Efficient Support for Interactive Operations in Multi-resolution Video Servers

Prashant J. Shenoy and **Harrick M. Vin***

Distributed Multimedia Computing Laboratory
Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188

E-mail: {shenoy,vin}@cs.utexas.edu, Phone: (512) 471-9732, Fax: (512) 471-8885
URL: http://www.cs.utexas.edu/users/dmcl

Abstract

In this paper, we present a placement algorithm that interleaves multi-resolution video streams on a disk array and enables a video server to efficiently support playback of these streams at different resolution levels. We then combine this placement algorithm with a scalable compression technique to efficiently support interactive scan operations (i.e., fast-forward and rewind). We present an analytical model for evaluating the impact of the scan operations on the performance of disk-array-based servers. Our experiments demonstrate that: (1) employing our placement algorithm substantially reduces seek and rotational latency overhead during playback, and (2) exploiting the characteristics of video streams and human perceptual tolerances enables a server to support interactive scan operations without any additional overhead.

Keywords

Multi-resolution video servers, multi-resolution playback, scan operations, fast-forward, rewind, disk arrays, scalable compression

1 Introduction

Recent advances in computing and communication technologies promise to create an infrastructure in which computer systems will support a wide range of interactive multimedia services in a variety of commercial and entertainment domains (e.g., advertising, online news, customer support, video on-demand, etc.). In its simplest configuration, such services will be provided by multimedia servers connected to client sites via high-speed networks. Clients will dial-up the service and request the retrieval of information objects (consisting of audio, video, text, imagery, etc.) stored at the server. The resolution at which an object is requested depends on the capabilities of the client and the speed of the network connection. The retrieval can be interactive, in the sense that clients can stop, pause, resume, fast-forward, or rewind the presentation.

Amongst all these data types, since video is the most demanding (with respect to its data rate and real-time performance requirements), several techniques for designing video servers that can meet the real-time playback requirement of single resolution video streams have been developed [1, 10, 16, 18, 20]. However, methods for efficiently supporting interactive operations (playback, fast-forward and rewind) for multi-resolution video streams have not been adequately investigated.

*This research was supported in part by an IBM Faculty Development Award, Intel, the National Science Foundation (Research Initiation Award CCR-9409666 and CAREER award CCR-9624757), NASA, Mitsubishi Electric Research Laboratories (MERL), and Sun Microsystems Inc.

1.1 Relation to Previous Work

In general, a multi-resolution video stream consists of multiple sub-streams [6]. Whereas all sub-streams must be retrieved to display the video stream at the highest resolution, only a subset of the sub-streams need to be retrieved for playback at a lower resolution. Due to the large storage space requirement of multi-resolution video, most video servers employ disk arrays as their underlying storage medium. To effectively utilize the array bandwidth, the server stripes or interleaves each video stream among disks in the array [16]. In the simplest case, the server can stripe each sub-stream independently across the array. However, such a scheme can impose significant disk seek and rotational latency overheads while retrieving these sub-streams during playback. In contrast, multiplexing all component sub-streams into a single stream and striping this mutiplexed stream eliminates these latency overheads. Since the server accesses data in terms of disk blocks (rather than frames), a limitation of this approach is that the server may have to access more information than necessary while accessing a particular sub-stream. Hence, a key challenge is to design a placement scheme that allows each sub-stream to be independently accessed, without imposing significant seek and rotational latency overheads. Most existing placement schemes have been developed for single resolution video streams [2, 16]; placement schemes that support efficient playback of multi-resolution video streams have not been adequately investigated.

Several techniques for supporting interactive operations such as fast-forward and rewind (collectively referred to as scan) have been proposed in the literature. For instance, a video server can support fast-forward by: (1) displaying frames at a rate higher than normal playback [7, 8], or (2) skipping frames [4, 13], or (3) using an independently encoded fast-forward stream. In the first scheme, to support fast-forward at n-times the normal playback rate, the server is required to retrieve n-times as many frames (as compared to the normal playback), yielding an n-fold increase in the load. If additional resources (e.g., disk and network bandwidth) are not available to meet the increased requirements, the client request must be delayed until the necessary resources become available. To minimize the waiting time, the server can set aside some resources to accommodate such dynamic transitions from playback to fast-forward. The additional resources that must be set aside is dependent on the probability of clients requesting a transition from playback to fast-forward, as well as the duration for which a client remains in the fast-forward mode [7].

In schemes that skip frames, on the other hand, fast-forward at n-times the playback rate is achieved by displaying every n^{th} frame at the normal playback rate. However, such frame skipping schemes may not be directly applicable for video streams encoded using compression algorithms that exploit temporal redundancy between successive frames (e.g., the MPEG compression standard [9]). This is because, such compression techniques create inter-frame dependencies which may prevent every n^{th} frame to be independently decoded. To avoid this problem, a fast-forward scheme in which: (1) video streams are stored on disks in terms of *segments* that consist of a group of independently decodable frames; and (2) fast-forward at n-times the normal playback is achieved by accessing and displaying every n-th segment has been proposed [4]. Such an approach eliminates the problem introduced by inter-frame dependencies. However, since each segment may contain a large number of successive frames (10-15 frames in MPEG), skipping entire segments may result in noticeable discontinuities during fast-forward, and hence, may be unacceptable.

To emulate the VCR fast-forward operation, a server can encode a fast-forward stream that is independent of the parent video stream and utilize it only during fast-forward. By properly selecting the encoding procedure, the server can ensure that accessing such a specialized stream does not require any additional disk or network bandwidth. However, maintaining such a fast-forward stream may incur a substantial storage space overhead. To minimize storage space overhead, the MPEG standard has proposed the creation of a video stream containing D frames, which contain only the DC coefficients of the transform blocks [9]. However, this yields a video stream with very poor quality, which is not acceptable for most applications.

Since rewind is similar to fast-forward, all of the above approaches for fast-forward can be easily extended to support rewind. In addition to these server-based approaches, several client-based approaches that do not require any server intervention during rewind have also been proposed [3, 8]. These techniques require a client to cache previously displayed frames for rewind. For instance, a scheme in which the client caches all previously displayed frames on a local disk has been recently proposed [3]. In this scheme, frames decoded during normal playback are re-encoded such the stream does not contain any inter-frame dependencies, thereby enabling a client to independently decode every n-th frame during rewind. A limitation of this approach, however, is the additional storage space required at the client to support rewind. If the storage space available at the client is limited, then the client can cache only a small number of previously displayed frames [8]. In such a scenario, the duration of the rewind operation that can be supported without imposing any load on the server is limited by the size of the cache (referred to as a VCR-window). If a rewind operation exceeds this duration, then server intervention is required to retrieve and transmit additional frames.

In summary, most conventional schemes require additional resources either at the server or at the client to support scan operations. To be practical, a scheme that supports scan operations must : (1) minimize the storage space overhead of maintaining information pertinent to scan operations, (2) minimize the increase in the bit rate during scan (and hence, minimize the overhead on the disk and network subsystems), and (3) provide acceptable video quality.

Techniques for efficiently supporting playback, fast-forward, and rewind in multi-resolution video servers constitutes the subject matter of this paper.

1.2 Research Contributions of This Paper

In this paper, we make three contributions. First, we present a placement algorithm that interleaves the storage of multi-resolution video streams on a disk array and ensures that: (1) each sub-stream within a stream is independently accessible, and (2) the seek time and rotational latency overheads incurred while accessing these sub-streams during playback is minimized. Our placement algorithm enables the server to efficient support playback of video streams at different resolution levels.

Second, we present an encoding technique that when combined with our placement algorithm enables a server to efficiently support interactive scan operations (fast-forward and rewind). Our encoding technique uses scalable compression techniques in the temporal and chroma dimensions to derive a video stream that can efficiently support scan operations. Specifically, it first partitions the parent video stream in the temporal dimension to create a *base* and an *enhancement* sub-stream. Each frame in the base sub-stream is then partitioned in the chroma dimension yielding a *low-resolution* and a *residual* component. Whereas frames contained in all three components are merged during normal playback, scan operations are supported by using only the low-resolution component of the base sub-stream. By appropriately sub-sampling in the temporal dimension, the encoder can support multiple fast-forward/rewind rates. Furthermore, by appropriately controlling the chroma dimension partitioning of base sub-stream, the encoder can ensure that: (1) the bit rate of the low-resolution base sub-stream during scan is not significantly higher than that of the parent video stream during normal playback, and (2) the resultant low-resolution base sub-stream provides acceptable video quality for scan operations. By employing our placement algorithm to interleave these sub-streams, the server can efficiently retrieve a video stream during playback or scan without imposing any additional disk latency overheads. We demonstrate the efficacy of our methodology for the MPEG-2 compression algorithm.

