SUPPORT FOR FULLY INTERACTIVE PLAYOUT IN A DISK-ARRAY-BASED VIDEO SERVER

Ming-Syan Chen, Dilip D. Kandlur and Philip S. Yu

IBM Research Division
Thomas J. Watson Research Center
Yorktown Heights, New York 10598
{mschen, kandlur, psyu}@watson.ibm.com

ABSTRACT

In a video–on–demand (VOD) system, it is desirable to provide the user with interactive browsing functions such as "fast forward" and "fast backward." Typically, these functions impose additional resource requirements on the VOD system in terms of storage space, retrieval throughput, network bandwidth, etc. Moreover, prevalent video compression techniques such as MPEG impose additional constraints on the process since they introduce inter-frame dependencies. In this paper, we devise methods to support variable rate browsing for MPEG-like video streams and minimize the additional resources required. Specifically, we consider retrieval for a disk-array-based video server and address the problem of distributing the retrieval requests across the disks.

Our overall approach for interactive browsing comprises (1) a storage method, (2) placement and sampling methods, and (3) a playout method, where the placement and sampling methods are two alternatives for video segment selection. These alternatives address two distinct scenarios: (a) providing the browsing functions for any desired browsing speed, and (b) providing the browsing functions for some specific speedup rates. The segment sampling scheme supports browsing at any desired speed, while balancing the load on the disk array. The scheme also minimizes the variation on the number of video segments skipped between samplings. On the other hand, the segment placement scheme supports completely uniform segment sampling across the disk array for some specific speedup rates. Finally, we describe experimental results on the visual effect of the proposed frame skipping approach.

Index Terms — Multimedia, interactive browsing, video-on-demand, disk-array-based video server.

1 Introduction

Rapid advances in computing, storage, and communications technology have made possible the creation of exciting new applications such as video–on–demand. In a video–on–demand (VOD) system multimedia streams are stored on a storage server (the video server) and played out to the user station upon request. The multimedia streams (videos) consist of compressed video and audio, where the prevalent standard for the compressed video/audio is ISO MPEG [1]. Inter-frame compression techniques such as MPEG provide significant advantages in storage and transmission, and consequently they are universally accepted for VOD applications. In order to facilitate storage and retrieval, the MPEG standard defines a compressed stream whose rate is bounded.

The video server must provide the capacity to store a large number of videos and each video occupies approximately 1–2 GBytes of storage. Therefore, the video server is usually constructed as a hierarchy of storage media that includes semiconductor memory, DASD, and tertiary mass storage such as optical jukeboxes [2, 3, 4]. Of these, DASD and tertiary storage form the bulk of the storage system. During normal playout, data blocks belonging to the multimedia stream are retrieved from the storage system, temporarily buffered in memory, and transmitted to the receiving station. The receiving station decodes the incoming stream and plays it out. Typically, the receiving station has a small buffer to compensate for network jitter and variation in the compressed video frame size. In order to ensure uninterrupted playout the VOD system must allocate resources at every stage of the pipeline: storage system read bandwidth, memory buffers, processing bandwidth, and network bandwidth.

In a VOD system, in addition to providing the basic "start/stop" functions, it is highly desirable to provide the user with VCR–like search or scan functions such as "fast forward" (FF) and "fast backward" (FB). There are several possible approaches to implementing these functions, some of which mimic the scan operation of an analog VCR or movie projector. However, as explained below, each of these approaches imposes additional resource requirements on the system. As an example, in the following discussion we assume that the video has to be scanned (forward) at 3X of the normal playout rate.

- The multimedia stream is retrieved and transmitted at 3X of the normal playout rate, and the end station filters and plays out the data. It is apparent that this solution requires additional resources (3X normal) at the storage system, the memory buffers, and the network. Moreover, it requires additional resources at the end station to process the incoming data.

- The storage system retrieves and transmits every third frame to the end station. This scheme also requires significant additional system resources since the multimedia file must now be indexed to retrieve individual frames and the amount of retrieved data is higher than normal due to the structure of the inter-frame coding (i.e., inter-frame dependencies in MPEG; see Section 2 for more details). In addition, the retrieval process for these frames may not be efficient since it is difficult to ensure individual frame alignment with track/block boundaries on the disk.

- The system switches over to a separately coded "scan forward" stream to provide the scan operation. This solution eliminates any additional read bandwidth or network bandwidth. However, it is extremely expensive in terms of storage space and inflexible in that it supports a fixed scan rate.

In this paper, we address the problem of supporting variable rate video browsing operations on inter-frame compressed video streams such as MPEG. We note that in a conventional VCR, the FF or FB speed is fixed and supported by the mechanical rotation. However, the capability of varying the FF/FB speed is highly desirable, as viewers often want to scan the video with a very high speed at the beginning and gradually reduce the speed when they reach closer to the target scene. The desired speed may also depend upon the viewer's familiarity with the video, the peculiarity of the target scene, and the personality of the viewer.

Consequently, we devise methods to support variable rate browsing for MPEG-like video streams in a disk-array-based video server. The proposed methods not only support variable rate browsing, thus providing more versatility to the viewers than the conventional VCR, but also satisfy the constraints of the decoder and require a minimum of additional system resources. We specifically consider retrieval for a disk-array-based video server whose model is shown in Figure 1. We consider two cases: (1) providing FF functions for any desired browsing speed, and (2) providing FF functions for some specific speedup rates. In the first case, a segment sampling scheme is presented where a segment consists of a group of video frames which have inter-frame dependencies. This scheme minimizes the variation on the number of segments skipped between every two consecutive samplings while balancing the load on the disk array.

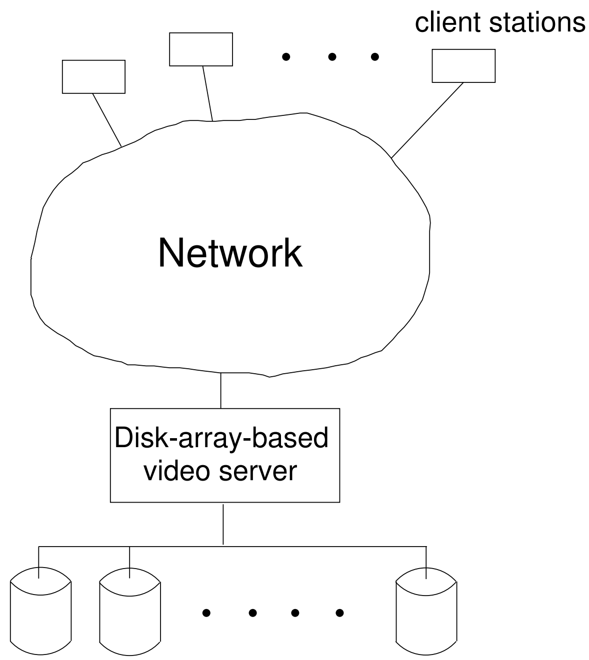

Figure 1: A disk-array-based video server.

