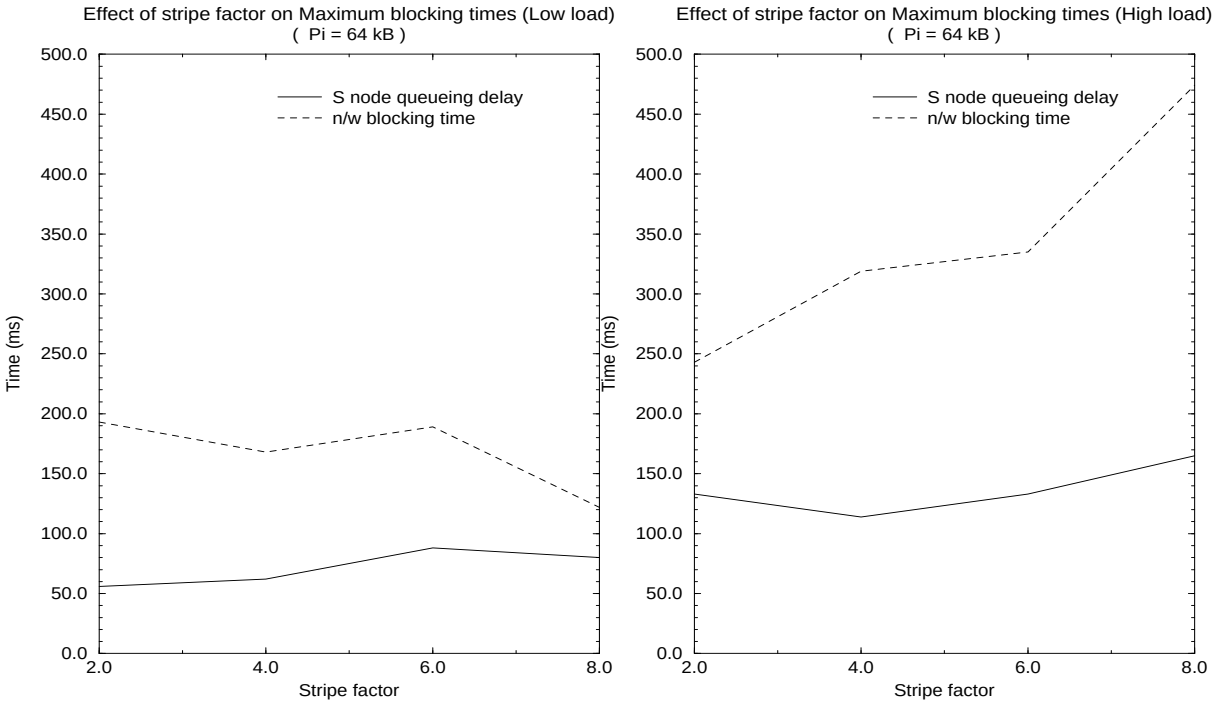

Figure 9: Summary of Effect of stripe factor on maximum S node queueing and network blocking delays at low and high loads

above, increasing the stripe factor causes N_Q to decrease at low stream loads, while increasing the stripe factor causes N_Q to increase at higher stream loads.

5.3 Configuration tradeoffs

In a third set of experiments, we investigated the tradeoffs involved in varying the ratio of interface nodes to server nodes, given a certain number of total nodes. We used a total of 41 nodes, with stripe factor of 4 and a P_S value of 64 kB. In the first case, the server was configured as 8 I nodes and 33 S nodes, while in the second case, it was configured as 6 I nodes and 35 S nodes. Thus the ratio of the number of S nodes to the number of I nodes was approximately 4:1 and 6:1, respectively. Figure 10 shows the component delays as a function of stream load for the two cases. We observe that in the former case, the S node queueing delay is the largest individual component, while in the latter case, the network blocking time is the largest component. This is because the number of S nodes to store media data decreases in the first case, while at the same time the total number of streams that must be served increases. Hence the server nodes become the throughput bottleneck.