Finally, to evaluate the efficacy of our encoding technique and placement algorithm, we present an analytical model that predicts the effects of making a transition from playback to scan on the array load. We use the analytical model to compute the contingency bandwidth that must be reserved at the server to accommodate any increase in the load caused by playback to scan transitions.

We have implemented a prototype encoder and an event-driven disk array simulator to evaluate our schemes. We generated several traces using our encoder and used them for extensive trace-driven simulations. Our results demonstrate that: (1) the placement algorithm substantially reduces disk latency overheads during playback, and (2) exploiting characteristics of video streams and human perceptual tolerances enables a server to support scan operations without any additional overhead.

The rest of the paper is organized as follows. We present our placement algorithm in Section 2. In Section 3, we present techniques for supporting fast-forward and rewind in a video server, and then derive an analytical model for evaluating the effects of these operations on the array load. We evaluate our scheme through extensive simulations in Section 5, and finally, Section 6 summarize our results.

2 Efficient Placement Of Multi-resolution Video On Disk Arrays

In general, a multi-resolution video stream consists of multiple sub-streams. Whereas all sub-streams must be retrieved to display the video stream at the highest resolution, only a subset of the sub-streams need to be retrieved for playback at a lower resolution. To efficiently support the retrieval of such streams at different resolutions, the placement algorithm must ensure that the server can *access only as much data as needed and no more*. To ensure this property, the placement algorithm must interleave video streams such that: (1) each sub-stream as well as its components are independently accessible, and (2) the seek and rotational latency while accessing any subset of the sub-streams is minimized. Whereas the former requirement can be met by storing sub-stream blocks on disk access boundaries (e.g., a sector), the latter requirement can be met by storing blocks of sub-streams that are likely to be accessed together adjacent to each other on disk. Observe that this placement policy is general, and can be used to interleave any

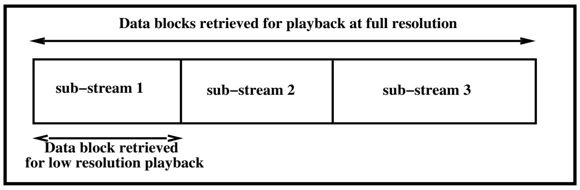

Figure 1 : Contiguous placement of sub-stream blocks

multi-resolution stream on the array. Thus, the placement algorithm can be employed to support applications such as multi-resolution video-on-demand, retrieval and multicasting of layered video over the Internet, etc.

To precisely describe the placement algorithm, consider a video server that exploits the periodic nature of video playback by servicing clients in terms of periodic rounds. During each round, the server retrieves a fixed number of video frames for each client. To ensure continuous playback, the number of frames accessed for each client during a round must be sufficient to meet its playback requirements. Let us assume that the server employs a disk array to store video streams. To effectively utilize the array bandwidth, the server interleaves the storage of each video stream among disks in the array. The unit of interleaving, referred to as a *media block* or a *striping unit*, denotes the maximum amount of logically contiguous data stored on a single disk. Each media block can contain a fixed number of storage units (i.e, bytes) or a fixed number of frames. If a video stream is compressed using a variable bit-rate (VBR) compression algorithm, then the sizes of frames will vary. Hence, if media blocks are assumed to be of fixed size, then each block will contain a variable number of frames. On the other hand, if each block contains a fixed number of frames, then media blocks will have variable sizes. Thus, depending on the type of media blocks used by the server, a request for a fixed number of frames in each round will require the server to access a fixed number of variable-size blocks or a variable number of fixed-size blocks [2, 14, 19].

If the server employs variable-size blocks for storing video streams, then it can minimize the seek and rotational latency incurred while servicing requests by: (1) storing all the frames of a sub-stream accessed during a round in the same media block, and (2) storing blocks of different sub-streams accessed in the same round adjacent to each other on disk (see Figure 1). Moreover, to ensure that each variable-size block is individually accessible, it must be stored at a disk-access unit boundary (e.g., a sector). Successive blocks of a sub-stream are then stored on consecutive disks in a round-robin manner.

In contrast, if the server employs fixed-size blocks to store media streams, then the number of frames in each sub-stream block will vary due to variable frame sizes. Consequently, the fixed-size block placement policy can not guarantee that sub-stream blocks stored adjacent to each other will always be accessed together. To maximize the probability that adjacent blocks are accessed together, the server must ensure that these blocks contain frames that will be requested in the same round. That is, sub-stream blocks stored adjacent to each other must contain an overlapping set of frames. To precisely describe a placement policy that achieves these objectives, let $\mathcal{F}_m$ and $\mathcal{L}_m$ denote the frame numbers of the first and the last frame, respectively, stored in a block of sub-stream m. Then the placement policy for fixed-size blocks is as follows:

1. Set $d = 1$
2. Let sub-stream i be the sub-stream whose next block has the lowest value of $\mathcal{L}_i$ (i.e., $\mathcal{L}_i = \min(\mathcal{L}_1, \mathcal{L}_2, ...,))$.
3. Place the next block of sub-stream i on disk d.
4. Place the next block of each sub-stream j that satisfies either $\mathcal{F}_i \leq \mathcal{F}_j \leq \mathcal{L}_i$ or $\mathcal{F}_j \leq \mathcal{F}_i \leq \mathcal{L}_j$ contiguously with the block of sub-stream i on disk d. {`* Store sub-stream blocks containing an overlapping set of frames adjacent to each other on disk d and skip storing blocks of sub-streams not satisfying this condition on disk d. *`}
5. Set $d = (d + 1) \bmod D$ and repeat steps (2)-(5) until all sub-stream blocks have been placed on the array.

Given that sub-streams are interleaved using the above placement policy, the server retrieves all sub-stream blocks that are stored contiguously if at least one of those blocks is requested by the client. The blocks that are not requested in the current round are buffered for use in future rounds. Since playback is sequential, the server is guaranteed that these sub-blocks would be requested in the next few rounds. Note that, such a placement policy is simpler to implement than

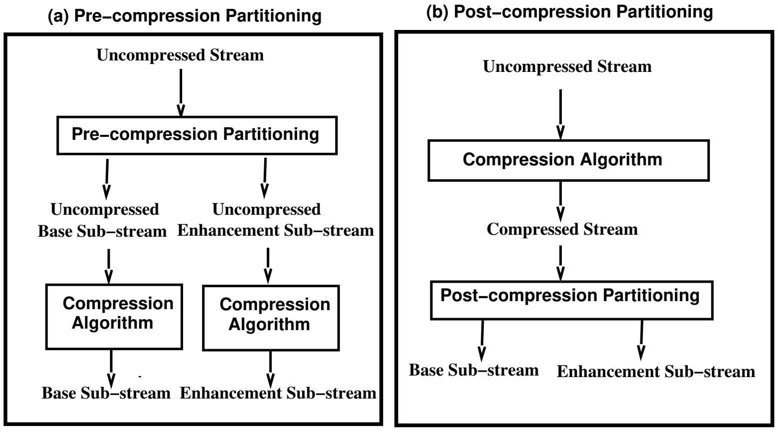

Figure 2 : Temporal partitioning techniques

the variable-size size placement policy (which must manage blocks of different sizes). However, since some blocks are retrieved ahead of their access, the policy incurs a higher buffer space requirement.

The server can reduce the buffer space requirement by employing a hybrid placement policy in which the block size can vary across sub-streams, but is fixed for a given sub-stream. To maximize performance, the block sizes chosen for different sub-streams must be proportional to their average bit-rates. Thus, if r_i and r_j denote the average bit rates of sub-streams i and j, the block size is chosen such that

$$\frac{B_i}{B_j} = \frac{r_i}{r_j}$$

where B_i and B_j denote the block sizes of sub-streams i and j, respectively. The server then stores successive blocks of each sub-stream on consecutive disks in a round-robin manner and stores blocks of all sub-streams stored on the same disk adjacent to each other. Since the block size chosen for a sub-stream is proportional to its average bit rate, such a policy ensures that sub-stream blocks stored adjacent to each other will contain approximately the same number of frames. This maximizes the probability of adjacent blocks being accessed together and reduces the buffer space requirement.

