Design and Evaluation of Data Storage and Retrieval Strategies in a Distributed Memory Continuous Media Server [*]

Chutimet Srinilta Divyesh Jadav
ECE Department
Syracuse University, Syracuse, NY 13244
{csrinilt, divyesh}@cat.syr.edu

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

Abstract

High performance servers and high-speed networks will form the backbone of the infra-structure required for distributed multimedia information systems. Given that the goal of such a server is to support hundreds of interactive data streams simultaneously, various tradeoffs are possible with respect to the storage of data on secondary memory, and its retrieval therefrom. In this paper we identify and evaluate these tradeoffs. We evaluate the effect of varying the stripe factor and also the performance of batched retrieval of disk–resident data. We develop a methodology to predict the stream capacity of such a server. The evaluation is done for both uniform and skewed access patterns. Experimental results on the Intel Paragon computer are presented.

1 Introduction

Digitalization of traditionally analog data such as video and audio, and the feasibility of obtaining networking bandwidths above the gigabit-per-second range are two key advances that have made possible the realization, in the near future, of interactive distributed multimedia systems. Multimedia data differs from unimedia data in the diversity of data sizes and the need to provide real-time guarantees for playback. On account of these differences, it is important for a continuous media server to provide efficient data storage and retrieval mechanisms.

1.1 Related Work

Researchers have proposed various approaches for the storage and retrieval of multimedia data. [9] proposed a disk arm scheduling approach for multimedia data, and characterized the disk-level tradeoffs in a multimedia server. [8] proposed a model based on constrained block allocation. [5] increased the effective retrieval bandwidth by striping media data across several disks in a round robin fashion. Various striping tradeoffs have been studied in [5, 1]. [4] studied cost tradeoffs and scalability issues in high performance media-on-demand (MOD) servers. Techniques for improving reliability and availability of the storage subsystem were studied in [2, 3].

1.2 Research Contributions

Due to the large size of multimedia objects , these objects must reside in secondary storage devices almost all the time. Only the portion that is needed for the display at a given point of time is retrieved from the disk. Because disk accesses are involved throughout the playback duration which is usually long, multimedia applications are considered I/O intensive. This paper focuses on storage and retrieval techniques to support real-time playback of continuous media objects. We implemented a strategy to organize objects across the disks and a strategy for their retrieval. We identify the storage and retrieval parameters that affect server performance and evaluate the effect of varying their values. In [7] we considered a restricted retrieval strategy whereby all the nodes across which a media file is striped are accessed in a service round. In this paper we consider a more generalized retrieval model called *batch retrieval*. The performance metrics that we used are the number of concurrent sessions supported by the server and the average storage node queuing delay. We derive bounds on the retrieval capacity of a storage node. This leads to the development of a methodology to *predict* the stream capacity of a distributed memory media server. Due to the fact that some objects, such as newly released movies, may be more popular than others, the frequency of request to each object is different. We studied the effect of storage and retrieval parameters under various request patterns.

[*] This work was supported in part by Intel Corporation, NSF Young Investigator Award CCR-9357840 and NSF grant CCR-9509143. The authors thank the Caltech CCSF facilities for providing access to the Intel Paragon.

1063 7133/97 $10.00 © 1997 IEEE

Proceedings of the 11th International Parallel Processing Symposium (IPPS '97)
1063-7133/97 $10.00 © 1997 IEEE

The rest of this paper is organized as follows. Section 2 explains the architecture of the server along with its data storage, retrieval and buffering strategies. Section 3 outlines the performance metrics. Section 4 discusses the effects of parameters on server performance. The experimental model and results as well as stream capacity prediction methodology is presented in section 5. Section 6 concludes the paper.

2 The Server Model

A parallel machine is a good candidate for a server model because of its ability to serve multiple clients simultaneously, its high disk and node memory, and the parallelism of data retrieval that can be obtained by data striping. In this model, we assume that

- The server is connected to clients by a high-speed wide-area network which delivers data to clients reliably and at the required bandwidth.
- Clients have limited storage space and *hard* deadlines i.e. they cannot tolerate jitter in delivered data. Consequently, the server must retrieve and supply data at almost the same rate as their consumption by clients.
- Objects are stored at the server in compressed digital form. The decompression is done at client site.
- No caching is performed at any node.