6 Exploiting Data Access Patterns

It is natural that certain objects in a database are accessed more frequently than other objects. For example, in a video-on-demand application, it is highly likely that the demand for newly released movies will be higher than that for older movies. We present two modifications of the basic retrieval

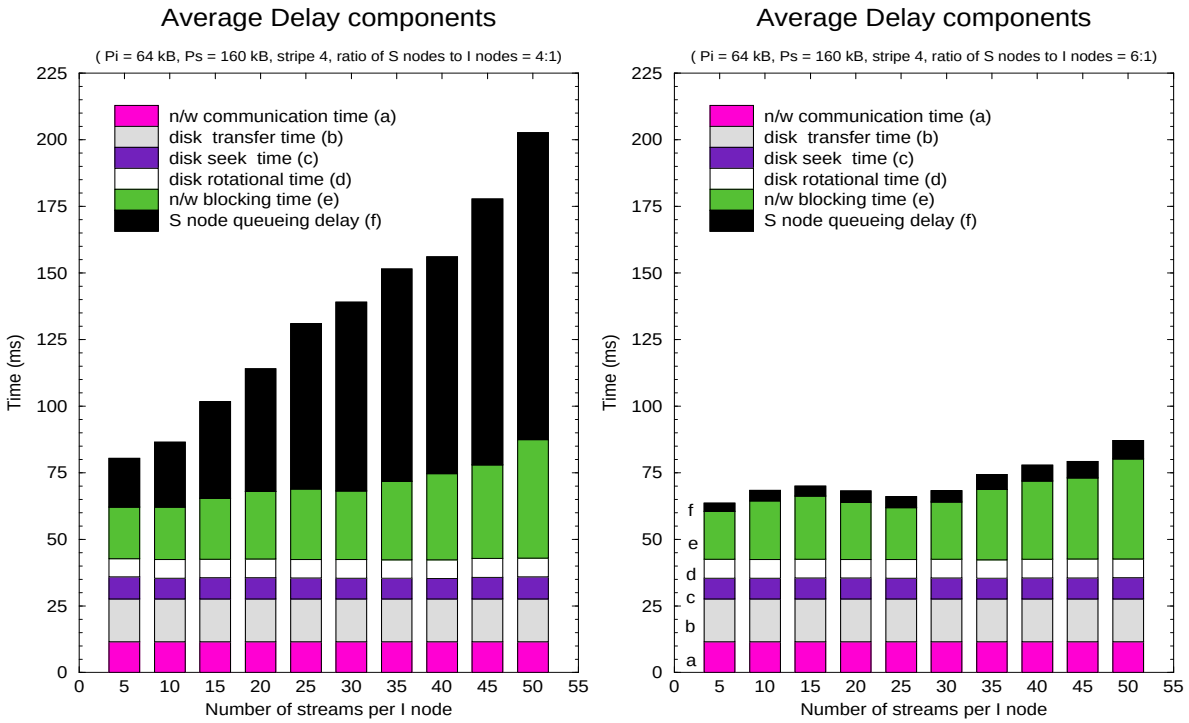

Figure 10: Effect of the ratio of number of S nodes to number of I nodes for the same total number of nodes. Figure on the left is for a ratio of 4 : 1, while that on the right is for a ratio of 6 : 1

algorithm that address this issue. The basic 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.

6.1 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.

6.2 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.

6.3 (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.

6.4 Comparison of RDSS, LDSS and LMSS schemes

We noted the performance of the algorithms for a server configuration of 6 interface nodes and 24 server nodes, and a stripe factor of 4. The composition of the requests was varied as follows : starting from requests for unique media objects, the percentage of requests for the same object was successively increased. Figure 11 shows the maximum number of streams that could be simultaneously supported using each of the three policies.

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.