In the second case, a segment placement scheme is described which completely eliminates any variation in the number of segments skipped and supports uniform sampling across the disks. Overall, our approach for interactive browsing comprises (1) a storage method which divides a video stream into video segments, (2a) a segment sampling scheme, (2b) a segment placement scheme, and (3) a playout method, where (2a) and (2b) are two alternatives for segment selection (see Figure 2).

It is noted that the implementations for interactive browsing considered here are not a direct emulation of VCR operations. While our schemes have several advantages in terms of resource usage and are especially relevant for inter-frame compressed video, it is important to ascertain whether the proposed segment skipping approach is visually acceptable to end users. To investigate this issue, we have implemented a prototype that creates and plays out video streams with segment skipping. Our initial experiments with this prototype have shown that segment skipping is a viable approach to video browsing.

There has not been much work reported on supporting interactive browsing in a disk-array-based video server. [2] presents alternatives for providing FF in an optical storage system. Their work does not consider the use of a disk array, let alone the problem of balancing the

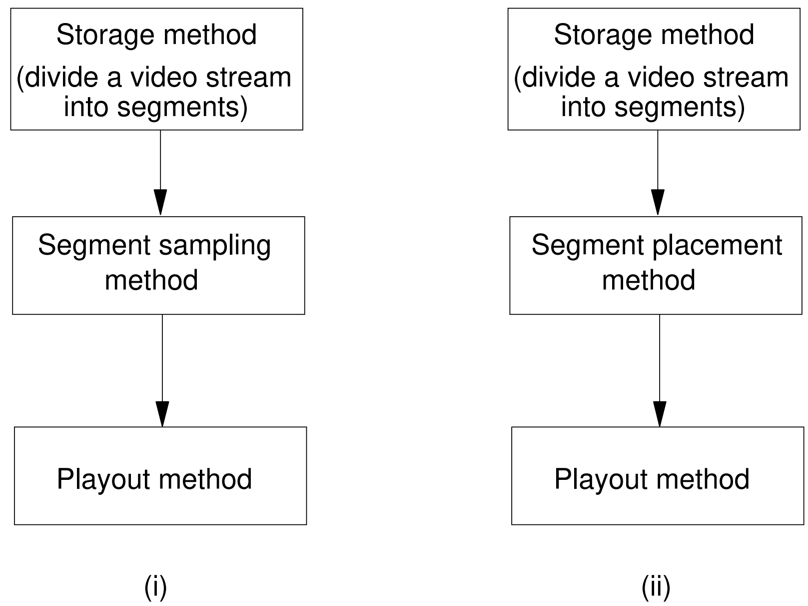

Figure 2: Two solution procedures for FF retrieval in a disk-array-based video server.

load across the disks. [5] mentions that it is possible to support FF with data striping for certain specific stride rates. Moreover, neither [2] nor [5] address the issue of inter-frame dependencies in the compressed video. The MPEG-2 draft standard [6] proposes the creation of special **D** frames that do not have any inter-frame dependency to support video browsing. However, the **D** frames contain only the DC coefficients of the transform blocks and consequently have very poor resolution. In other related work, issues such as admission control and selection of service size to support multimedia applications are addressed in [4, 7]. Disk scheduling schemes for optimizing the throughput of the DASD storage system are considered in [8, 9, 10].

This paper is organized as follows. In Section 2, the storage method and the playout method are presented. The segment sampling and segment placement schemes are devised in Section 3. Some experimental results are described in Section 4. This paper concludes with Section 5. Proofs for some lemmas and theorems are given in the appendix for better readability of this paper.

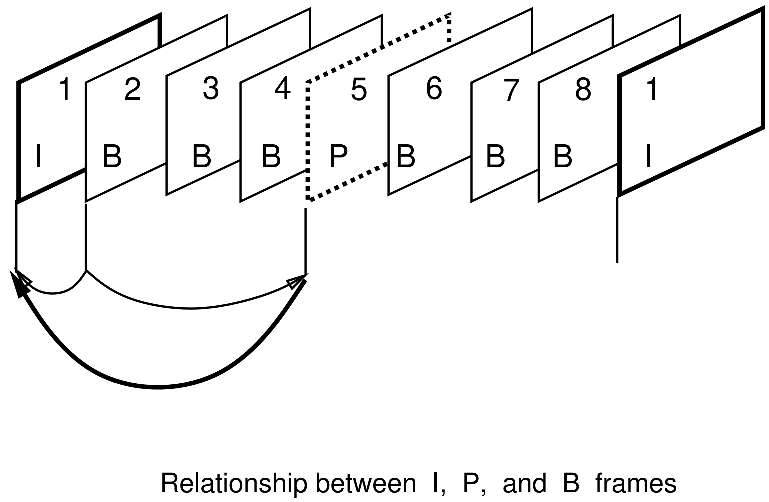

Figure 3: Sequence of MPEG frames.

2 Video Data Organization and Playout

The structure and definition of the compressed video stream imposes several constraints on the video data storage and playout. Consider an MPEG video stream which consists of intra frames (**I**), predictive frames (**P**), and interpolated frames (**B**). A representative sequence of frames is shown in Figure 3. In this stream **I** frames are coded such that they are independent of any other frames in the sequence. On the other hand, **P** frames are coded using motion estimation and have a dependency on the preceding **I** or **P** frame. Similarly **B** frames depend on two "anchor" frames: the preceding **I**/**P** frame and the following **I**/**P** frame. Since the **P** and **B** frames use inter-frame compression, they are substantially smaller than **I** frames. As a rule of thumb, an **I** frame is twice as large as a **P** frame which is in turn twice as large as a **B** frame.