Thus, our placement algorithm enables a server to efficiently support playback of video streams at different resolution levels. In what follows, we describe techniques for efficiently supporting scan operations for multi-resolution video.

3 Efficient Support for Scan Operations

3.1 General Methodology

Consider a video server that supports scan operations by skipping frames. To achieve fast-forward or rewind at n-times the normal playback rate, the server must transmit every n^{th} frame to client sites. Since data is accessed from disk in terms of blocks rather than frames, in the worst case, the server will be required to access all the frames prior to selectively transmitting every n^{th} frame to client sites, thereby incurring an n-fold increase in the load. To address this limitation, the server can temporally partition each video stream into two sub-streams, such that the first sub-stream (referred to as the *base sub-stream*) contains every n^{th} frame, and the other sub-stream (referred to as the *enhancement sub-stream*) contains all the remaining frames. In such a scenario, to support fast-forward or rewind, the server will be required to access only the base sub-stream. During normal playback, on the other hand, the server will need to access and merge both sub-streams.

Observe that, such temporal partitioning can be accomplished either prior-to or after compression (referred to as *pre-compression* and *post-compression* partitioning, respectively) (see Figure 2). The effectiveness of these approaches, however, is dependent on the compression algorithm. In intra-frame compression algorithms (e.g., JPEG [15]), since successive frames are encoded and decoded independently, both pre-compression and post-compression partitioning techniques are logically equivalent. Moreover, for such compression algorithms, temporal partitioning does not have any adverse effects on compression efficiency. In inter-frame compression algorithms (e.g., MPEG [9] and MPEG-2 [11]), on the other hand, since the temporal redundancy between successive frames is used to efficiently encode the video stream, the degree of compression is critically dependent on the correlation between successive frames. Consequently, if the video stream is temporally partitioned prior to compression, then the resultant reduction in correlation between successive frames within each sub-stream may substantially degrade compression efficiency. While post-compression partitioning does not suffer from this drawback, the dependencies between frames introduced by the compression algorithm may complicate the partitioning process. This is because, for temporal partitioning to be effective, frames in the base sub-stream should be independently decodable (i.e., without decoding the corresponding enhancement sub-stream).

Regardless of the partitioning technique used, as we shall demonstrate later, the bit rate of streams compressed using inter-frame compression algorithms during scan is generally higher than that during normal playback. Since human perception is tolerant to a slight degradation in the video quality during scan, the bit rate requirement can be reduced by partitioning the base sub-stream in the chroma dimension into low-resolution and residual components, and utilizing only the low-resolution component for scan.

In summary, to efficiently support scan operations, an inter-frame compression algorithm must use a combination of temporal and chroma scalability techniques. The exact algorithm for deriving these streams using post-compression partitioning, however, is dependent on the idiosyncrasies of the inter-frame compression technique as well as the desired rate of fast-forward. In what follows, we show how these techniques may be employed to support fast-forward and rewind in the MPEG compression algorithm.

3.2 Supporting Fast-Forward in MPEG

3.2.1 The MPEG Compression Standard

The MPEG compression algorithm exploits the temporal and spatial redundancies present within a sequence of images to achieve a high degree of compression [9]. A group of pictures (GOP) in an MPEG stream is defined to be the smallest set of consecutive frames that is independently decodable. A GOP can contain three kinds of frames : (1) I frames (intra-coded without any reference to other frames), (2) P frames (predictively coded using a previous I or P frame), (3) B frames (bidirectionally interpolated from both the previous and the following I and/or P frame). By using temporal prediction from both the past and future, B frames achieve the highest compression ratios. The intra-coded I frames, on the other hand, achieve the lowest compression ratios.

To derive these types of frames, MPEG partitions each image into 16x16 pixel areas called macro blocks. Macro blocks belonging to I frames are independently encoded. Macro blocks belonging to B and P frames, on the other hand, are temporally interpolated from the corresponding reference frame(s), and the error between the actual and interpolated values is computed. The interpolation process also produces up to two motion vectors for each macro block, which denote the positions of the interpolated macro blocks in the reference frames. Regardless of the type of frame it belongs to, each macro block is partitioned into six 8x8 pixel blocks — four luminance and two chrominance blocks. Each of these 8x8 pixel blocks are transformed into the frequency domain using discrete cosine transform (DCT). The DCT un-correlates each pixel block into an 8x8 array of coefficients such that the most amount of energy is packed in a small number of low frequency coefficients. Whereas the lowest frequency coefficient (referred to as the DC coefficient) captures the average brightness and color within the pixel block, the remaining 63 coefficients (referred to as the AC coefficients) capture the details within the pixel block. The DC coefficients of successive blocks are difference encoded and then quantized. The AC coefficients within each block are quantized to remove high frequency coefficients, scanned in a zig-zag manner to obtain an approximate ordering from the lowest to the highest frequency, and finally run-length and entropy encoded. The motion vectors in the P and B frames are difference and entropy encoded in a lossless manner. Since P and B frames exploit temporal redundancies, they achieve much higher compression ratios as compared to I frames. Figure 3 depicts the main steps involved in the MPEG compression algorithm. The MPEG-2 standard extends this algorithm by supporting scalability in the spatial, temporal, and chroma dimensions [5, 11]. MPEG-2 also allows hybrid scalability—a technique in which a combination of these scalable modes may be used.

An important feature of the MPEG compression algorithm is that the encoding pattern (i.e., the relative frequency of occurrence of I, P, and B frames) can be controlled by the application. Specifically, an application can control the encoding pattern by specifying: (1) N, the distance between successive reference (i.e., I or P) frames, and (2) M, the distance between successive I frames. In what follows, we illustrate how this flexibility can be exploited to derive the base and the enhancement sub-streams without adversely affecting compression efficiency.

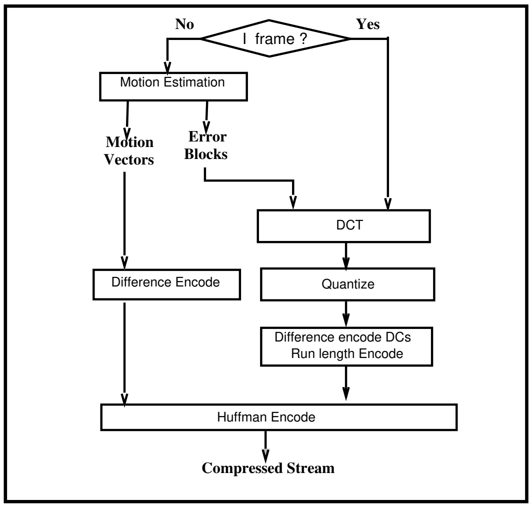

Figure 3 : The MPEG compression algorithm

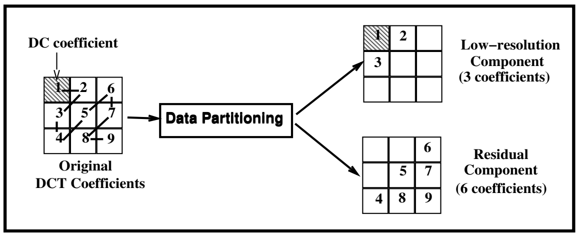

Figure 4 : The data partitioning technique

3.2.2 Deriving the Base and the Enhancement Sub-streams

Since only the base sub-stream is accessed during scan operations, one of the key requirements of post-compression temporal partitioning is that the base sub-stream must have no dependencies on frames contained in the enhancement sub-stream. That is, the base sub-stream must be independently decodable. Consequently, the base sub-stream can not contain a P or a B frame whose reference frame belongs to the enhancement sub-stream. To construct a base sub-stream that meets this requirement and supports fast-forward at n-times the normal playback rate, the encoding pattern must be constrained such that:

$$\begin{aligned} N = &\quad k \cdot n, \quad k = 1, 2, 3, ... \text{ and} \\ M = &\quad m \cdot N, \quad m = 1, 2, 3, ... \end{aligned}$$

Observe that the resultant encoding pattern (i.e., $IB^{N-1}PB^{N-1}P...$) contains $(N-1)$ B frames between successive I and P reference frames, and $(m-1)$ P frames between successive I frames. Such a stream, on temporal partitioning, yields: (1) a base sub-stream containing all the I and P frames, as well as every n^{th} B frame (if any) between successive reference frames; and (2) an enhancement sub-stream containing all the remaining B frames. For instance, to support fast-forward at twice the normal playback rate (i.e., for $n = 2$), the encoder can choose $N = 2$ and $M = 6$ (i.e., the encoding pattern of $IBPBPBI...$) and assign alternate frames to base and enhancement sub-streams, respectively. This yields a base stream containing all the I and P frames and an enhancement stream with all the B frames. Alternatively, the encoder can choose $N = 4$ and $M = 12$ (i.e., $IBBBPBBBPBBBI...$) as the encoding pattern. If B_1, B_2, and B_3 denote the three B frames between any two reference frames, respectively, then, after temporal partitioning, the base sub-stream would contain all the I, P, and B_2 frames, and the enhancement sub-stream would contain all the B_1 and B_3 frames. By recursively partitioning the base sub-stream in the temporal dimension, the server can support multiple fast-forward rates.