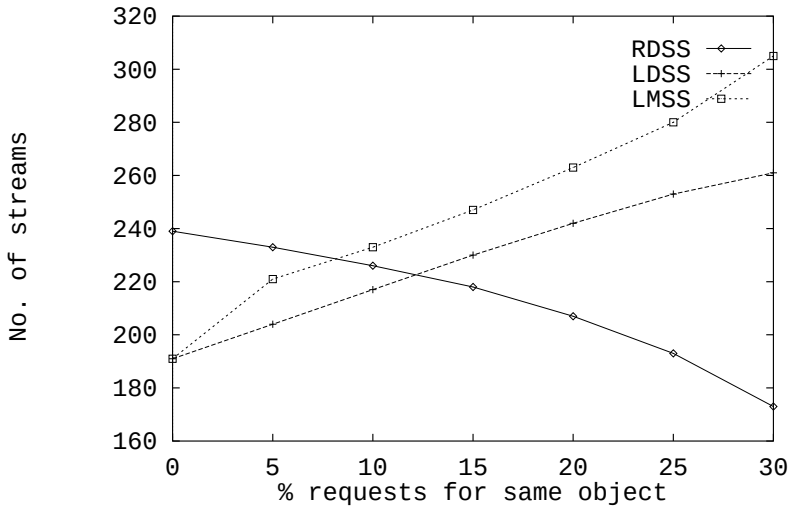

Figure 11: Maximum number of streams that can be supported for each algorithm for 6 I nodes and 24 S nodes, for varying number of requests for the same object, on the Intel Paragon

7 A Dynamic Admission Control Algorithm

In this section we discuss how dynamic knowledge of server workload can be used to develop an admission control algorithm for accepting client requests. When a client requests a stream from the server, the server commits to servicing the client only if it can guarantee the real time bandwidth required by the new stream, while at the same time continuing to serve existing streams without degradation in service. The subsystem of the server's software that is responsible for this decision is called the *admission control policy*.

With reference to the RDSS scheme, equation 2 gives the average theoretical time to retrieve P_S bytes from a S node. However, this equation does not take into account the network and S node blocking times, which have been shown to be critical in determining the retrieval time. To take this effect into account, the server can be modeled abstractly as a *weighted undirected bipartite graph*,citett $G = (I, S, E)$, where I is the set of interface nodes, S is the set of server nodes, and E is the set of edges connecting nodes in I to nodes in S. An edge connecting nodes i and j has a weight w_{ij}, whose value is calculated as explained below. Given this model, an interface node can accept a new client request if the following conditions are satisfied :

7.1 Sufficient Buffer Space

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

buffer space for the new stream :

$$B_{I_{tot}} > \sum_{j=0}^{n} B_{I_j} + B_{I_M} \tag{4}$$

7.2 Sufficient Retrieval Capacity

The weight w_{ij} of edge ij of the graph $G = (I, S, E)$ represents the average time to retrieve P_S bytes from a server node *under existing server workload i.e. at the time of invocation of the admission control algorithm.* This is a more accurate estimation of retrieval time than δ_{io} of equation 2, which gives the average retrieval time in the absence of blocking effects. In order to estimate the value of w_{ij},two possibilities exist. The candidate I node may already be retrieving fragments (in the course of serving existing streams) from some or all of the S nodes storing the data of the new stream. In that case, it can directly estimate the value of w_{ij} for the connection to server node S_j by using past history (for example, the average of the last m time delays to retrieve fragments from S_j). For the remaining S nodes storing the new stream's data, communication is required. Each server node keeps track of the actual time (including the queueing time) taken to retrieve the last m stripe fragments. The average of these m values gives the average fragment retrieval time at that S node. The second component that contributes to w_{ij} is the interconnection network delay. It is the sum of the time to send a request to the S node (δ_{rq}) and the time to transfer P_S bytes over the network (inclusive of the network blocking time). The candidate I node sends a dummy request to the remaining S nodes storing the requested stream. Each S node sends a dummy data packet of size P_S to the I node. The round trip time then gives the network cost of retrieval. In addition, each S node sends the estimated retrieval time in the dummy message to the I node. This time, when added to the measured network time, gives the value of w_{ij}.