In order to simplify buffering at the decoder, the MPEG standard requires that the decoder be presented with frames in an ordering that is appropriate for decoding. Specifically, a frame is presented to the decoder only after all frames on which it is dependent have been presented. It can be seen that this presentation order is different from the temporal order for a **B** frame since it has a dependency on the following anchor frame (**I** or **P**) (see Figure 4).

The inter-frame dependency implies that it is not possible to decode a **P** frame without the preceding **I** or **P** frame. Similarly, it is not possible to decode and play out **B** frames without the corresponding **I** and/or **P** frames. This in turn means that it is not possible to play out

```
Temporal Order:          I  B  B  P  B  B  P  B  B  I  B  B  P  B  B  P ...
Frame Number:            1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16

Presen./Storage Ord.:    I  P  B  B  P  B  B  I  B  B  P  B  B  P  B  B ...
Frame Number:            1  4  2  3  7  5  6  10 8  9  13 11 12 16 14 15
```

Figure 4: Temporal order (to the viewer) and presentation order (to the decoder) of MPEG frames.

every third frame of the MPEG stream shown in Figure 3 to achieve 3X playout since this subset would include **B** frames without the corresponding anchor **I** or **P** frames. It is possible, on the other hand, to generate a subset of the MPEG stream with fixed temporal separation between frames by selectively dropping **B** frames, for some specific fast forward rates. However, it is clear that such a subset has a substantially higher data rate than the original stream, thereby requiring additional resources for retrieval and transmission. A critical problem then arises in the receiving station since this higher data rate of the subset may cause a buffer overrun there. The receiving station in a VOD system is typically a price sensitive device which is designed for a particular playout rate and contains very limited buffer space. While this buffer space is sufficient to handle the full MPEG stream, it may be inadequate for such a higher data rate subset. The buffer overrun would result in a corrupted data stream and have serious consequences on the decoder.

Consequently, the challenge is to devise a scheme for fast forward operations which would satisfy the constraints of the decoder and require a minimum of additional system resources. The solution policies devised below for video data storage and playout meet these criteria.

2.1 Storage Policy for Video Segments

To comply with the dependency among different types of frames as described above, the multimedia stream is divided into *media segments* where each segment consists of consecutive frames beginning with an **I** frame and ending before another **I** frame. Each segment has a fixed size and forms the primary unit of storage and retrieval. Allocation and storage of the multimedia stream is in units of media segments. The disk blocks belonging to a media segment are co-located (preferably on the same cylinder) to minimize the seek overheads in retrieval and maximize the throughput. As will be shown in Section 3 later, consecutive segments will be placed in different disks so as to maximize the retrieval throughput in the disk array.

The storage order of frames within the segment is chosen to correspond to the order of presentation to the decoder. This choice minimizes the processing overheads during the normal playout. The relationship between the storage order and the temporal order is illustrated in Figure 4. It is noted that the **B** frames immediately preceding an **I** frame will be stored after the **I** frame. Thus, in the border condition, it can be seen that media segment i contains **B** frames that must be played out before the first frame of the segment, i.e., those **B** frames relating to segment $i-1$ in the temporal order. For the example frame sequence in Figure 4, the two **B** frames 8 and 9, immediately preceding **I** frame 10 in the temporal order, are stored after that **I** frame in the storage order. These two **B** frames will be played out before the first frame of the segment starting from **I** frame 10, since they relate to the previous segment (i.e., the one starting from **I** frame 1) in the temporal order.

Typically, the segment is large and comprises one or more disk tracks. It is advantageous to allocate entire disk tracks to segments since this results in a substantial increase in the access efficiency for storage and retrieval [3, 8]. Hence, the segment size is chosen such that it matches closely with a multiple of the disk track size. For example, given an MPEG stream of data rate 160 KBytes/sec (approx. 1.5 Mbps) and a track size of 41 Kbytes, we choose a segment size of 15 frames which corresponds to 0.5 seconds of playout and yields a storage unit of 2 disk tracks.

2.2 Playout Policy for Video Segments

Normal playout of the multimedia streams proceeds in "rounds". In each round, for every multimedia stream, media segments are read from each disk in the disk array. Consecutive media segments are stored on different disks in the array, and it is thus possible to retrieve n consecutive segments from n disks for each stream. The media segments are buffered temporarily in the server and transmitted at a fixed rate to the end station to be played out.

In the fast forward mode, the disk-array-based video server retrieves segments based on the segment selection methods to be described in Section 3 later. There are two different segment selection methods, the segment sampling method and the segment placement method.

In both methods, segments retrieved are properly skipped to achieve the effect of fast forward. Also, both methods ensure that for any given stream the segments to be retrieved all reside on different disks. Hence, the retrieval, buffering and transmission characteristics for the video server during the FF mode are similar to those during the normal playout.

The client station, on the other hand, has the responsibility of parsing the incoming stream and creating a valid input stream for the decoder. Note that these processing steps are straightforward as described below:

1. Discard intermediate frames that do not have the associated anchor frames. Note that although the media segment begins with an **I** frame, it contains **B** frames that have an anchor frame outside the media segment (such as **B** frames 8 and 9 in the segment starting from **I** frame 10 in the presentation/storage order in Figure 4). These **B** frames are located immediately following the **I** frame as shown in Figure 5, and should be ignored since they depend on the last **P** frame in the preceding media segment (which is not retrieved).

2. Adjust the presentation time-stamps embedded in the stream. The presentation time-stamp determines the time at which a video frame is to be displayed. It has to be adjusted to compensate for the skipped segments and also for the dropped **B** frames, so as to reflect the correct playout time.

These functions can also be performed at the video server, if necessary, to further implify the end station processing.

2.3 Remarks

Overall, as shown by our experiments conducted in Section 4, the video data storage and playout methods result in a piecewise continuous playout sequence. The viewers are allowed to use fast forward segment retrieval to examine scenes in order to quickly locate the scene of interest. The advantages of this methodology are manyfold.

- The server needs to retrieve only one media segment for the stream, as in the normal playout mode. Moreover, since the storage system stores media segments independently, the skipping of intermediate segments does not have any adverse impact on the retrieval process.

