Efficient Support for Scan Operations in Video Servers

Prashant J. Shenoy and **Harrick M. Vin**
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) 471-9738, Fax: (512) 471-7866

Abstract

In this paper, we present an algorithm that integrates scalable compression techniques with placement algorithms for disk-arrays to provide efficient support for interactive scan operations (i.e., fast-forward and rewind) in video servers. We demonstrate that by suitably exploiting the characteristics of video streams and human perceptual tolerances, the overhead of such interactive operations can be substantially reduced. We present an analytical model for analyzing the impact of the fast-forward operation on the performance of the disk-array-based server. We validate the model through extensive simulations and analyze our results.

1 Introduction

1.1 Motivation

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, product announcements, customer support, video on-demand, etc.). In its simplest configuration, such multimedia services will comprise of multimedia information servers connected to client sites via high-speed networks. Multimedia servers will digitally store media information such as entertainment movies, educational documentaries, and advertisements, on a large array of extremely high-capacity storage devices such as optical or magnetic disks. Clients will dial-up the service via high-speed networks and request the retrieval of media objects stored at the server for real-time playback. The retrieval can be interactive, in the sense that clients can stop, pause, resume, fast-forward, or rewind the presentation.

Since multimedia objects containing digital audio and video streams significantly differ in characteristics (e.g., size, format, tolerance to errors, real-time playback requirements) from textual and numeric data, the design of video servers that can provide service to a large number of clients simultaneously poses interesting research challenges. Whereas techniques for meeting the real-time playback requirement of video streams have been extensively studied [1, 7, 11, 13, 14, 16], methods for efficiently supporting interactive scan operations have not been adequately investigated.

1.2 Relation to Previous Work

In general, schemes that support fast-forward and rewind either: (1) display frames at a rate higher than normal playback [5], or (2) skip frames [3]. In the former scheme, fast-forward at n-times the normal playback rate requires n-times as many frames to be retrieved (as compared to the normal playback), yielding an n-fold increase in the load on the server. If the available bandwidth at the server is not sufficient for meeting the increased requirements, then the client request must be delayed until the necessary resources become available. To minimize the expected waiting time, the server may be required to set aside some bandwidth (both disk and network) to accommodate such dynamic transitions from playback to fast-forward. The additional bandwidth that must be set aside, however, is dependent on the probability of clients requesting a transition from playback to fast-forward, as well as the expected duration for which a client will remain in the fast-forward mode [5].

In schemes that skip frames, fast-forward at n-times the normal playback rate is achieved by transmitting every n^{th} frame to the client site. Although conceptually elegant, such frame skipping schemes may not be directly applicable for video streams that are encoded using compression techniques that exploit temporal redundancy between successive frames (e.g., MPEG compression standard [6]). This is because, such compression techniques create inter-frame

dependencies, which may prevent every n^{th} frame selected by the above method to be independently decoded (see Section 2 for a detailed exposition of this problem for the MPEG compression standard).

To avoid this problem, Chen et al. [3] have recently proposed a fast-forward scheme in which: (1) video streams are stored on disks in terms of *segments*, that consist of a group of video frames which have inter-frame dependencies; and (2) fast-forward at n-times the normal playback is achieved by accessing and displaying only one out of every n successive segments. Clearly, 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 segments may result in noticeable discontinuities in the sequence of frames being displayed, and hence, may not be acceptable.

To emulate the corresponding VCR operation, a server may 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, one can ensure that accessing such a specialized stream does not impose any additional overhead on disk or network bandwidth. However, independently maintaining such a fast-forward stream may incur substantial storage space overhead. To minimize the storage space overhead, the MPEG standard has proposed the creation of a video stream containing D frames, which only contain the DC coefficients of the transform blocks [6]. However, this yields a video stream with very poor resolution, which is not acceptable for most applications.

In summary, techniques for supporting fast-forward operation by skipping frames are likely to impose lower load (as compared to methods for displaying frames at a rate higher than normal) on the disk and network subsystems. However, to be practical, such techniques must: (1) minimize the storage space overhead of maintaining information pertinent to fast-forward operations, (2) minimize the increase in the bit rate requirement during fast-forward (and hence, minimize the effect on the disk and network subsystems), and (3) provide acceptable video quality during fast-forward. The design and evaluation of an efficient scheme for fast-forward of video streams that meets the above requirements is the subject matter of this paper. Due to the similarity in the semantics and performance requirements of the fast-forward and rewind operations, the techniques for fast-forward presented in this paper can be easily extended to rewind.