The condition to be satisfied for the retrieval capacity can now be stated. The candidate I node determines the value of w_{ij} for each of the S server nodes (recall that S is the stripe factor of a stream) storing the requested streams. For example, figure 12 shows node I_2 (candidate node) trying to determine if it can accept a request for a stream stored on nodes S_1, S_3, S_4, and S_6.

An interface node incurs operating system overhead due to sending and receiving packets, copying data and scheduling transfers. Let the net operating system overhead at the candidate I node be denoted by δ_{ov}. Recall that δ_S denotes the time for which the data retrieved in a service round lasts at an interface node. Then, a candidate interface node i can accept a request for a new client stream if, and only if, the maximum edge weight among all the j server nodes on which the

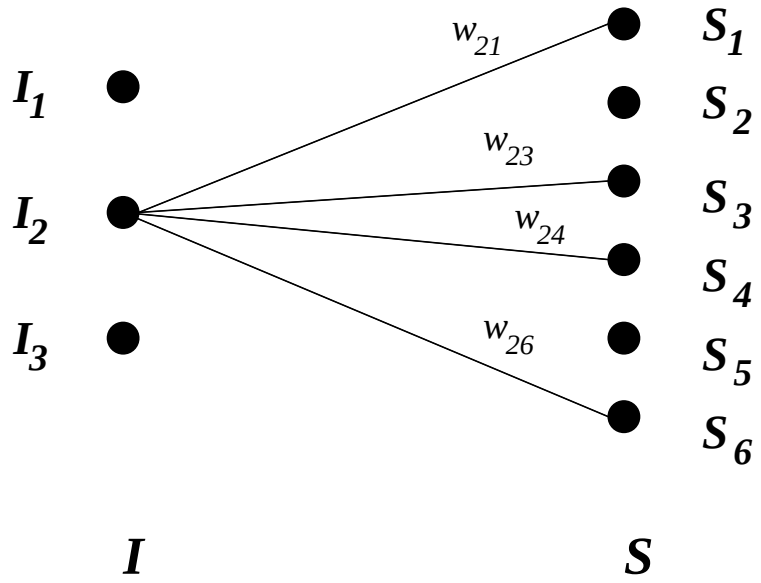

Figure 12: Abstract model of the server - a bipartite graph with two sets of nodes (I and S), with edge weights representing the time to retrieve P_S bytes from a server node. Figure shows admission control policy trying to determine if node I_2 can accept a client request for sourcing a stream whose data is stored on nodes S_1, S_3, S_4, and S_6.

new stream's data is striped does not exceed the permissible retrieval time :

$$max(w_{ij}) < \delta_S - \delta_{ov} \tag{5}$$

The cost of using a dynamic admission control policy such as this one is the overhead of maintaining history of data retrieval delays and communication costs between server nodes to determine retrieval delay when no history exists. Consequently, the length of the interval over which history is maintained is a crucial design parameter. We are in the process of determining suitable values for this parameter.

8 Conclusions

We now summarize the tradeoffs involved in choosing values for the design parameters for a media-on-demand server.

1. **Buffering requirements vs. Frequency of data retrieval.**
   Given a stripe factor, the buffer space needed at an I node to source a client stream is inversely proportional to the frequency of media retrieval from the S nodes. Smaller the buffer space available at an I node for a stream, smaller is the size of the stripe fragments. The average total retrieval time was found to be directly proportional to the value of P_S. While small

values of P_S have the advantages of low buffering requirements and low average total retrieval time, they give the worst *maximum* retrieval times at high stream loads. Large values of P_S result in lower variation in both the average as well as total retrieval times at both low and high loads, and will thus be attractive if sufficient buffer space is available and clients require a uniform (high) quality of service under all load conditions.