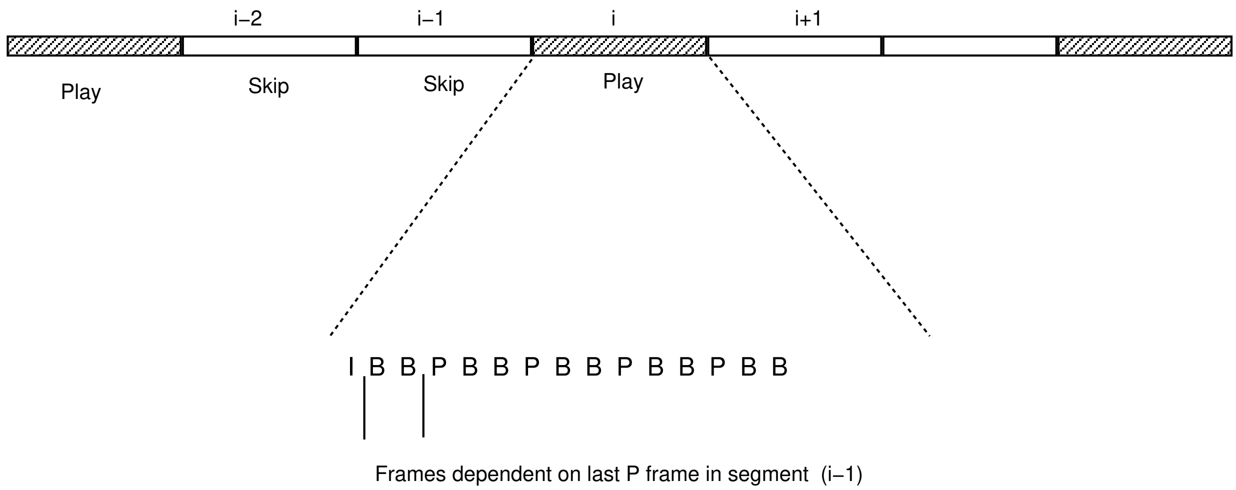

Figure 5: Fast forward sequence for MPEG frames.

- Since the segment size is fixed, no additional buffer or transmission bandwidth is required for the stream.
- The segment maintains the average data rate of the stream, and it is hence acceptable to the decoder in the client station.
- Although the receiving station has the additional responsibility of adjusting the time-stamps, the required function is not expensive. Moreover, such a function is required for any form of fast forward operation.

3 Segment Selection Schemes

In this section, we shall describe two alternative methods for video segment selection, namely the segment sampling method and the segment placement method. As shown in Figure 2, each of these two methods, in conjunction with the video segment storage and playout methods described in Section 2, provides a solution procedure for FF retrieval in a disk-array-based video server.

3.1 Segment Sampling Method

The segment sampling method assumes that media segments belonging to any multimedia stream are stored in a robin-robin manner on the disk array. It determines the retrieval order of segments for FF operations for any desired FF rate. The segments selected by this method are distributed uniformly across the disk array and have a minimal variation on the number of segments skipped between every two consecutive retrieved segments.

Consider a disk-array-based video server which contains n disks in each string, with media segments distributed in a round-robin manner across the disks. Formally, segment g is stored in disk $k = f_1(g, n)$ in a disk array of n disks, where

$$f_1(g, n) = [g]_n.$$

Here, $[g]_n$ denotes g mod n for notational simplicity. An example of the round-robin segment placement for $n = 10$ is given in Figure 6.

As described in Section 2.2, the retrieval process proceeds in rounds and in each round n segments are retrieved for each active stream. To achieve the FF feature, one has to appropriately select (sample) the n segments to be retrieved in each round. Clearly, the sampling process depends upon the FF speed desired. For example, if the FF speed is m times the normal speed, we would on the average sample one segment out of m segments. To provide the best output during the FF mode, we want the segments sampled to be distributed as uniformly as possible. For example, to double the speed, one can simply select even segments while skipping the odd ones. However, it can be verified that this naive method, while uniformly sampling the segments, will not achieve the maximal throughput when there are even number of disks in the array (such as the situation in Figure 6 where $n = 10$). In that case, only half of the disks (i.e., disks with even numbers) will participate in the segment retrieval process whereas the other half (i.e., disks with odd numbers) remain idle. Hence, to develop an FF retrieval scheme in a disk array, we not only want to sample the segments as uniformly as possible but also need to ensure that the maximal throughput is attained. To achieve these objectives, we devise the segment sampling procedure, algorithm S, as described below.

For illustrative purposes, Table 1 shows the segments retrieved in the first two rounds of retrieval by algorithm S for the case where the number of disks in the disk array $n = 9$ and the FF speedup $m = 3$. Let lcm(n, m) denote the least common multiple of n and m; for the example of Table 1, we have lcm(3,9)=9. The basic concept of algorithm S is to shift the position of the segment to be retrieved by one for every lcm(n, m) segments involved. It can be seen that the segments retrieved in Table 1 (i.e., those marked with *'s) form a zig-zag curve. Note that a complete zig-zag curve involves two rounds of disk retrieval and there are

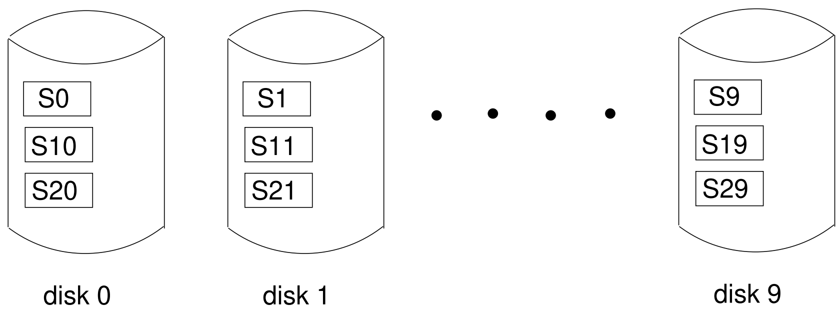

Figure 6: A round-robin segment placement in a disk array of 10 disks.

$n * m$ segments involved in each round of retrieval. Hence, we get the length of a zig-zag curve to be equal to $\frac{2*n*m}{\text{lcm}(n,m)}$. For example, the length of a zig-zag curve in Table 1 is 6. Using these concepts, it can be proved that algorithm S described below will lead to a zig-zag type of segment retrieval.