Observe that, an appropriate choice of the encoding pattern enables temporal partitioning to be performed *after* the spatial and temporal redundancies have been exploited by the motion-estimation and DCT stages in MPEG. This

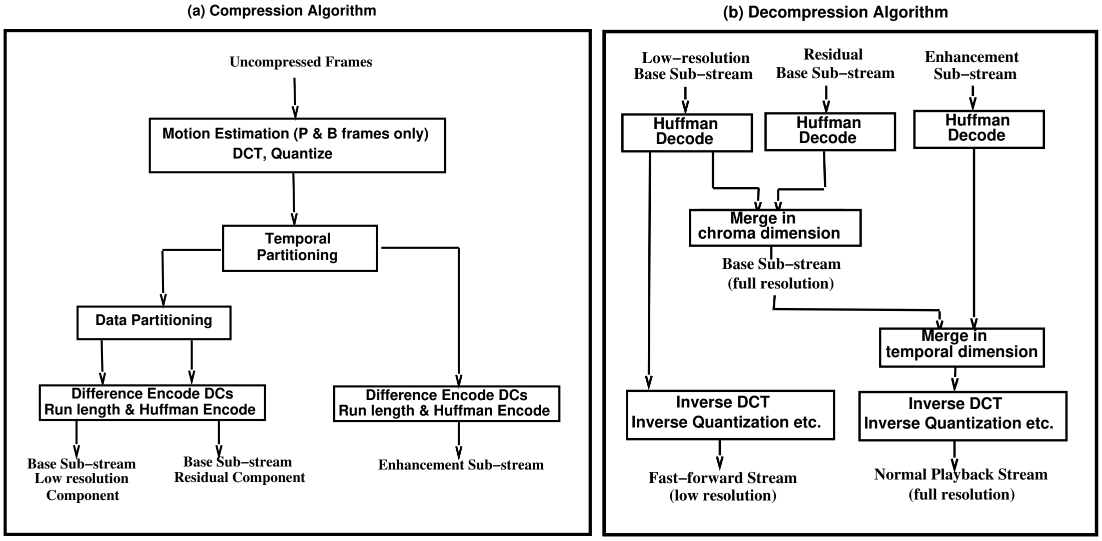

Figure 5 : The compression and decompression algorithms

eliminates any adverse effects of temporal partitioning on the compression efficiency. However, it yields a base sub-stream with a relatively higher frequency of I and P frames as compared to the original stream. Since during fast-forward, frames from only the base sub-stream are transmitted without changing the playback rate (thereby achieving an n-fold increase in the *effective* playback rate), the bit rate of the resultant stream is higher than that during normal playback. The bit rate can be reduced by partitioning the base sub-stream into low-resolution and residual components, and utilizing only the low-resolution component for fast-forward. Examples of such chroma partitioning techniques include the SNR scalability and the data partitioning modes of the MPEG-2 standard [11]. Whereas the SNR scalability mode creates the low-resolution and residual components by controlling the granularity of quantization, the data partitioning technique achieves a similar effect by explicitly dividing the frequency domain coefficients between the two components. Figure 4 illustrates the data partitioning technique for a 3x3 array of DCT coefficients. The key issue here is to determine an appropriate of number of DCT coefficients that must be included in the low-resolution component of the base sub-stream so as to ensure acceptable video quality during fast-forward without substantially increasing the bit rate.

In summary, our post-compression partitioning technique yields three components per stream: the low-resolution and the residual components of the base sub-stream and the enhancement sub-stream. Whereas only the low-resolution component of the base sub-stream is utilized during fast-forward, all three components are retrieved and merged during normal playback. The main steps involved in the compression and decompression algorithm are illustrated in Figure 5. The compression and decompression algorithms may be implemented by either using a MPEG-2 compliant codec (which supports hybrid scalability), or by suitably extending a MPEG-1 codec to incorporate partitioning in the temporal and chroma dimensions.

3.3 Supporting Rewind in MPEG

The post-compression partitioning scheme described in Section 3.2.2 can also be used to support rewind. However, the rewind operation differs from fast-forward in the following characteristics:

- During rewind, data blocks from the low resolution component of the base sub-stream are retrieved sequentially in the reverse order.
- The inter-frame dependencies in MPEG require a group of pictures to be decoded in the forward direction and displayed backwards. To illustrate, if $I_1P_2P_3I_4$ denotes a group of pictures, then even though the frames are displayed in the order $I_4P_3P_2I_1$ during rewind, they must still be decoded in the order $I_1P_2P_3I_4$ due to frame dependencies.

- Since data blocks are retrieved in the reverse order during rewind, the first frame of the GOP arrives after all other frames in the GOP have arrived at the client. Consequently, the decoder must wait for the entire GOP to arrive before it can begin decoding the stream, thereby introducing a initiation latency at the client. Moreover, since a GOP is decoded in the forward direction and displayed backwards, the entire GOPs must be decoded and buffered before a frame can be displayed. This increases the buffer requirements at the client and adds to the initiation latency.

3.4 Discussion

In this section, we compare our scheme with other schemes for scan operations proposed in the literature. Specifically, we compare our scheme with (1) a scheme that displays frames at n-times the normal playback rate, (2) a scheme that displays every n-th frame and skips intermediate frames, and (3) a scheme that maintains an independently encoded stream for scan operations.

Load on the server and the network

Displaying frames at n-times the playback rate during scan causes the server to retrieve and transmit n-times as many frames, yielding a n-fold increase in the load imposed on the server and the network. Supporting scan by skipping intermediate frames and displaying every n-th frame can be done either at the server or at the client. In the former approach, the server retrieves and transmit n-times as many frames; the client decodes every n-th frame and discards remaining frames. This imposes an n-fold increase in the server and the network load. In the latter approach, the server retrieves and transmits every n-th frame from the video stream. Since skipping frames causes a relative increase in the number of I and P frames during scan, this approach also imposes an overhead on the server and the network. To illustrate, displaying alternate frames from a stream encoded as $IBPBPBP$ yields a stream with the sequence $IPIPIP$. I and P frames are substantially larger than B frames, yielding an increase in bit rate during scan. Although smaller than a n-fold increase, our experiment indicate that this overhead can be as large as 70% (see Section 5.2 and Table 2). One approach to compensate for this increased bit-rate is to lower the frame rate during scan. For instance, fast-forward at four times the playback rate can be achieved by displaying every eight frame at half the normal playback rate. Both reducing the frame rate and skipping over a large number of intermediate frames can cause scan to be jerky. In contrast, our approach reduces the picture quality during fast-forward (by displaying only the low resolution component of the base sub-stream) to maintain the same bit-rate. Thus, the two schemes have different tradeoffs. One reduces the frame rate during scan while maintaining the same picture quality (causing scan to be jerky). The other reduces the picture quality during scan while maintaining the same frame rate (resulting in a lower resolution). Finally, by carefully encoding an independent stream for scan, it is possible to support fast-forward and rewind without any overhead on the server or the network.

Storage space overhead

By employing post-compression partitioning, our scheme eliminates any adverse degradation in compression efficiency for various components of the video stream. Hence, the storage space overhead of our scheme (as compared to the original stream) is negligible. Since schemes that display frames at a higher rate as well as schemes that skip frames do not require any modifications to the video stream, they do not impose any storage space overhead on the server. In contrast, maintaining an independently encoded stream to support scan imposes a substantial storage space overhead on the server.