2. **Effect of varying stripe factor**

   In practice, the stream buffer space at an I node will be limited. Note that each stream that an I node has to source requires a stream buffer. Hence, smaller the buffer per stream, larger is the number of streams that the server can source. Given a stream buffer of a certain size, there is a tradeoff in the choice of the stripe factor used. We observed that larger the stripe factor, greater is the parallelism of data retrieval and lesser is the average total retrieval time for a stream. A high stripe factor implies that the total number of messages flowing through the network and the number of S nodes serving a given stream increases. This increases the processing, scheduling and buffering overheads. Consequently, high stripe factors result in high *maximum* blocking delays at high stream loads.

3. **Ratio of S nodes to I nodes**

   Economic factors can limit the total number of nodes available to the designer of a multimedia server. Given a fixed number of nodes, interesting tradeoffs are possible in designating the nodes as server nodes or interface nodes. 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. On the other hand, since it is the server nodes that actually store the media data, it is desirable that their number be large also. We showed how a low S to I ratio resulted in higher average total retrieval time compared to a high S to I ratio. We saw that the S node queueing delay is much higher for a low S/I ratio than it is for a high S/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. **Caching Tradeoffs**

   It is natural to assume that all the data objects in a media-on-demand database will not be accessed with the same frequency. The LDSS and LMSS scheduling schemes showed the benefit of being able to dynamically reconfigure the server so that an I node also becomes

a S node. This could be of use in cases where some media objects are accessed with a high frequency. The throughput of the server can then be increased by *migrating* the frequently accessed media object from the S nodes on which it is stored to local disk(s) at an I node. The I node could then be dedicated to serving requests for the popular media object. An even better gain in performance is attained if it is possible to store a substantial part of the data of the popular object in main memory at the interface node. The price that must be paid for the performance gain is operating system overhead and increased complexity of scheduling software for trapping access patterns, migrating and storing the popular object; the possible need for expensive high-performance disk arrays at the interface nodes, and the need for larger main memory at the interface nodes.

In this paper we have investigated and evaluated the effects of varying the design parameters in a media-on-demand server. We have shown that different tradeoffs are possible by choosing different values for these parameters. We should how data access patterns can be exploited to improve server throughput. A dynamic admission control policy that takes existing workload into account was developed. The different components of server operation affect each other in complex ways. In conclusion, we note that the choice of values for the parameters is guided by quality of service requirements of clients, anticipated and actual load conditions, access patterns of clients, quantity and quality of server resources, and economic considerations.

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.

[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.

[ReW94] A. Reddy and J. Wyllie. I/O issues in a multimedia system. *IEEE Computer*, March 1994.

[RVR92] P. V. 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.

[PRP94] C. Papidimitriou, S. Ramanathan and P. V. Rangan. Information caching for delivery of personalized video programs on home entertainment channels. *Proc. of the Intl. Conference on Multimedia Systems and Computing*, May 1994.

[LV94] T. Little and D. Venkatesh. Prospects for interactive video-on-demand. *IEEE Multimedia*, vol. 1, number 3 (Fall 1994).

[VG+94] H. Vin, A. Goyal, et. al. An observation-based admission control algorithm for multimedia servers. *Proc. of the Intl. Conference on Multimedia Systems and Computing*, May 1994.

[DK+92] A. L. C. Drapecu, R. Katz, G. Gibson, et. al. RAID-II: a scalable storage architecture for high-bandwidth network file service. *University of California at Berkeley technical report UCB:CSD-92-672*, 1992.

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

[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.

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

[KJ+84] M. McKusick, W. Joy, S. Leffler and R. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, 2(3), August 1984.

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

[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.

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

[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.

[Liu68] C. L. Liu. *Introduction to Combinatorial Mathematics.* McGraw-Hill Book Company, 1968.