Algorithm S: /* Suppose m is the desired fast retrieval speedup and n is the number of disks in the disk array. */

In the r-th round of retrieval, the k-th disk, $0 \leq k \leq n-1$, retrieves the segment with segment number $h_1(k,r)$, where, with $z = \frac{n*m}{\text{lcm}(n,m)}$,

$$h_1(k,r) = \begin{cases} (rz - z + [k]_z) * \text{lcm}(n,m) + k + n * [\lfloor \frac{m-[n]_m}{z} \rfloor \lfloor \frac{k}{z} \rfloor]_{m/z} & \text{if } r \text{ is an odd number,} \\ (rz - 1 - [k]_z) * \text{lcm}(n,m) + k + n * [\lfloor \frac{m-[n]_m}{z} \rfloor \lfloor \frac{k}{z} \rfloor]_{m/z} & \text{otherwise.} \end{cases}$$

Note that the case that r is an odd number corresponds to the first half of a zig-zag (such as the first round of retrieval in Table 1), and the case that r is an even number corresponds to the second half of a zig-zag (such as the second round of retrieval in Table 1). For example, consider the case that $n = 9$ and $m = 3$ as in Table 1. We have $z = \frac{9*3}{\text{lcm}(9,3)} = 3$. For the example of $r = 1$ and $k = 4$, we get $h_1(4{,}1) = (3 - 3 + [4]_3) * \text{lcm}(9,3)$ $+4+$ $9 * [\lfloor \frac{3-[9]_3}{3} \frac{4}{3} \rfloor]_1 =$

Disk No.	0	1	2	3	4	5	6	7	8
1st round	0*	1	2	3*	4	5	6*	7	8
	9	10*	11	12	13*	14	15	16*	17
	18	19	20*	21	22	23*	24	25	26*
2nd round	27	28	29*	30	31	32*	33	34	35*
	36	37*	38	39	40*	41	42	43*	44
	45*	46	47	48*	49	50	51*	52	53
1st rd segments	0	10	20	3	13	23	6	16	26
2nd rd segments	45	37	29	48	40	32	51	43	35

Table 1: A fast retrieval for n=9 and m=3, where the length of zig-zag curve is 6.

9+4+0=13. For another example of $r = 2$ and $k = 6$, we get $h_1(6,2) = (2*3-1+[6]_3)*\mathrm{lcm}(9,3)$ $+6+\ 9*[\lfloor \frac{3-[9]_3}{3}\frac{6}{3}\rfloor]_1 =\ 5*9$+6+0=51, as indicated in Table 1.

It can be seen from Table 1 that instead of retrieving segments whose numbers are multiples of three (which would make disks 1, 2, 4, 5, 7 and 8 idle), the proposed scheme properly shifts the data segments retrieved in some retrievals. For example, segment 10 is fetched instead of segment 9, and segment 20 is fetched instead of segment 19. Such shifting, though making the segments retrieved not perfectly uniformly distributed, ensures the maximal throughput is achieved. The group of segments retrieved in the first round of retrieval in Table 1 consists of segments 0, 10, 20, 3, 13, 23, 6, 16 and 26, which are then displayed in the order of 0, 3, 6, 10‡, 13, 16, 20‡, 23, 26, where ‡ indicates a shift for retrieved segment numbers. Similarly, the segments retrieved in the second round of retrieval in Table 1 are segments 45, 37, 29, 48, 40, 32, 51, 43, and 35, which are then displayed in the order of 29, 32, 35, 37‡, 40, 43, 45‡, 48, 51.

Table 2 illustrates an example of FF retrieval where n=10 and m=6. It can be shown that there are 60 segments involved in one round of disk retrieval, and the length of zig-zag is 4. It follows that there are two shifts in a zig-zag curve (specifically, one from 24 to 31 and the other from 85 to 90). Except for these two shifts, all other data segment numbers retrieved are uniformly distributed (i.e., exactly separated by 6, the speedup factor). Note that unlike the situation in Table 1, $\mathrm{lcm}(n, m)$ is not equal to n in this case. The shape of zig-zag in Table 2 can be visualized from retrieved segments, 0, 31, 61, and 90, showing a zig-zag of length four. As an illustrative example, we have $z = \frac{10*6}{\mathrm{lcm}(10,6)} = 2$. For $r = 1$ and $k = 5$, we get $h_1(5,1) =$ $(2-2+[5]_2)*\mathrm{lcm}(10,6)$ $+5+\ 10*[\lfloor \frac{6-[10]_6}{2}\frac{5}{2}\rfloor]_3 =$ 30+5+20=55, as indicated in Table 2.

Disk No.	0	1	2	3	4	5	6	7	8	9
1st round	0*	1	2	3	4	5	6*	7	8	9
	10	11	12*	13	14	15	16	17	18*	19
	20	21	22	23	24*	25	26	27	28	29
	30	31*	32	33	34	35	36	37*	38	39
	40	41	42	43*	44	45	46	47	48	49*
	50	51	52	53	54	55*	56	57	58	59
2nd round	60	61*	62	63	64	65	66	67*	68	69
	70	71	72	73*	74	75	76	77	78	79*
	80	81	82	83	84	85*	86	87	88	89
	90*	91	92	93	94	95	96*	97	98	99
	100	101	102*	103	104	105	106	107	108*	109
	110	111	112	113	114*	115	116	117	118	119
1st rd segments	0	31	12	43	24	55	6	37	18	49
2nd rd segments	90	61	102	73	114	85	96	67	108	79

Table 2: A fast retrieval for n=10 and m=6, where the length of zig-zag curve is 4.

Table 3 illustrates an example of fast retrieval where $n = 7$ and $m = 3$. It can be seen that lcm(n, m)=$n * m$ in this case, and there are 21 segments involved in one round of disk retrieval. The length of a zig-zag curve is equal to two. As a result of lcm(n, m)=$n * m$, it can be verified that the number $z = \frac{n*m}{\text{lcm}(n,m)} = 1$ and the two formulas of $h_1(k, r)$ in algorithm S become the same. Hence, there is no shift required in this case. As an illustrative example, we have $z = \frac{7*3}{\text{lcm}(7,3)} = 1$. For $r = 1$ and $k = 4$, we get $h_1(4{,}1) = (1 - 1 + [4]_1) * \text{lcm}(7, 3) + 4 + 7 * [\lfloor \frac{3-[7]_3}{1} \frac{4}{1} \rfloor]_3$= 4+7*2=18, as indicated in Table 3.