The architecture of the proposed server model is based on a shared-nothing model [11], where each of the computing nodes in the system has private memory and peripheral storage. Communication between nodes is achieved by passing messages through the interconnection network. The logical configuration of the server model is shown in Figure 1. The lines between nodes represent data transfer, and the arrows indicate the transfer direction.

The physical server nodes can be classified into three groups, based on their functionality: the object manager node (O), the interface nodes (I) and the storage nodes (S). The object manager receives all incoming requests for objects from clients and delegates the responsibility of serving a request to one of the interface nodes. The interface nodes are responsible for scheduling and serving requests. Their main function is to request data packets from storage nodes, order the packets received, and send the packets to the clients. Storage nodes store multimedia objects on their secondary storage devices. They retrieve and transmit data packets to interface nodes upon request.

2.1 Storage Strategy

Efficient I/O operations are the key to improving the performance of the system as a whole. The way objects are organized over the storage nodes is important because the time to retrieve an object depends primarily on the load on the disk(s), which is proportional to the load on the storage node(s) where the object is stored.

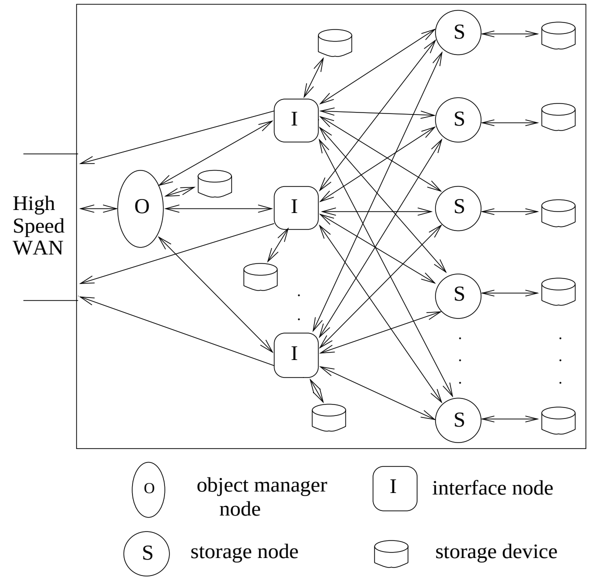


Figure 1. Logical configuration

Striping is used in the server model. Each object is striped and stored across a group of storage nodes. Striping helps distribute the load on the storage nodes as well as the network. Hot-spots and network congestion are avoided, and higher effective bandwidth from parallel data retrieval is obtained. The number of storage nodes across which an object is striped is called the stripe factor (*S*) of the object.

The stripe factor of an object can be any number from one to the total number of storage nodes in the system, but the bandwidth of *S* storage nodes altogether must be at least equal to the playback bandwidth requirement of the object.

Each object is broken down into a sequence of equal size fragments. If the first fragment of an object is stored on node *j*, then the *i*th fragment of an object will be stored at the $((j + ((i-1) \bmod S)) \bmod N)$th storage node, where N is the total number of storage nodes.

2.2 Buffer Space Management

At an interface node, there is a fixed size buffer assigned to each playback session. The size of the buffer is important because it defines the session startup latency as the playback cannot begin until a certain amount of the buffer is filled. Moreover, it also determines the refilling period of the buffer. By using a multi-section buffer, the deadlines can be deferred [10]. All sections in a buffer are used circularly. A 2-sided buffer, where one side is for incoming packets and the other side is for outgoing packets , is used in the model. An interface node manages the buffer in such a way that the refilling and the consuming rates are equal. Thus, the two sides can be used interchangeably.

As data are taken from a buffer at a constant rate—the playback bandwidth of a session—the buffer needs to be refilled at the same rate in order to provide enough data and, at the same time, not overflow the buffer.

2.3 Retrieval Strategy

For each playback session, there are some related terms which are defined below:

i-packet Unit of data transfer from an I node (P_i bytes).

s-packet Unit of data transfer from an S node (P_s bytes).

delivering period (T_d) The time interval between the release of successive i-packets.

refilling period (T_r) The time interval that an interface node requests for data from storage node(s).

round-trip delay (T_{rt}) The time interval between the issuing of a refilling request and the arrival of the corresponding s-packet.

Three retrieval parameters are defined for each playback session:

session buffer size (*b*) The size of buffer space allocated for a playback session at an interface node.

batch size (*B*) The number of requests that an interface node issues in each refilling period.

s-packet size (P_s) The size of one s-packet.