Modifications to existing decoders

Although our encoding technique stores each sub-stream separately on the array, it is not essential to do so to support scan operations. The encoder can employ the hybrid scalability mode of MPEG-2 to produce sub-streams and multiplex these sub-streams within a single program/transport stream, resulting in a MPEG-2 compliant stream.[1] The server

[1]It is important to distinguish a MPEG-2 compliant stream from one that conforms to a MPEG-2 profile. Since the number of combinations of encoding parameters that can generate valid MPEG-2 streams is large, the standard defines various commonly used subsets of these parameters and refers to them as profiles. A decoder that can decode a stream corresponding to a profile

can then retrieve and transmit only the low resolution component of the base sub-stream during scan operations. By displaying this sub-stream component at n times the normal playback rate, MPEG-2 decoders that support hybrid scalability can also support scan operations. However, since data is accessed from disk in terms of blocks rather than frames, the server may have to retrieve more data than is necessary, thereby imposing an additional load during scan. Our placement scheme overcomes these drawbacks by interleaving each component sub-stream separately on the array so as to allow independent access to a component. This results in a deviation from the MPEG-2 standard, since each sub-stream is now stored within a separate program stream, with all program streams sharing a common time base.

All other schemes also require the decoder to be modified (albeit to a smaller extent) to understand transitions between playback and scan and to decode the incoming stream appropriately (for instance, MPEG time codes that determine the display instant of each frame must be interpreted appropriately during scan operations).

Thus, different schemes have different tradeoffs. Our scheme imposes negligible overhead on the server and the network, but requires modifications to existing codecs. Existing schemes require either substantial additional disk and network bandwidth or additional storage space to support scan, and require only minor modifications to codecs. The choice of an appropriate scheme depends on the application environment.

Given that video streams generated using the above encoding technique, the server can use the placement algorithm described in Section 2 to interleave sub-stream blocks on disk. In such a scenario, the load on the array depends on two factors: (1) change in bit-rate of a client when a client switches to scan, and (2) the relative number of clients in playback and scan modes. In what follows, we present an analytical model to determine the effects of these factors on the load on the array. Since analysis for fast-forward and rewind are similar, we present the model only for fast-forward.

4 Determining the Overhead of Fast-forward Operation

Consider a video server that services N clients, each retrieving a video stream (say $S_1, S_2, ..., S_N$, respectively). Let $f_1, f_2, ..., f_N$ denote the number of frames retrieved from streams $S_1, S_2, ..., S_N$, respectively, during each round. To support fast-forward at n times the normal playback rate, let each stream S_i be partitioned into three components, namely, the low resolution and the residual components of the base sub-stream and the enhancement sub-stream (denoted by S_i^1, S_i^2, and S_i^3, respectively). During normal playback, the server retrieves $\frac{f_i}{n}$, $\frac{f_i}{n}$, and $\frac{(n-1)\cdot f_i}{n}$ frames of sub-stream S_i^1, S_i^2, and S_i^3, respectively, in each round, which are then merged to obtain f_i frames of the original stream S_i. During fast-forward, however, the server retrieves f_i frames of only the low resolution component of the base sub-stream (i.e., S_i^1) in each round. Let each sub-stream be interleaved on disk using *variable-size blocks*. Then, each variable-size sub-stream block contains all the frames accessed from that sub-stream in a round during normal playback. Thus, depending upon whether client i is in the playback or the fast-forward mode, it accesses either a single block from each of the three components of a stream or n blocks from the low-resolution component of the base sub-stream during a round. Hence, the set of clients accessing a disk moves in a *lock-step* manner. Specifically, clients that access disk i in the playback mode during a round will access disk $(i+1)$ mod D in the following round, and those that access disks $i, (i+1)$ mod $D, \cdots, (i+n-1)$ mod D in the fast-forward mode during a round will access $(i+n)$ mod $D, \cdots, (i+n+1)$ mod $D, \cdots, (i+2n-1)$ mod D in the following round. Although the set of clients in the fast-forward and playback modes move in a lock-step manner, the set of clients in the fast-forward mode move n-times faster than those in normal playback. Consequently, the number of blocks accessed during a round can vary from one disk to another. Hence, some disks can be more heavily loaded than others, resulting in different service times for different disks. This load imbalance can cause the service time of some of the heavily loaded disks to exceed the round duration, causing playback discontinuities at client sites.

Recall that, the chroma domain partitioning used by our encoding technique ensures that the bit rate of the stream does not significantly change during fast-forward. Consequently, a client accesses approximately the same amount of data in a round, regardless of whether it is in the playback of the fast-forward mode. The only difference between the two modes is that instead of accessing a single chunk of data containing blocks of all three components of a stream, the client accesses n smaller blocks containing only the low resolution component of the base sub-stream. Thus, the fast-forward operation increases the number of blocks accessed from the server, thereby increasing the seek and rotational latency overheads, and hence, the service time of disks. The resulting increase in the load on the server can

---

is said to conform to that profile. Streams produced using our encoding technique do not conform to any MPEG-2 profile; nevertheless they are MPEG-2 compliant.

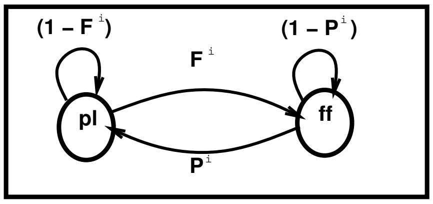

Figure 6 : The Markov model for the behavior of a client

cause the the service time of some of the heavily loaded disks to exceed the round duration, causing discontinuities in video playback.

To minimize the frequency of playback discontinuities caused by: (1) the load imbalance due to the clients in the fast-forward mode, and (2) the increased seek and rotational latency overheads in the fast-forward mode, the server must set aside some contingency disk bandwidth so that the service time of the most heavily loaded disk in the array rarely exceeds the round duration. To estimate the amount of contingency bandwidth that must be reserved, the server must be able to determine the impact of a playback to fast-forward transitions on the service time of the most heavily loaded disk. We now present an analytical model that uses the probability of a playback to fast-forward transition and the change in stream bit rate after such a transition to determine the service time on the most heavily loaded disk.

To precisely describe the model, let us assume that a client in the playback mode can switch to the fast-forward mode at any random instant and vice versa, and that such a behavior can be modeled using a two state Markov chain [17] (see Figure 6). Let F^i denote the probability of switching from playback to fast-forward, and P^i denote the probability of switching from fast-forward to playback mode for client i as shown in the figure. If P_s^i and F_s^i denote the steady state Markov probability of client i being in playback and fast-forward, respectively, then using the theory of Markov chains we get $P_s^i = \frac{P^i}{P^i+F^i}$ and $F_s^i = \frac{F^i}{P^i+F^i}$ [17].

Let the random variable $\mathcal{X}_i^j$ denote the number of blocks accessed by client i from disk j in a round. Then assuming that $n \leq D$,

$$\mathcal{X}_i^j = \begin{cases} 1 & \text{if client } i \text{ accesses disk } j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Clearly, $\mathcal{X}_i^j = 1$ only if client i accesses a block from disk j during either playback or fast-forward. Furthermore, client i accesses a block from disk j during fast-forward only if the first of the n media blocks requested during a round is retrieved from disk j or any one of the $n-1$ previous disks. Since retrieval of each individual stream from disk proceeds in lock-step, it is possible to exactly compute the set of disks being accessed by the stream. However, since transitions from fast-forward to playback and vice-versa occur at random instants, and since the duration for which a client stays in the fast-forward mode is a random variable, after a small number of such transitions, the first media block is equally likely to be accessed from any of the disks in the array. Consequently,

$$P(\mathcal{X}_i^j = 1) = p_i = P_s^i \cdot \frac{1}{D} + F_s^i \cdot \frac{n}{D} \tag{2}$$

Thus, if the random variable $\mathcal{B}^j$ denotes the total number of blocks accessed from disk j, then

$$\mathcal{B}^j = \sum_{i=1}^{N} \mathcal{X}_i^j \tag{3}$$

Assuming that playback can begin from a random point in the media stream and that playback to fast-forward transitions occur independently of each other, the set of media blocks accessed by different streams are independent of each other. That is, $\mathcal{X}_i^j$'s are independent. Hence, we get:

$$Z(\mathcal{B}^j) = \prod_{i=1}^{N} Z(\mathcal{X}_i^j) \tag{4}$$

where $Z(\mathcal{B}^j)$ and $Z(\mathcal{X}_i^j)$ are the z-transforms [2] of the random variables $\mathcal{B}^j$ and $\mathcal{X}_i^j$, respectively [17].[3] Then the number of blocks accessed from the most heavily loaded disk is given by

$$\mathcal{B}_{\max} = \max(\mathcal{B}^1, \mathcal{B}^2, \cdots, \mathcal{B}^D)$$

Note that, sub-stream blocks are interleaved on the array in a round-robin manner and a client can access up to n blocks in a round. Hence, the number of blocks accessed from a disk is not independent of the load on its neighboring disks. Since the precise dependence of these random variables on each other is difficult to characterize, and since the maximum of D dependent random variables is difficult to compute, as an approximation we assume that $\mathcal{B}^j$s are independent of each other. Later in this paper, we demonstrate that this approximation does not cause any inaccuracies in the predictions of the model. Then, the distribution of $\mathcal{B}_{\max}$ can be computed as

$$F_{\mathcal{B}_{\max}}(x) = F_{\mathcal{B}^1}(x) \cdot F_{\mathcal{B}^2}(x) \cdots F_{\mathcal{B}^D}(x) \tag{6}$$

where $F_{\mathcal{B}_{\max}}(x)$ denotes the probability distribution function of $\mathcal{B}_{\max}$.