1.3 Research Contributions of This Paper

In this paper, we present an encoding technique and a placement algorithm which together efficiently support fast-forward of video streams from video servers. Our encoding technique combines scalability in the temporal and chroma dimensions supported by most compression algorithms to derive a video stream that can efficiently support the normal playback and fast-forward of video streams from a disk-array-based storage server. Specifically, it first sub-samples the parent video stream in the temporal dimension (to create a *base* and an *enhancement* sub-stream), and then partitions each frame in the base sub-stream into a *low-resolution* and a *residual* component. Whereas frames contained in both the low-resolution and residual components of the base sub-stream as well as the enhancement sub-stream are integrated during normal playback, fast-forward operation is 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 rates. Similarly, by controlling the base sub-stream partitioning process, the encoder can ensure that: (1) the bit rate yielded by utilizing the low-resolution base sub-stream for fast-forward is not significantly higher than the parent video stream, and (2) the resultant low-resolution base sub-stream provides acceptable video quality for fast-forward operations. We demonstrate the efficacy of our methodology for the MPEG compression algorithm.

The placement algorithm, on the other hand, interleaves the storage of the enhancement sub-stream as well as the low-resolution and residual components of the base sub-stream on disk-arrays and ensures that: (1) each sub-stream as well as its components are independently accessible, and (2) no additional overhead (i.e., seek and rotational latency) is incurred while accessing these sub-streams during normal playback. For this placement algorithm, we present an analytical model for predicting the effects of making a transition from playback to fast-forward on the load and performance of the disk-array.

We have implemented the encoding technique and the placement algorithm, and have utilized it for extensive trace-driven simulations. Our results demonstrate that: (1) by exploiting the characteristics of video streams and human perceptual tolerances, and (2) by integrating encoding techniques with policies for block placement on disk, the overhead of interactive scan operations (e.g., fast-forward and rewind) can be substantially reduced.

The rest of the paper is organized as follows: In Section 2, we present our techniques for supporting fast-forward operation in a video server. A model for evaluating the effects of a fast-forward operation on the load on a disk array

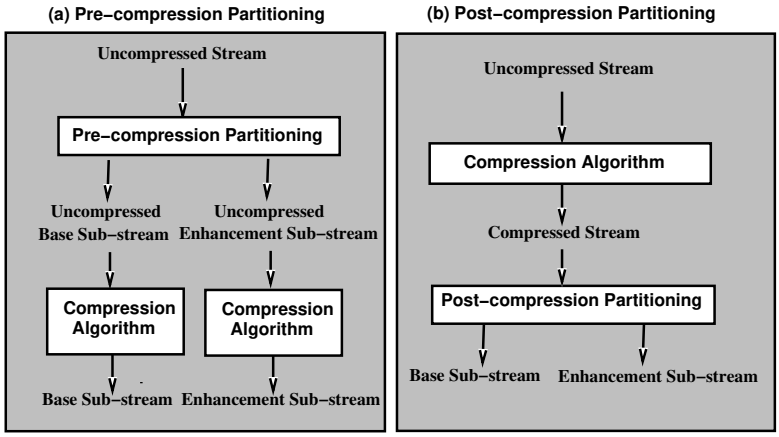

Figure 1 : Temporal Partitioning Techniques

is presented in Section 3. In Section 4, we validate our model through extensive simulations and analyze our results. Finally, Section 5 summarizes our results.