Batch size (*B*) and s-packet size (P_s) have to be chosen in such a way that the data retrieved in each refilling period fit in the buffer space allowed (*b*). *B* can be any number from one to *S* (stripe factor of an object), but the total bandwidth from *B* storage nodes has to be at least equal to the required playback bandwidth.

During a playback session, among *S* storage nodes containing the object, a set of *B* nodes takes turns retrieving data in every refilling period. *B* refilling requests are sent to *B* storage nodes every T_r units of time. In response to these requests, *B* s-packets are retrieved and sent back to the requesting interface node.

Having batch size less than stripe factor enables the multiplexed storage node access in which different sets of storage nodes are accessed during successive refilling periods. Only one set will be active at any given time; thus the inactive sets can service other sessions. This helps avoid a group of storage nodes containing frequently requested objects from becoming bottlenecks of the system.

2.3.1 Deadlines

For each playback session, T_d is equal to $\frac{P_i}{R_{pl}}$. R_{pl} is the playback bandwidth requirement of an object. Thus, for a session buffer of *b* bytes, the full outgoing half will last $\frac{b*T_d}{2*P_i}$ units of time, which, in turn, is T_r. Within T_r units of time after requesting for s-packets, an interface node must receive them in its buffer.

In other words, T_d and T_r define deadlines by which i-packet and s-packet must arrive at the client and the interface node, respectively. The deadlines defined by T_d are called *delivering* deadlines, and those defined by T_r are called *refilling* deadlines. Figure 2 shows a timing diagram for a playback session. T_{rt} of every s-packet must not exceed T_r.

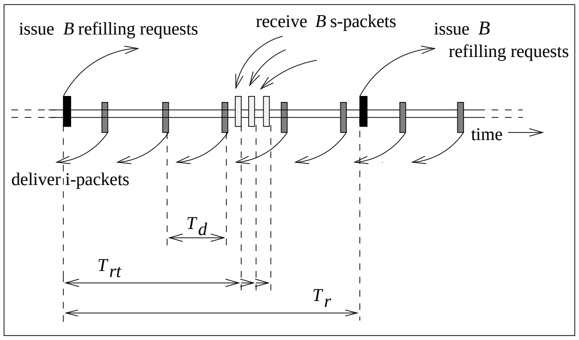


Figure 2. Timing diagram at an I node

2.4 Servicing Multiple Playback Sessions

Due to the periodic nature of media retrieval, the server services multiple sessions by proceeding in service rounds. Each session takes turns getting service according to its deadlines. The time left after taking care of these deadlines is used for receiving s-packets from storage nodes.

3 Performance Metrics

3.1 Maximum Number of Concurrent Sessions

Each session requires a certain I/O bandwidth for continuous playback. Assuming that the interconnection network of the server, and the wide-area network for data delivery are not performance bottlenecks, disk bandwidth limits the number of concurrent sessions that a storage node, and consequently the system, can support.

On average, the effective bandwidth (B_{eff}) of a storage node in retrieving one s-packet is $\frac{P_s}{\delta_{seek_{avg}} + \delta_{rot_{avg}} + \delta_{tr}}$.

The aggregate bandwidth (B_{tot}) provided is equal to $(number\ of\ storage\ nodes) * B_{eff}$.

Assuming that the load is balanced among the storage nodes in the system, at a steady state when every storage node provides service at its maximum capability, the ideal maximum number of concurrent sessions that all storage nodes can support (MAX_{ses}) is $\frac{B_{tot}}{R_{pl}}$. However, after taking into account the communication time between interface and storage nodes, and the overhead at both interface and storage nodes, the actual number of concurrent sessions that the server model can support will be lower.

The other factor that limits the number of concurrent sessions is the deadlines, both delivering and refilling. The ability to maintain the deadlines depends on activities at both interface and storage nodes. The upper bound for the round-trip delay is T_r. T_r is composed of two major parts: time spent at a storage node (T_s) and time spent on the network (T_n). Let $T_{s_{max}}$ denote the maximum time allowed to spend at a storage node. This includes waiting time and disk service time. Considering the average situation, the number of outstanding refilling requests (the request being serviced and the requests waiting in the queue) that a storage node carries must not exceed MAX_{ref} at all times,

$$MAX_{ref} = \frac{T_{s_{max}}}{\delta_{seek_{avg}} + \delta_{rot_{avg}} + \delta_{tr}}. \qquad (1)$$