Having determined the distribution of the number of blocks accessed from the most heavily loaded disk, the service time of the disk can then be computed by using a disk model. We use one such model that has been proposed in the literature [12, 19]. The service time to access $\widehat{\mathcal{B}}_{\max}$ blocks as predicted by the disk model is:

$$\tau = \widehat{\mathcal{B}}_{\max} * (t_s + t_r + t_t) \tag{7}$$

where t_s and t_r denote the expected seek time and rotational latency incurred while accessing a block from disk, respectively and $\widehat{\mathcal{B}}_{\max} = E(\mathcal{B}_{\max})$. Assuming that the $\widehat{\mathcal{B}}_{\max}$ blocks are equally spaced across the $\mathcal{C}$ cylinders of a disk, we define $t_s = t_{seek}\left(\left\lfloor \frac{\mathcal{C}}{\widehat{\mathcal{B}}_{\max}} \right\rfloor\right)$, where:

$$t_{seek}(x) = \begin{cases} 0 & \text{if } x = 0 \\ a\sqrt{x-1} + b(x-1) + c & \text{otherwise} \end{cases} \tag{8}$$

and a, b, and c are constants (determined using physical characteristics of a disk) [12]. The average rotational latency, t_r, is defined to be half of the maximum rotational latency. The transfer time, t_t, on the other hand, can be computed as a weighted average of the transfer times during playback and fast-forward, where the weights are the total number of blocks retrieved during playback and fast-forward, respectively. Let t_t^p denote the expected transfer time to retrieve a contiguous chunk of data consisting of one block from each of the three components during playback and t_t^f denote the expected transfer time to retrieve a block of the low resolution component of the base sub-stream during fast-forward. Since N clients access the server, the number of playback blocks requested in a round is $N \cdot P_s$ and the number of fast-forward blocks is $N \cdot F_s \cdot n$, where P_s and F_s denote the steady state probabilities of being in playback and fast-forward, respectively. [4] Then, the expected transfer time of a block is

$$t_t = \frac{(N \cdot P_s) \cdot t_t^p + (N \cdot F_s \cdot n) \cdot t_t^f}{N \cdot P_s + N \cdot F_s \cdot n} = \frac{P_s \cdot t_t^p + F_s \cdot n \cdot t_t^f}{P_s + F_s \cdot n} \tag{9}$$

Thus, given the server configuration (i.e., disk array characteristics, number of clients, and their data rate requirements) and the steady-state Markov probabilities, Equation (7) derives the expected service time for the most heavily loaded disk.

---

[2] The z-transform of a random variable $\mathcal{U}$ is the polynomial $Z(\mathcal{U}) = a_0 + za_1 + z^2a_2 + \cdots$ where the coefficient of the i^{th} term in the polynomial represents the probability that the random variable equals i. That is, $P(\mathcal{U} = i) = a_i$. If $\mathcal{U}_1, \mathcal{U}_2, ..., \mathcal{U}_n$ are n independent random variables, and $\mathcal{Y} = \sum_{i=1}^{n} \mathcal{U}_i$, then $Z(\mathcal{Y}) = \prod_{i=1}^{n} Z(\mathcal{U}_i)$. The distribution of $\mathcal{Y}$ can then be computed using a polynomial multiplication of the z-transforms of $\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n$

[3] In the event that all the clients are homogeneous (i.e., $p_1 = p_2 = \cdots = p_N = p$), $\mathcal{B}^j$ reduces to a binomial random variable, yielding:

$$P(\mathcal{B}^j = x) = \binom{N}{x} p^x (1-p)^{N-x}, \quad 0 \le x \le N \tag{5}$$

[4] If the steady state probabilities for different clients are different, then $P_s = \frac{1}{N} \cdot \sum_{i=1}^{N} P_s^i$ and $F_s = \frac{1}{N} \cdot \sum_{i=1}^{N} F_s^i$.

Disk capacity	2 GBytes
Number of disks in the array	16
Bytes per sector	512 KB
Sector per track	99
Tracks per cylinder	21
Cylinders per disk	2627
Minimum seek time	1.7 ms
Maximum seek time	22.5 ms
Maximum rotational latency	11.1 ms

Table 1 : Disk Parameters of Seagate-Elite3 disk used in the paper

Given the model for determining the service time of the most heavily loaded disk, we can compute the fraction of the disk bandwidth (referred to as the contingency bandwidth) that must be reserved to prevent server saturation due to playback to fast-forward transitions. To do so, let τ_1 denote the service time of the most heavily loaded disk obtained for $F^i = 0$ (i.e., when all clients are in the playback mode), and let τ_2 denote the service time of the most heavily loaded disk obtained using the specified values of Markov probabilities F^i and P^i. Let $\mathcal{R}$ denote the duration of a round. Then the contingency bandwidth, defined as the fraction of the disk bandwidth that must be reserved by the server to accommodate any load increase due to playback to fast-forward transitions, can be computed as

$$\mathcal{C} = \max\left(0, \frac{\tau_2 - \tau_1}{\mathcal{R}}\right) \tag{10}$$

5 Experimental Evaluation

To evaluate our placement algorithm and encoding technique, we have developed a prototype encoder and an event-driven disk array simulator. The data for our experiments and simulations was generated by digitizing several video clips, the largest one of which was a ten minute sequence of a Frasier episode[5] which yielded 18,000 frames. We used our encoder to encode these video clips; the resulting streams could support both multi-resolution video playback as well as scan operations.

5.1 Evaluation of the Placement Algorithm

The simulation environment for evaluating the placement algorithm consisted of an array of 16 disks. The characteristics of each disk in the array are shown in Table 1. First, we interleaved each sub-stream independently across disks in the array using fixed-size blocks of size 32KB. Each client accessing the array was assumed to retrieve all sub-streams of a randomly selected video stream at a playback rate of 30 frames/s. The duration of a round was set to 1s. We varied the number of clients accessing the array and measured the average service time of a disk in a round.

Next, we interleaved each video stream using our placement algorithm for fixed-size blocks and repeated the experiment. As shown in Figure 7(a), the average service time of a disk was smaller by about 40% when our placement algorithm was used. This is because the placement algorithm places sub-stream blocks that are likely to be accessed in the same round adjacent to each other on disk and thereby reduces seek and rotational latency overheads.

We repeated the above experiment for variable-size blocks. Again, our placement algorithm resulted in smaller service time per disk for a given workload (see Figure 7(b)). This demonstrates the efficacy of our placement algorithm in supporting multi-resolution video playback.

[5]The Frasier series is a copyright of NBC.