2 Efficient Fast-Forward for Compressed Video Streams

2.1 General Methodology

Consider a multimedia server that supports the fast-forward of video streams by skipping frames. To achieve fast-forward at n-times the normal playback rate, the server must transmit every n^{th} frame to the client site. To do so, however, the server must maintain additional data structures that permit individual frames to be indexed. Moreover, if we assume that information is accessed from disk in terms of fixed-size blocks, then due to the variable frame sizes, in the worst case, the server will be required to access all the frames from the disk prior to the selectively transmitting every n^{th} frame to the client sites, thereby incurring an n fold increase in the load. To address this limitation, the server can partition each video stream into two sub-streams in the temporal dimensions, 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 at n-times the normal rate will require the server to access only the base sub-stream. During normal playback, however, the server will need to access both sub-streams of a video stream. Clearly, such temporal partitioning can be accomplished either prior-to, or after compression (referred to as *pre-compression* and *post-compression* partitioning, respectively) (See Figure 1). The applicability between the two approaches, however, is dependent on the compression algorithm. Compression algorithms that exploit only the spatial redundancies within a frame (e.g., JPEG [10]) independently encode successive frames of a video stream. In such a scenario, both pre-compression and post-compression partitioning are logically equivalent and do not affect compression efficiency.

Inter-frame compression algorithms (e.g., MPEG [6]), on the other hand, exploit both the spatial and temporal redundancies present in a video stream, and hence, create dependencies between encoded frames. The extent of compression achieved by these algorithms is critically dependent on the correlation between successive frames of a video stream. 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 techniques do not suffer from the above drawbacks, the dependencies between frames introduced by the compression algorithm may complicate the partitioning process. This is because, for temporal partitioning to be effective, frames constituting the base sub-stream should be independently decodable (i.e., without decoding the corresponding enhancement sub-stream). Consequently, the exact algorithm for post-compression partitioning depends upon the idiosyncrasies of the inter-frame compression technique as well as the desired rate of fast-forward.

Regardless of the partitioning technique used, the fast-forward operation usually results in an increase in the bit rate requirement of the video stream. However, since human perception is tolerant to a slight degradation in the video quality during fast-forward, the bit rate requirement 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 [4]. 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.

In summary, in order to efficiently support the fast-forward operation, an inter-frame compression algorithm must use a combination of temporal and chroma scalability techniques. The precise algorithm for deriving these streams, however, are dependent upon the desired speed of fast-forward and the flexibility provided by the compression algorithm. In what follows, we show how these techniques may be employed for the MPEG compression algorithm.

2.2 Supporting Fast-Forward for MPEG Streams

2.2.1 The MPEG Compression Standard

The MPEG compression algorithm exploits the large temporal and spatial redundancies present within an image to achieve a high degree of compression [6]. MPEG supports four kinds of frames : (1) I frames (intra-coded without any reference to other frames), (2) P frames (predictively coded using temporal prediction from a previous I or P frame, (3) B frames (bidirectionally interpolated from both the previous and the following I and/or P frame, and (4) D frame (intra-coded with only the DC coefficient of the DCT retained in each 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.

A raw image consists of a luminance component and two chrominance components. MPEG requires the chrominance components to be down-sampled by factor of two in the horizontal and vertical dimensions. MPEG partitions each image into areas called macro blocks. Each macro block consists of six 8x8 pixel blocks, four luminance blocks and two chrominance blocks. Macro blocks belonging to B and P frames are temporally predicted from the corresponding reference frame(s), and the error between the actual and predicted values is computed. The prediction process also produces up to two motion vectors for each macro block (depending upon whether the macro block belongs to a B frame or a P frame), which denote the positions of the predicted macro blocks in the reference frames. Each 8x8 pixel block in the error macro block is transformed into the frequency domain using the discrete cosine transform (DCT). The DCT uncorrelates each pixel block into an 8x8 array of coefficients such that the most amount of energy is packed in the fewest number of low frequency coefficients. Whereas the lowest frequency coefficient (referred to as the DC coefficient) captures the average brightness of 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 independent of the AC coefficients. Within each block, the AC coefficients 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 are difference and entropy encoded in a lossless manner. The 8x8 pixel blocks of the macro blocks of an I frame are directly transformed using the DCT, quantized, and run-length and entropy encoded. Figure 2 depicts the main steps involved in the MPEG compression algorithm. The MPEG-2 standard extends the MPEG compression standard by permitting the use of scalable compression techniques while encoding video streams [4]. Specifically, MPEG-2 provides support for scalability in the spatial, temporal, and chroma dimensions.

In general, due to the inter-frame dependencies introduced by MPEG-like compression algorithms (which exploit temporal redundancies), partitioning of a video stream into the base and the enhancement sub-streams must be performed prior to compression, thereby substantially degrading compression efficiency. An important feature of the MPEG compression algorithm is that the relative frequency of occurrence of I, P, and B frames (i.e the encoding pattern) can be controlled by the application. In what follows, we illustrate how this flexibility can be exploited to derive the base and the enhancement sub-streams without adversely affecting the compression efficiency.