In case batch size is less than stripe factor, two successive batches of refilling requests will not address the same set of storage nodes. Let *C* denote *cycle*, which is the number of refilling periods between two successive batches of refilling requests (of the same session) addressing the same storage node. *C* is simply calculated by

$$C = \frac{S}{B}. \qquad (2)$$

Since every object has the same *B* and *S*, all requests arriving at a storage node in one cycle belong to different sessions. Their service time altogether cannot exceed the duration of a cycle itself which is *C* refilling periods. Hence, the maximum number of client requests that a storage node can handle (MAX_{req}) is determined by

$$MAX_{req} = \frac{C * T_r}{\delta_{seek_{avg}} + \delta_{rot_{avg}} + \delta_{tr}}. \qquad (3)$$

In summary, there are two constraints at the storage nodes. Each storage node can handle at most MAX_{req} client requests, and the number of outstanding refilling requests must not exceed MAX_{ref} requests at any time.

3.2 Storage Node Queuing Delay

The process of retrieving an s-packet from a storage node is made up of a number of activities. Round-trip delay is composed of disk queuing time, disk service time and network communication time.

Although a storage node serves many sessions simultaneously, disk access can be performed only for one session at any given time. Refilling requests entering a storage node when it is busy servicing another request will be queued up. The time that a request must wait in the queue is equal to the disk service time of all the requests arriving before it. Thus, it is proportional to the number of the requests arriving earlier and the total amount of data that those requests are asking for. As storage and retrieval strategies determine frequency of refilling requests as well as the amount of data to be retrieved, they directly affect the queuing delay. Therefore, we chose storage node queuing delay to evaluate the effect of our storage and retrieval strategies.

4 Parameters Affecting Performance

4.1 Storage Parameter

When the access pattern is uniform, the load on storage nodes roughly balances itself out. For non-uniform access patterns, the stripe factor helps balance the load on storage nodes as it defines the distribution of objects in the system.

The effect of the stripe factor is more significant under non-uniform access patterns. However, striping an object across large number of disks also has a disadvantage in terms of system reliability [2, 3].

4.2 Retrieval Parameters

The three retrieval parameters are dependent. As data are consumed from the buffer at a constant rate, data in a larger buffer will last longer. Hence, the frequency of requests to storage nodes is less, which means that the refilling period is longer. As a consequence, the upper bound for T_{rt} is higher. Since T_r is longer, according to equation 3, a storage node can handle a greater number of sessions.

With the same stripe factor and buffer space, small batch size yields longer cycle. According to equation 3, as long as the number of outstanding refilling requests is less than MAX_{ref} (from equation 1), a storage node supports more client requests when batch size is small.

Even though a smaller batch size yields a greater effective bandwidth, it may cause higher queuing delay, especially when the load on a storage node is high. This is simply because the service time of each s-packet is higher, and thus a refilling request will have to wait longer in the queue.

The load on interface nodes also varies according to the retrieval parameters. With smaller P_s (larger *B*), each interface node sends out more refilling requests as well as receives more s-packets. This increases the load, and thus increases the probability of missing deadlines.

4.3 Access Patterns

In a non-uniform access pattern, some objects are requested more often than others. For a popular object, a large number of requests will arrive at the group of disks that the object is striped across. The increase in popularity of an object basically means that there will be more refilling requests arriving at the disks containing that object.

The interval between two client requests asking for an object is shorter when its popularity is higher. This directly affects the number of requests in the queues. When object is highly popular, the effect of the queuing delay is more pronounced. Moreover, the number of outstanding refilling requests is higher and may exceed the limit (MAX_{ref}) defined by equation 1.

5 Performance Evaluation

5.1 Experimental Model

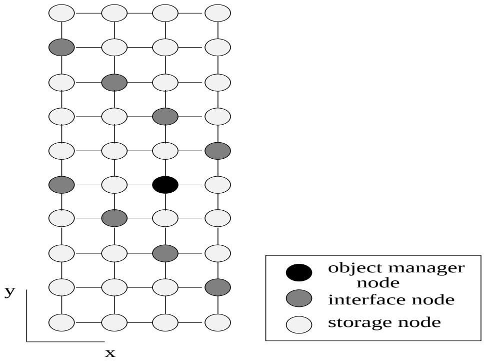


Figure 3. Physical configuration

The server model was implemented on the Intel Paragon parallel computer. A packet is routed from one node to another by traversing in the x-direction and then the y-direction. The experimental model consisted of 40 computing nodes (Figure 3). They were configured as 1 object manager node, 8 interface nodes and 31 storage nodes. The object manager node was placed in the middle of the configuration. The interface nodes were scattered throughout the system in a diagonal shape. The rest of the nodes were assigned as storage nodes. According to the routing scheme, this configuration tended to provide reasonable load distribution on the links. The same configuration was used in every experiment.