Clearly, the fewer shifts, the better sampling an FF retrieval achieves. It can be proved that the number of shifts incurred by algorithm S is the minimum among all fast retrieval schemes that could achieve the maximal throughput. We have the following lemma and theorem for the optimality of algorithm S, whose proofs can be found in the Appendix.

Lemma 1: The number of shifts in segment numbers in each round of retrieving n segments by algorithm S is $\frac{n*m}{\text{lcm}(n,m)} - 1$.

Theorem 1: The number of shifts incurred by algorithm S is the minimum among all FF retrieval schemes that could achieve the maximal throughput.

Disk No.	0	1	2	3	4	5	6
1st round	0*	1	2	3*	4	5	6*
	7	8	9*	10	11	12*	13
	14	15*	16	17	18*	19	20
1st rd segments	0	15	9	3	18	12	6

Table 3: A fast retrieval for n=7 and m=3, where the length of zig-zag curve is 2.

3.2 Segment Placement Method

In contrast to the segment sampling method that selectively retrieves segments from a disk array in which segments are stored in a round-robin manner, the segment placement method allocates segments to disks judiciously such that no special provision is needed for sampling and the segment can be completely uniformly sampled in an FF mode for some pre-determined FF speeds.

To derive the segment placement function, consider an FF operation in which the playout rate is m times the normal playout rate. In this FF mode, the sequence of retrieved segments from a given starting segment i is $\{i, i+m, i+2m, i+3m, ...\}$. Since n media segments are retrieved in each round, there are $n*m$ segments involved in each round of disk retrieval. Thus, in the r-th round of retrieval, the segments to be retrieved are $\{(r-1)nm, (r-1)nm+m, (r-1)nm+2m, ..., (r-1)nm+(n-1)m\}$. It is necessary to ensure that these segments be mapped to different disks so as to obtain the maximal throughput.

The segment placement function, f_2(g,n), defines a mapping from media segment g to a disk, $k \in [0, n)$, in the disk array with n disks. Assuming that m is a sub-multiple of the number of disks n, i.e., $[n]_m$=0, the segment placement function is derived as follows:

$$f_2(g,n) = [g + \lfloor g/n \rfloor]_n.$$

The first term, i.e., g within $[.]_n$, represents a regular scattering of the segments on the n disks and the second term, i.e., $\lfloor g/n \rfloor$ within $[.]_n$, represents a skew factor. It can be shown that $f_2(g,n)$ maps the segments $\{(r-1)nm, (r-1)nm+m, (r-1)nm+2m, ..., (r-1)n*m+(n-1)m\}$ to different disks for any r, as stated by Lemma 2 and Theorem 2 below where $f_2(g,n)$ is denoted by $x_n(g)$ for ease of presentation.

Lemma 2: The function x_n provides a bijective mapping of the set $\mathcal{T} = \{i, i+n, i+2n, ..., i+$

Disk No.	0	1	2	3	4	5
1st round	0*	1	2*	3	4*	5
	11	6*	7	8*	9	10*
2nd round	16*	17	12*	13	14*	15
	21	22*	23	18*	19	20*

Table 4: Segment layout on a disk array of 6 disks.

$(n-1)n\}$ onto the set $\{0, 1, .., n-1\}$ for any integer i.

Theorem 2: The function x_n provides a bijective mapping of the set $\mathcal{S} = \{i, i+m, i+2m, ..., i+(n-1)m\}$ onto the set $\{0, 1, .., n-1\}$ when i is of the form $i = (pn + a); a < m$ and m divides n.

Note that the segment placement function, $f_2(g, n)$, is not dependent on m, and can in fact be applied to any playout rate m provided that m is a sub-multiple of n. For example, the segment placement for $n = 6$ is shown in Figure 7. Based on the segment placement determined by $f_2(g, n)$, it can be proved that algorithm P below will provide completely uniform segment sampling for FF operations.

Algorithm P: /* Suppose m is the desired FF speedup, n is the number of disks in the disk array, and $[n]_m = 0$. */
In the r-th round of retrieval, the k-th disk, $0 \leq k \leq n-1$, retrieves the segment with segment number $h_2(k, r)$, where

$$h_2(k, r) = ((r-1)m + [k]_m)n + [k - [(r-1)m + [k]_m]_n]_n.$$

For example, when $r = 1$, $m = 3$ and $k = 4$ in Figure 7, we get $h_2(4, 1)$= $([4]_3)6$+ $[4 - [[4]_3]_6]_6$= 6+3=9. As mentioned above, the segment placement is optimized for an FF rate m as long as m is a sub-multiple of the number of disks n. Thus, the segment layout shown in Figure 7 can support 2X, 3X and 6X FF speedups. For illustrative purposes, the case that the FF speedup $m = 2$ for the segment placement in Figure 7 is shown in Table 4, where the segments retrieved are marked with *'s. For example, when $r = 2$, $m = 2$ and $k = 1$, we get $h_2(1, 2)$= $(2 + [1]_2)6$+ $[1 - [2 + [1]_2]_6]_6$= 18+4=22, as indicated in Table 4.

This layout ensures that the media segments to be retrieved in a round all reside in different

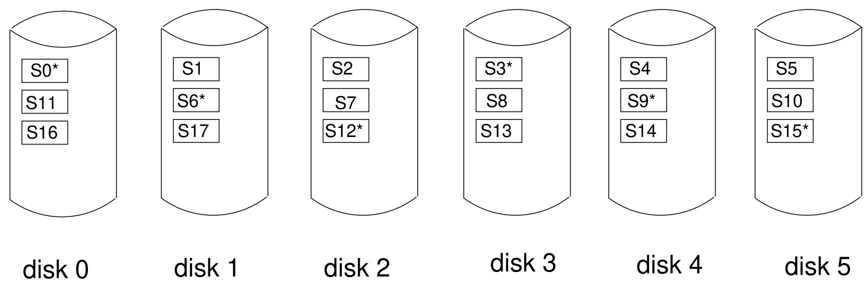

Figure 7: The segment placement in a disk array of 6 disks.

disks, so that the load imposed on the storage system by the retrieval process in FF mode is identical to the load under the normal operation.