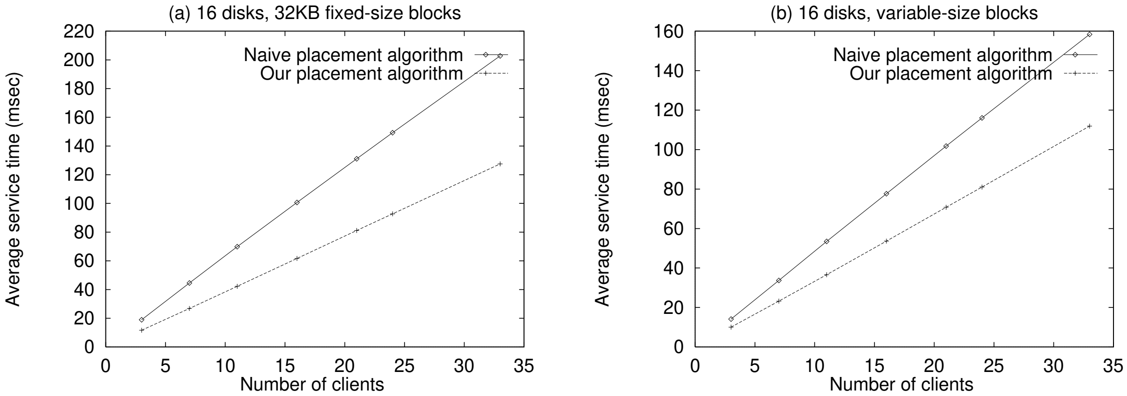

Figure 7 : Service time of a disk during normal playback: Our placement algorithm yields a smaller service time per disk as compared to one that interleaves each sub-stream independently across the array.

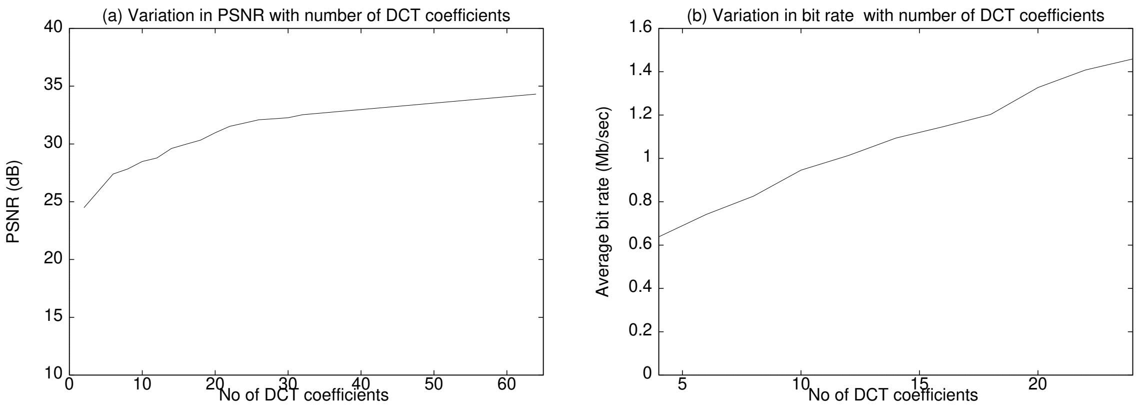

Figure 8 : Variation of PSNR and fast-forward bit rates with number of DCT coefficients for the Frasier sequence.

5.2 Evaluation of the Encoding Technique and the Analytical Model

5.2.1 Creating the Base Sub-stream

For encoding each video stream, we assumed a fast-forward speed of twice the normal playback rate. Since $n = 2$, each video stream was encoded with $N = 4$ and $M = 12$ as the encoding pattern (i.e, $IBBBPBBBPBBBI$). Let B_1, B_2 and B_3 denote the three B frames between any two reference frames. Then, the encoder assigned the I, P and B_2 frames to the base sub-stream and the B_1 and B_3 frames to the enhancement sub-stream. Since the relative frequency of I and P frames in the base sub-stream was higher than that in the original stream, using the base sub-stream for fast-forward yielded an increase in the bit rate (see Table 2). For streams under consideration, the increase in bit rate ranged from 40-70%. Hence, to lower the bit rate during fast forward, the encoder further partitioned the base sub-stream in the chroma domain to create low resolution and residual components.

5.2.2 Determining Parameters for Data Partitioning of the Base Sub-stream

The key issue in data partitioning is to determine the number of DCT coefficients that must be included in the low resolution component of the base sub-stream so as to ensure acceptable video quality during fast-forward without any substantial increase in the bit rate. To quantify the quality of a decompressed image, we use the *Peak Signal to Noise Ratio (PSNR)* as our metric. For an AxB pixel-image with a resolution of r bits/pixel, if $p(x, y)$ and $p'(x, y)$ denote the pixel values at location (x, y) in the original and the decompressed images, respectively, then the PSNR of the image is defined as:

$$PSNR = 10.log_{10}(\frac{(2^r-1)^2}{\sigma^2}) \text{ dB}$$

Stream	Normal bit rate	Base sub-stream bit rate	Number of coefficients in the low resolution component					
			18		20		22	
			PSNR	bit rate	PSNR	bit rate	PSNR	bit rate
Hockey	1.61	2.12	33.1	1.57	34.5	1.67	35.7	1.7
Flintstones	1.62	2.77	24.97	1.56	27.6	1.7	30.29	1.8
Frasier	1.35	1.96	30.33	1.20	30.97	1.32	31.52	1.4
News	1.48	2.1	24.46	1.2	26.48	1.31	28.46	1.4

Table 2 : PSNR and bit rates of MPEG streams during fast-forward for various sequences. The base sub-stream bit rate indicates the bit rate during fast forward without chroma partitioning. The bit rate and PSNR values have units of Mb/s and dB, respectively.

$$\text{where } \sigma = \sum_{x=1}^{A} \sum_{y=1}^{B} (p(x,y) - p'(x,y))^2$$

Figure 8(a) shows the variation in the average video quality during fast-forward obtained by varying the number of DCT coefficients in the low-resolution component. It shows that the video quality improves by increasing the number of coefficients contained in each DCT block. However, this improvement in video quality is at the expense of an increase in the bit rate requirement during fast-forward (see Figure 8(b)).

Depending on the relative importance of the bit rate and the video quality during fast-forward, the encoder can use one the following two metrics to determine the number of coefficients that must be included in the low resolution component of the base sub-stream: (1) choose the number of coefficients such that the bit rate during fast-forward approximately equals that during normal playback, or (2) choose the number of coefficients such that the video quality during fast-forward is at least equal to a specified minimum value. In the former approach, the amount of contingency bandwidth that must be reserved by the server is small. However, the picture quality can vary from one stream to another. In contrast, the latter approach yields a minimum acceptable video quality, possibly at the expense of an increase in the bit rate during fast-forward. To illustrate these tradeoffs, consider the streams listed in Table 2. If the number of coefficients in the low resolution component of the base sub stream is chosen so as to optimize bit rate, then the encoder would choose 18 coefficients for both the Hockey and Flintstones sequences (so that the bit rate during fast-forward approximately equals that during normal playback). However, the picture quality obtained during fast-forward for these sequences is 33.1 dB and 24.97 dB respectively. On the other hand, if the number of coefficients is chosen so as to obtain a minimum video quality (say 30 dB), then the number of coefficients included in the low resolution component of the two sequences would be 18 and 22, respectively. While this does not cause an increase in bit rate for the Hockey sequence, it causes an increase in bit rate for the Flintstones sequence during fast-forward.

For the streams under consideration, our experiments indicate that by assigning 18-20 coefficients to the low resolution component of the base sub-stream, we can get acceptable video quality (about 27dB) without any significant increase in the bit rate during fast-forward. Consequently, the average size of a block retrieved during fast-forward is about half of that retrieved during normal playback (see Figure 9). Since the server retrieves twice the number of blocks in each round during fast-forward, there is no significant change in the stream bit rate.

5.2.3 Determining the Contingency Bandwidth

Regardless of whether data partitioning is performed to optimize the bit rate or the picture quality, the server must reserve contingency bandwidth to accommodate any increase in load due to playback to fast-forward transitions. Figure 10(a) depicts the variation in the service time of the most heavily loaded disk obtained from the analytical model and trace-driven simulations with increase in the steady state Markov probability of a client being in fast-forward mode. The figure shows that the service times predicted by the model are within 3-5% of those obtained from simulations. Moreover, when the low resolution component of the base sub-stream contains 20 DCT coefficients, the service time of the most heavily loaded disk is independent of the probability of being in fast-forward (and hence, the number of clients in the fast-forward mode). Thus, the fast-forward operation imposes a negligible overhead on the server.