5.2 Assumptions

In the experiments, we assumed that all the objects belonged to a single media type, video, with a constant bandwidth requirement of 1.5 Mbits/s (MPEG-1). Every object was assumed to have the same storage and retrieval parameters. Disk access operations were simulated. The disk seek time and rotational latency were generated at run-time using a random number generator. The disk transfer time varied according to the s-packet size. Then, the total disk service time was simulated by elapsing the system timer.

5.3 Experiments

The main objective is to study the effect of storage and retrieval parameters under various access patterns. The combinations of storage and retrieval parameters used in the experiments are shown in Table 1.

Table 1. Storage and retrieval parameters
(x is the combination under which an experiment was conducted)

Set No.	Retrieval Parameters:		Storage Parameter: S			
	b	B, P_s	1	4	16	31
1	2MB	1, 1MB	x	x	x	x
2		2, 512KB		x	x	x
3		4, 256KB		x	x	x
4	1MB	1, 512KB	x	x	x	x
5		2, 256KB		x	x	x
6		4, 128KB		x	x	x

The values of the fixed parameters used in the experiments are shown in Table 2.

Table 2. Fixed parameters

Description	Value
Required playback rate (R_{pl})	1.5 Mbits/s
Size of an interface node packet (P_i)	64 Kbytes
Maximum seek time ($\delta_{seek_{max}}$)	10 ms
Average seek time ($\delta_{seek_{avg}}$)	6.97 ms
Maximum rotational latency ($\delta_{rot_{max}}$)	15 ms
Disk data transfer rate (δ_{tr})	10 Mbytes/s

Four access patterns were given to each set: one uniform access pattern (pattern A) and three non-uniform access patterns (patterns B, C and D). In pattern A, each object was equally likely to be requested. 10, 50 and 60 percent of the client requests in patterns B, C and D, respectively, were for a single object. The load on the server model was varied by increasing the number of sessions from 80 to the maximum that the server could support.

The playback duration of a session varied between four to five minutes, depending on its starting time.

The performance metrics are the number of concurrent sessions supported by the server model and the average storage node queuing delay.

5.4 Methodology for Predicting Stream Capacity

In every experiment, storage nodes that have a high possibility of becoming overloaded are those containing the popular object. These nodes are referred to as *popular nodes*.

For non-uniform access patterns, both popular object requests as well as regular object requests may request objects residing at the popular nodes. The number of client requests asking for objects residing at the popular node (R_{pop}) is determined by

$$\begin{aligned} R_{pop} &= req_{pop} + \frac{S}{s\_nodes} * req_{reg} \\ &= pop * req_{tot} + \frac{S}{s\_nodes} * req_{reg} \qquad (4) \end{aligned}$$

where, req_{pop} is the number of popular requests,
req_{reg} is the number of regular requests,
req_{tot} is the total number of clients' requests, and
pop is percentage of popular requests.

Table 3 shows the number of client requests addressed to popular node under access patterns B.

Table 3. R_{pop} under access pattern B

req_{tot}	req_{pop}	Requests to a Popular Node (R_{pop})			
		$S = 1$	$S = 4$	$S = 16$	$S = 31$
80	8	10.32	17.29	45.16	80
320	32	41.29	69.16	180.65	320
560	56	72.26	121.03	316.13	560
800	80	103.23	172.90	451.61	800
1040	104	134.19	224.77	587.10	1040

The maximum numbers of s-packets that a storage node can retrieve in a refilling period (MAX_{pkt}) under various P_s's are shown in Table 4.

Table 4. MAX_{pkt} in one refilling period

session buffer	T_r (ms)	MAX_{pkt} at P_s (in KB)			
		128	256	512	1024
1 MB	2796.2	99.86	69.90	43.02	24.32
2 MB	5592.4	199.73	139.81	86.04	48.63

For the given storage and retrieval parameters, and access pattern, the maximum number of sessions supported by the server model can be estimated as follows:

1. Find the corresponding MAX_{pkt} from Table 4.

2. Multiply MAX_{pkt} with the cycle (*C*), which is determined from equation 2 to obtain MAX_{req}.

3. Find the closest match between MAX_{req} and the values in the access pattern table (Table 3 for pattern B) under the corresponding stripe factor.