3.3 Remarks

The segment sampling method uses a simple round-robin placement policy and a sophisticated segment sampling strategy for retrieval. It can be used to provide FF retrieval at any desired rate with only a minimal deviation from the uniform sampling sequence. Moreover, for certain FF rates m and the disk array size n where m and n are relatively prime, the sampling method produces a uniform sampling sequence. When the desired FF rates are known *a priori* it is possible to simplify the segment retrieval process by using a more sophisticated segment placement method. The placement method guarantees a uniform retrieval sequence that presents an evenly distributed load on the disk array. It is noted that although these methods have been described in the context of FF retrieval they can also be applied to fast-backward retrieval with some changes in the end station for playout. For example, in the first round of Table 1 the display order for fast backward will be 26, 23, 20, 16, 13, 10, 6, 3, and 0.

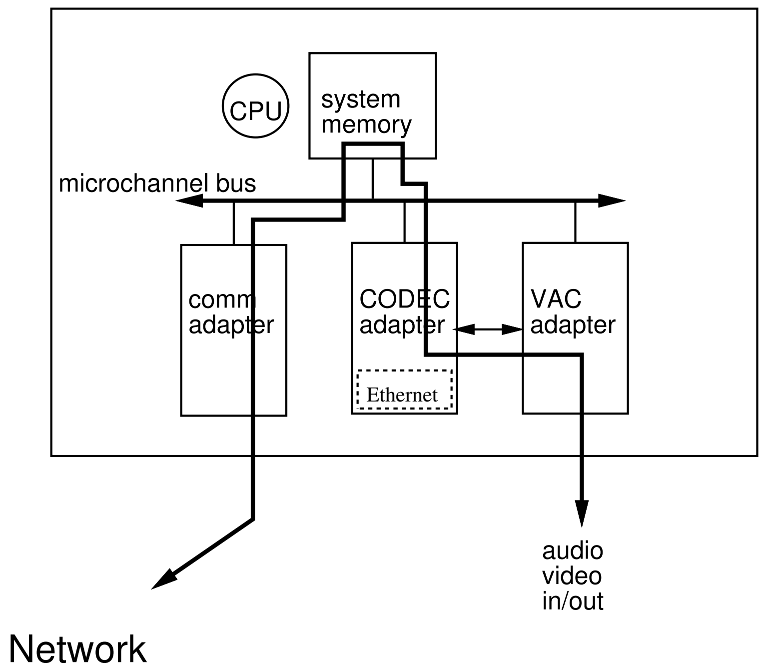

Figure 8: The MMT System.

4 Visual Experiments

As explained in Section 2, there are several compelling reasons for using the segment skipping method to implement fast-forward operations. However, it is necessary to study the visual impact of this method to determine whether it is acceptable to the end-users of the VOD system, namely the viewers.

In order to perform this visual experiment, we have used the MMT (Multimedia Multiparty Teleconferencing) system, a prototype desktop collaboration system developed at the IBM Thomas J. Watson Research Center [11]. The current MMT hardware consists of an IBM PS/2 computer equipped with two custom-built adaptors. One adaptor (VAC) interfaces with analog video and audio components and performs capture (digitization) and playback functions. The other adaptor (MMT) is for compression, networking, processing, and decompression of video and audio. It uses the JPEG compression algorithm [12] for compressing individual frames of the video input. The system runs AIX, a variant of the popular UNIX operating system.

Figure 8 shows one of the possible data paths through the MMT system. In this scenario, the video application extracts video and audio data from the MMT adaptor and transports it across the network. This video application has been modified to (a) store the video data to a file, and (b) playout video data from a file. In the first instance, the timing/pacing for the recording process is provided naturally by the video source and data is generated whenever a new video frame is captured and compressed. However, in the case of stored video playout no suitable timing signal is available from the operating system to pace the playout process. We therefore chose to exploit the video source to provide the pacing for the playout. The playout application configures the MMT adaptor for capture and playout of video. It then interprets the incoming video data to demarcate frames and paces the playout by ensuring that the number of video frames played out to the MMT adaptor closely corresponds to the number of video frames captured.

We have also developed a filter program that processes the stored video to produce an output file with the appropriate segment size and skip size. This filter represents an off-line implementation of the fast-forward mechanism. Using these tools we have experimented with the fast-forward playout of video using different segment sizes and skip factors (i.e., the FF speedup rate m). Since the objective of our experiments is to determine the visual impact of the segment skipping, and not the efficiency of the compression, our use of JPEG as the compression format does not alter the results. Moreover, since each JPEG frame is independent, it also permits us to play every mth frame to achieve VCR-like fast-forward.

Our initial assessment has been that the segment skipping method of fast-forward is visually acceptable. The viewer gets the impression of watching each scene (segment) at regular speed with jumps between scenes, similar to watching a slide projector operating at a high speed. Viewers can see the details in each scene so that they are able to locate the position of interest. A 3 minute video tape was made to show the effect of segment skipping. By varying the segment size and the skip interval, we have found that it is important to keep the size of the segment long – of the order of 1 second or more – to ensure that the discontinuities are minimized. This fits in well with the structure of MPEG streams since the interval time between **I** frames is typically one-half second or more.

5 Conclusions

In this paper, we presented frame skipping schemes to support variable rate FF and FB operations for MPEG video streams. We specifically considered retrieval for a disk-array-based video server. We considered the cases to support (a) disk arrays of any size for any desired FF/FB speedup and (b) disk arrays of a given size with some specific speedup rates. Our overall approach for interactive browsing comprises (1) a storage method, (2a) a segment sampling scheme, (2b) a segment placement scheme, and (3) a playout method, where (2a) and (2b) are two alternatives for segment selection. Experiments on implementing the frame skipping approach on our prototype were also described.

Acknowledgement

The authors would like to gratefully acknowledge Tsipora Barzilai for her assistance in the development of the recording and playout programs.

References

[1] "Coding of moving pictures and associated audio – for digital storage media at up to about 1.5Mbit/s." ISO Standard IS 11172, Nov. 1992.

[2] T. Mori, K. Nishimura, H. Nakano, and Y. Ishibashi, "Video-on-demand system using optical mass storage system," *Japanese Journal of Applied Physics*, vol. 1, pp. 5433–5438, Nov. 1993.