Figure 10(b) shows the contingency bandwidth that must be reserved for different values of the probability of being in the fast-forward mode. When the low resolution component of the base sub stream contains 20 coefficients, the bit rate during fast-forward approximately equals that during normal playback. Consequently, the overhead on

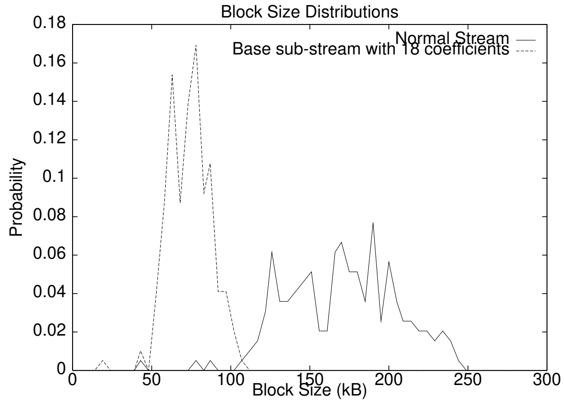

Figure 9 : Comparison of block size distribution of the base sub-stream with the normal stream

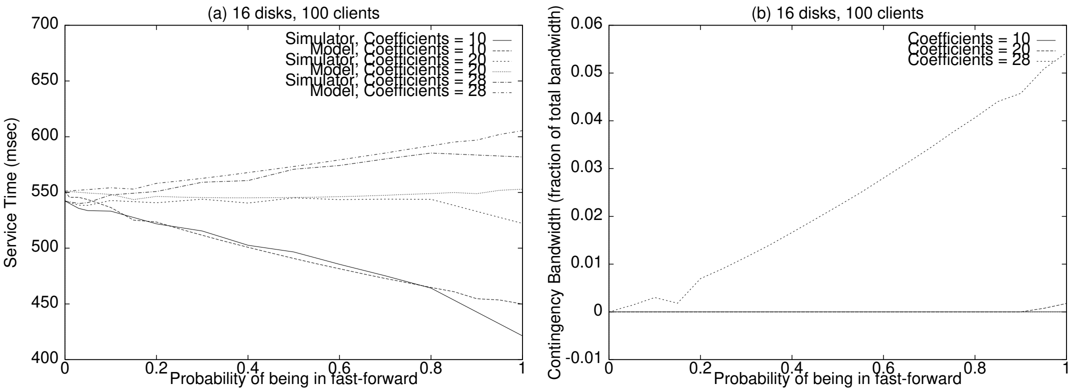

Figure 10 : Variation in service times and the contingency bandwidth obtained by changing the probability of fast-forward.

the server during fast-forward and the contingency bandwidth requirement are very small. Increasing the number of coefficients contained in the low resolution component beyond 20 enhances picture quality during fast-forward, at the expense of an increased bit rate and larger contingency bandwidth requirement. In contrast, decreasing the number of coefficients in the low resolution component below 20 cause the bit rate during fast-forward to be lower than that during normal playback. Hence, the contingency bandwidth requirement reduces to zero at the expense of poor picture quality during fast-forward.

6 Concluding Remarks

In this paper, we presented a placement algorithm that interleaves multi-resolution video streams on a disk array and ensures that: (1) each sub-stream is independently accessible, and (2) the seek and rotational latency overhead incurred in accessing any subset of these sub-streams is minimized. This enables a server to efficiently support playback of video streams at different resolution levels. We then presented an encoding technique which when combined with our placement algorithm enables a server to efficiently support scan operations. Our encoding technique combines scalability in the temporal and chroma dimensions to ensure that scan operations do not impose any additional overhead as compared to normal playback. We presented an analytical model for predicting the effects of making a transition from playback to fast-forward on the array load, and used it to compute the contingency bandwidth that must be reserved by the server to accommodate any load increase due to the fast-forward operation. Our experimental results demonstrate that: (1) the placement algorithm substantially reduces seek and rotational latency overhead during playback, and (2) by exploiting the characteristics of the compression algorithm and human perceptual tolerances, a server can support interactive scan operations without any significant increase in bit rate.

Although presented in the context of interactive scan operations, our analytical model can also be used to perform admission control in the presence of interactive scan operations. Our algorithms are being incorporated into Symphony—a multimedia file system being developed in our research laboratory.

REFERENCES

[1] D. Anderson, Y. Osawa, and R. Govindan. A File System for Continuous Media. *ACM Transactions on Computer Systems*, 10(4):311–337, November 1992.

[2] E. Chang and A. Zakhor. Scalable Video Placement on Parallel Disk Arrays. In *Proceedings of IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology, San Jose*, February 1994.

[3] M. S. Chen and D. D. Kandlur. Downloading and Stream Conversion: Supporting Interactive Playout of Videos in a Client Station. In *Proceedings of the International Conference On Multimedia Computing and Systems (ICMCS), Washington D.C.*, pages 73–80, May 1995.

[4] M. S. Chen, D. D. Kandlur, and P. S. Yu. Support for Fully Interactive Playout in a Disk-Array-Based Video Server. In *Proceedings of the Second International Conference on Multimedia*, pages 391–398, October 1994.

[5] T. Chiang and D. Anastassiou. Heirarchichal Coding of Digital Television. *IEEE Communications*, 32(4):38–45, May 1994.

[6] T. Chiueh and R. Katz. Multi-Resolution Video Representation for Parallel Disk Arrays. In *Proceedings of the ACM Multimedia'93, Anaheim, CA.*, pages 401–409, August 1993.

[7] J. K. Dey-Sircar, J. D. Salehi, J. F. Kurose, and D. Towsley. Providing VCR Capabilities in Large-Scale Video Servers. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 25–32, October 1994.

[8] W. Feng, F. Jahanian, and S. Sechrest. Providing VCR Functionality in a Constant Quality Video-On-Demand Transportation Service. In *Proceedings of the International Conference On Multimedia Computing and Systems (ICMCS), Hiroshima, Japan*, pages 127–135, June 1996.

[9] D. Le Gall. MPEG: A Video Compression Standard for Multimedia Apllications. *Communications of the ACM*, 34(4):46–58, April 1991.

[10] J. Gemmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, 1992.

[11] International Organisation for Standardisation. *Information Technology - Generic Coding of Moving Pictures and Associated Audio Systems: Systems, Video and Audio, International Standard (MPEG2), ISO/IEC 13818,*, November 1994.

[12] E.K. Lee and R.H. Katz. An Analytic Performance Model for Disk Arrays. In *Proceedings of the 1993 ACM SIGMETRICS*, pages 98–109, May 1993.

[13] J. C. S. Lui and K. W. Law. Load Balancing and VCR Functionalities Support via Subband Coding Techniques. In *Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 77–80, April 1995.

[14] S. Paek, P. Bocheck, and S. F. Chang. Scalable MPEG2 Video Servers with Heterogeneous QoS on Parallel Disk Arrays. In *Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, April 1995.

[15] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[16] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: A Disk Storage System for Video and Audio Files. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 393–400, August 1993.

[17] K. S. Trivedi. *Probability & Statistics With Reliability, Queuing, And Computer Science Applications*. Prentice-Hall, Inc., 1982.

[18] H. M. Vin, P. Goyal, A. Goyal, and A. Goyal. A Statistical Admission Control Algorithm for Multimedia Servers. In *Proceedings of the ACM Multimedia'94, San Francisco*, pages 33–40, October 1994.

[19] H.M. Vin, S.S. Rao, and P. Goyal. Optimizing the Placement of Multimedia Objects on Disk Arrays. In *Proceedings of the Second IEEE International Conference on Multimedia Computing and Systems, Washington, D.C.*, pages 158–165, May 1995.

[20] P. Yu, M.S. Chen, and D.D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. *Proceedings of Third International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego*, pages 38–49, November 1992.

Prashant Shenoy is a doctoral candidate in the Department of Computer Sciences at the University Texas at Austin. His research topics include multimedia file systems and operating systems. He is involved in the design and implementation of an integrated file system being built at Distributed Multimedia Computing Laboratory, UT Austin.

Harrick M. Vin received his Ph.D. in Computer Science from the University of California at San Diego in 1993. He is currently an Assistant Professor of Computer Sciences, and the Director of the Distributed Multimedia Computing Laboratory at the University of Texas at Austin. His research interests are in the areas of multimedia systems, high-speed networking, mobile computing, and large-scale distributed systems. Over the past 5 years, he has co-authored more than 55 papers in leading journals and conferences in the area of multimedia systems. He has been a recipient of several awards including the National Science Foundation CAREER award, IBM Faculty Development Award, AT&T Foundation Award, IBM Doctoral Fellowship, NCR Innovation Award, and the San Diego Supercomputer Center Creative Computing Award.