4. The total number of requests in the row where the match is found is the estimated maximum number of sessions that the system can support.

For example, under the configuration where *S* = 4, *b* = 2 MB, *B* = 2, and access pattern B, P_s is 512 KB and *C* is 2. From Table 4, MAX_{pkt} is 86.04. MAX_{req} is 172.08 (2*86.04) which is close to the value in the fourth column of the fourth row in Table 3. Thus, the given configuration should be able to handle about 800 concurrent sessions.

5.5 Experimental Results

Figure 4 shows the maximum number of concurrent sessions that the experimental model supported under various storage/retrieval parameters and selected access patterns.

Configurations with either smaller batch sizes or larger stripe factors generally support more concurrent sessions. However, there is an exception under access pattern D where session buffer is 2MB and stripe factors are 16 and 31. Configurations with batch size of 1 handle fewer sessions compared to those with batch sizes of 2 and 4. This is the effect of access pattern that the number of outstanding refilling requests exceeds the limit when batch size is small and popularity of the object is high.

Situations where batch size or stripe factor does not significantly effect the number of concurrent sessions supported occur when the access patterns are originally uniform or made uniform by using high stripe factors. The maximum number of concurrent sessions supported are 960-1040 sessions, independent of batch sizes and stripe factors. In addition, these values are lower than the estimated values. This means that the storage nodes did not reach their limits. One reason could be that the interface nodes could be overloaded. Another reason is that, we have not considered the performance limitations imposed by traffic load on the interconnection network and the overhead of scheduling at the interface nodes.

When other parameters are the same, configurations with larger session buffers generally support more concurrent sessions under every access pattern.

With the same storage and retrieval parameters, the model supports more sessions when access pattern is more uniform. The result is significant when stripe factor is 4.

When storage and retrieval strategies are not applied, i.e., when stripe factor is 1 or when batch size is equal to stripe factor, the model supports fewer concurrent sessions. This is more noticeable when the access pattern is less uniform.

For a session buffer of 2MB, the average storage node queuing delay per s-packet under various retrieval/storage parameters and selected access patterns is shown in Figure 5.

Under every access pattern, the average storage node queuing delay increases as the number of playback sessions increases. It is interesting that under non-uniform access patterns while the number of sessions is increasing, at some points (referred to as *knee points*), the delay suddenly jumps