[3] D. D. Kandlur, M.-S. Chen, and Z.-Y. Shae, "Design of a multimedia storage server," in *Proc. IS&T/SPIE Symposium on Electronic Imaging – Conference on High speed networking and Multimedia Applications*, SPIE, Feb. 1994. Also available as IBM Reseach Report **RC 19158**, Sept. 1993.

[4] H. M. Vin and P. V. Rangan, "Designing a multi-user HDTV storage server," *IEEE Journal on Selected Areas in Communication*, Jan. 1993.

[5] E. Chang and A. Zakhor, "Scalable video data placement on parallel disk arrays," in *Proc. IS&T/SPIE Symposium on Electronic Imaging – Conference on Image and Video Databases II*, SPIE, Feb. 1994.

[6] "Generic coding of moving pictures and associated audio." ISO/IEC Recommendation H.26x, working draft, Sept. 1993.

[7] P. V. Rangan and H. M. Vin, "Designing file systems for digital video and audio," in *Proceedings ACM Symposium on Operating Systems Principles*, pp. 81–94, Oct. 1991.

[8] P. S. Yu, M.-S. Chen, and D. D. Kandlur, "Grouped sweeping scheduling for DASD-based multimedia storage management," *Multimedia Systems*, vol. 1, pp. 99–109, 1993. Springer-Verlag.

[9] D. J. Gemmell, "Multimedia network file servers: Multi-channel delay sensitive data retrieval," in *Proc. ACM Multimedia '93*, pp. 243–250, ACM, Aug. 1993.

[10] A. L. N. Reddy and J. Wyllie, "Disk scheduling in a multimedia I/O system," in *Proc. ACM Multimedia '93*, pp. 225–234, ACM, Aug. 1993.

[11] M.-S. Chen, Z.-Y. Shae, D. D. Kandlur, T. P. Barzilai, and H. M. Vin, "A multimedia desktop collaboration system," in *Proceedings GLOBECOM 92*, IEEE, Dec. 1992.

[12] G. K. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, pp. 30–44, Apr. 1991.

Appendix: Proofs of theorems and lemmas

Proof of Lemma 1: In order to retrieve segments from different disks, algorithm Z shifts the retrieved segment number for every $\frac{\mathrm{lcm}(n,m)}{m}$ segment retrievals. This lemma then follows from the fact that n segments are retrieved in each round and one shift is saved for every round. ∎

Proof of Theorem 1: Note that for any $\frac{2*\mathrm{lcm}(n,m)}{m}$ consecutive segment retrievals to fall into different disks, at least one shift on the retrieved segment numbers is required. It then follows that for any $\frac{(i+1)\mathrm{lcm}(n,m)}{m}$ consecutive segment retrievals to fall into different disks, at least i shifts on the retrieved segement numbers are required. By letting $n = \frac{(i+1)\mathrm{lcm}(n,m)}{m}$, we get $i = \frac{n*m}{\mathrm{lcm}(n,m)} - 1$, proving this theorem. ∎

Proof of Lemma 2: Consider any two integers $a, b \in \mathcal{T}$. Without loss of generality b can be expressed as $a + k'n$ for some integer k'.

$$\begin{aligned}
x_n(a) &= [a + \lfloor a/n \rfloor]_n \\
x_n(b) &= x_n(a + k'n) = [a + k'n + \lfloor (a + k'n)/n \rfloor]_n \\
&= [a + \lfloor a/n \rfloor + k']_n
\end{aligned}$$

where $[\ell]_n$ represents the ℓth equivalence class modulo n. It is clear that $x_n(b) \neq x_n(a)$ when $k' < n$. ∎

Proof of Theorem 2: The proof for Theorem 2 is based on showing a bijective mapping g between elements of $\mathcal{S}$ and $\mathcal{T}$ with the property $x_n(c) = x_n(y(c))$ for any element $c \in S$. Once this mapping is established, Lemma 2 above can be used to prove that the elements are distinct.

Consider the element $(i + jm) \in \mathcal{S}$ where $j \in [0, n)$. Since $n = l * m$ is a multiple of m, $j * m$ can be expressed as

$$\begin{aligned}
jm &= (j_1 + j_2 l)m = j_1 m + j_2 n \text{ for some integers } j_1, j_2, \text{ where} \\
j_1 &= j \bmod l, \text{ and} \\
j_2 &= \lfloor j/l \rfloor.
\end{aligned}$$

Since $j \in [0, n)$ and $n = ml$, $0 \leq j_1 < l$ and $0 \leq j_2 < m$.

Define y as

$$y(i + j_1 m + j_2 n) = (i + (j_1 m + j_2)n).$$

Given that $j_1 \in [0, l)$ and $j_2 \in [0, m)$, $(j_1 m + j_2)$ is an integer in the range $[0, (ml = n))$, so $(i + (j_1 m + j_2)n) \in \mathcal{T}$. It can be seen that this mapping from $\mathcal{S}$ to $\mathcal{T}$ is one-one.

$$\begin{aligned} x_n(i + j_1 m + j_2 n) &= x_n((a + pn) + j_1 m + j_2 n) \\ &= [a + pn + j_1 m + j_2 n + \lfloor (a + pn + j_1 m + j_2 n)/n \rfloor]_n \\ &= [a + j_1 m + \lfloor (a + j_1 m)/n \rfloor + p + j_2]_n \end{aligned}$$

Since $a < m$ and $j_1 < l$, $a + j_1 m < n$ and so the expression reduces to $[a + p + j_1 m + j_2]_n$.

$$\begin{aligned} x_n(i + j_1 mn + j_2 n) &= x_n((a + pn) + j_1 mn + j_2 n) \\ &= [a + pn + j_1 mn + j_2 n + \lfloor (a + pn + j_1 mn + j_2 n)/n \rfloor]_n \\ &= [a + \lfloor a/n \rfloor + p + j_1 m + j_2]_n \\ &= [a + p + j_1 m + j_2]_n \text{ since } a < n. \end{aligned}$$

From these equations it is seen that $x_n(\mathcal{S})$ is identical to $x_n(y(\mathcal{S})) = x_n(\mathcal{T})$, which by the result of Lemma 2 is the set [0,n). ∎