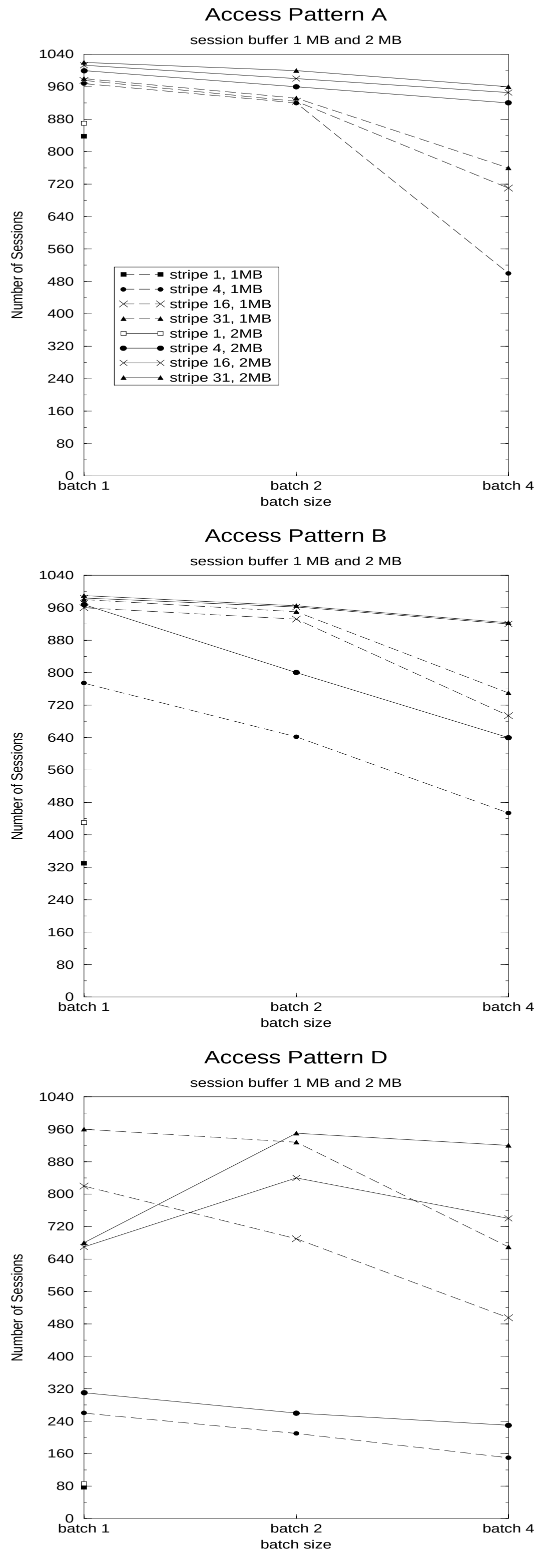


Figure 4. Maximum n umber of concurrent sessions supported by the server model

to a very high value. This is because the number of sessions currently supported by a storage node is reaching the maximum allowable value. Under the same access pattern, before the *knee points*, the average storage node queuing delay is almost the same for all the combinations of storage and retrieval parameters.

Under different access patterns where other parameters remain the same, *knee points* occur at a lower number of sessions when the access pattern is less uniform. This is because less uniform access patterns put more refilling requests in the queues at popular nodes , and thus the average waiting time is higher.

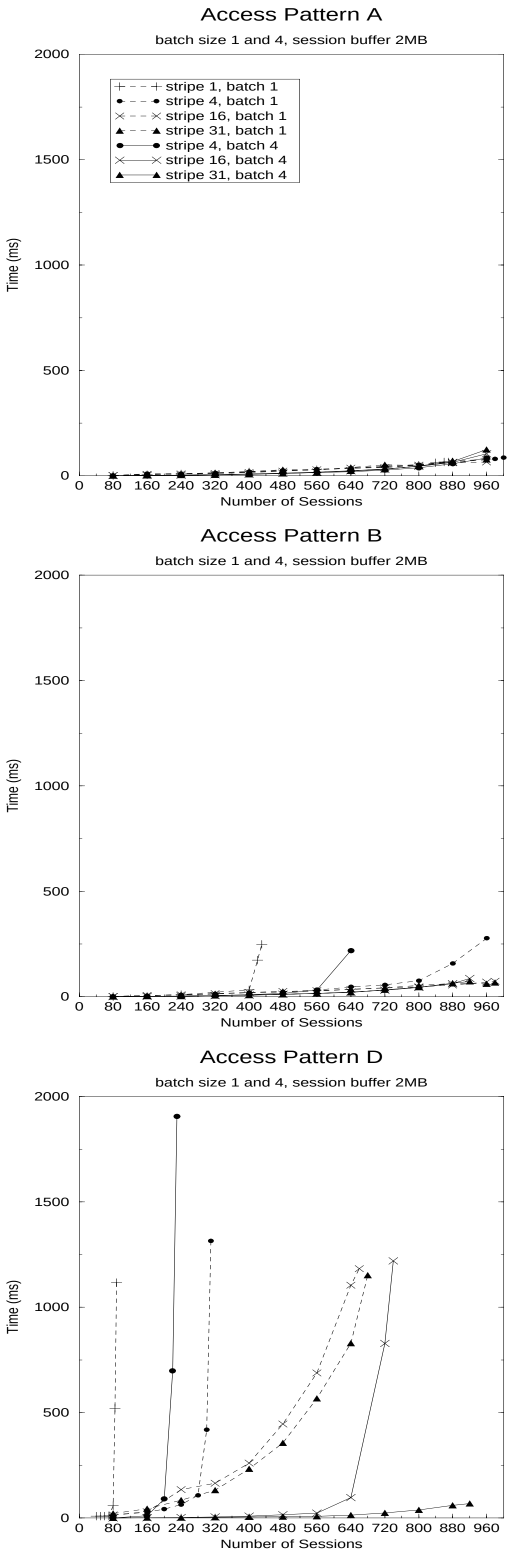
The effect of the stripe factor is prominent under non-uniform access patterns. With the same batch size and access pattern, configurations with larger stripe factors generally yield lower queuing delay under the same total number of sessions. In addition, under the same non-uniform access patterns, *knee points* occur at a lower number of sessions for the configurations with smaller stripe factors. This is because objects are not distributed enough when stripe factors are small. Configurations with larger stripe factors tolerate higher skewed (less uniform) access pattern.

With any given stripe factor, configurations with smaller batch sizes yield lower queuing delay under access patterns A and B. For the other patterns, when *S* is equal to 16 and 31, the queuing delay when *B* = 1 is higher than when *B* = 2 and 4. This clearly shows the effect of access pattern. The effect is more noticeable when *B* is 1 because at this value of *B*, P_s is large; therefore, the service time for each s-packet is long. As a consequence, s-packets spend a longer time waiting in the queue.

Under the same batch size and access pattern, the average storage node queuing delay for configurations with session buffer of 1MB is generally lower than that of configurations with session buffer of 2MB. This happens because with the same batch size, P_s is higher when session buffer is larger. Therefore, the disk service time for one s-packet is longer, resulting in a longer queuing delay. With session buffer of 1MB, the delay changes similarly to that of configurations with buffer of 2MB.

6 Conclusions and Future Work

The experimental results show that with the proposed storage and retrieval strategies, the system handles a large number of concurrent sessions under all access patterns. The configurations with high stripe factor and low batch size supported the highest number of sessions over a wide range of other parameter values. Various combinations of retrieval parameters can be set; the combinations with either higher buffer sizes or smaller batch sizes (larger s-packet sizes) generally yield lower storage node queuing delay and support more concurrent sessions.



Access Pattern A

batch size 1 and 4, session buffer 2MB

Access Pattern B

batch size 1 and 4, session buffer 2MB

Access Pattern D

batch size 1 and 4, session buffer 2MB

Figure 5. Average storage node queuing delay per s-packet

The access pattern also affects the storage node queuing delay and the maximum number of concurrent sessions supported. When the pattern is more skewed, the system incurs higher queuing delay and handles fewer sessions. This effect is significant when the stripe factor is small.

We developed a methodology to predict the maximum number of concurrent streams that the server model can support. In many cases, the observed stream capacity is less than the predicted stream capacity. One of the reasons for this discrepancy is that in the derivation of the aggregate server stream capacity, we did not consider the dynamic load on the interconnection network of the distributed memory computer. This issue was addressed in [6], and we intend to combine the algorithms developed therein with the lessons learned from this paper to develop a more accurate server stream capacity prediction methodology.

References

[1] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered striping in multimedia information systems. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management*, volume 23, June 1994.

[2] S. Berson, L. Golubchik, and R. R. Muntz. Fault tolerant design of multimedia servers. *Proceedings of the ACM 1995 Intl. Conference on the Management of Data*, pages 364–375, May 1995.

[3] A. Dan, M. Kienzle, and D. Sitaram. A dynamic policy of segment replication for load balancing in video-on-demand servers. *ACM Multimedia Systems Journal*, 3(3):93–103, 1995.

[4] C. S. Freedman and D. J. DeWitt. The spiffi scalable video-on-demand system. *Proceedings of the ACM 1995 Intl. Conference on the Management of Data*, pages 352–363, May 1995.

[5] S. Ghandeharizadeh and L. Ramos. Continuous retrieval of multimedia data using parallelism. *IEEE Trans. on Knowledge and Data Engineering*, 5(4), August 1993.

[6] D. Jadav, A. Choudhary, and P. B. Berra. Techniques for increasing the stream capacity of a multimedia server. In *Proc. of Multimedia Storage and Archiving Systems : SPIE's Intl. Symposium on Voice, Video and Data Communications*, Boston, November 1996.

[7] D. Jadav, A. Choudhary, P. B. Berra, and C. Srinilta. An evaluation of design tradeoffs in a high performance media-on-demand server. *ACM Multimedia Systems Journal*, January 1997.

[8] P. V. Rangan and H. M. Vin. Efficient storage techniques for digital continuous media. *IEEE Trans. on Knowledge and Data Engineering*, 5(6), August 1993.

[9] A. L. N. Reddy and J. Wyllie. Disk-scheduling in a multimedia i/o system. *Proceedings of the 1st ACM Intl. Conference on Multimedia*, August 1993.

[10] A. L. N. Reddy and J. Wyllie. I/o issues in a multimedia system. *IEEE Computer*, March 1994.

[11] M. Stonebraker. The case for shared nothing. *Database Engineering*, 9(1), 1986.