2.2.2 Deriving the Base and the Enhancement Sub-streams

One of the key requirements of post-compression temporal partitioning is to generate a base sub-stream which has no dependencies on any of the frames contained in the enhancement sub-stream. Consequently, the base sub-stream can not contain a P or a B frame whose reference frame belongs to the enhancement sub-stream. To achieve this objective, an encoder must ensure that all the reference frames (i.e., I and P frames) belong to the base sub-stream. Recall that, to support fast-forward at n-times the playback rate, the base sub-stream is constructed by selecting every n^{th} frame

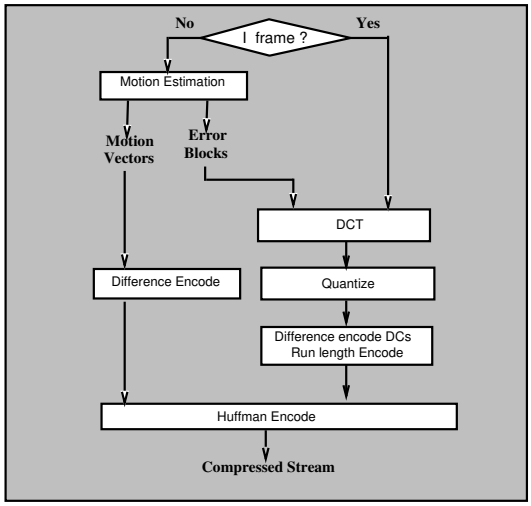

Figure 2 : The MPEG Compression Algorithm

from the original video stream. Consequently, to ensure that the above condition holds, the encoding pattern of the stream must be constrained.

The encoding pattern of an MPEG stream is specified by two parameters: (1) N, the distance between successive reference (i.e., I or P) frames, and (2) M, the distance between successive I frames. To guarantee that frames in the base sub-stream do not depend upon frames in the enhancement sub-stream, the encoder must choose an encoding pattern such that

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

The resultant encoding pattern (i.e., $IB^{N-1}PB^{N-1}P...$) contains $(N-1)$ B frames between successive reference frames, and $(m-1)$ P frames between successive I frames. Consequently, after temporal partitioning, the base sub-stream contains all the I and P frames, and every n^{th} B frame (if any) between successive reference frames. The enhancement sub-stream, on the other hand, contains all the remaining B frames. Figure 3(a) illustrates the main steps in the compression algorithm. As is evident from the figure, temporal partitioning occurs after the DCT and motion-estimation stages. Thus, by appropriately choosing an encoding pattern, the original video stream can be temporally partitioned *after* the spatial and temporal redundancies have been exploited. Consequently, there is no degradation in compression efficiency of the resultant compressed stream.

Observe that the relative frequency of I and P frames in the base sub-stream is higher than that in the original stream. Since frames from only the base sub-stream are transmitted during fast-forward, the bit rate requirement of the resultant stream is higher than that during normal playback. However, human perception is tolerant to a slight degradation in the video quality during fast-forward, and hence, the bit rate requirement 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. Such partitioning in the chroma dimension can be achieved by data-partitioning the sub-stream such that a certain number of low frequency DCT coefficients are assigned to the low-resolution component, and all the remaining coefficients are assigned to the residual component. The data partitioning technique for a 3x3 array of DCT coefficients is illustrated in Figure 4. Thus, by appropriately choosing the number of DCT coefficients in the low-resolution component of the base sub-stream, the data partitioning technique ensures an acceptable video quality without any increase in the bit rate requirement of the media stream during fast-forward.

During normal playback, frames from the enhancement sub-stream as well as both components of the base sub-stream are retrieved and merged prior to transmission to clients. However, during a fast-forward operation: (1) the server transmits frames from only the low-resolution component of the base sub-stream (thereby, effectively transmitting every n^{th} frame from the original stream), and (2) the playback rate remains unchanged. This results in an n-fold increase in *effective* playback rate of the original stream. The main steps in the decompression algorithm are illustrated in Figure 3(b).

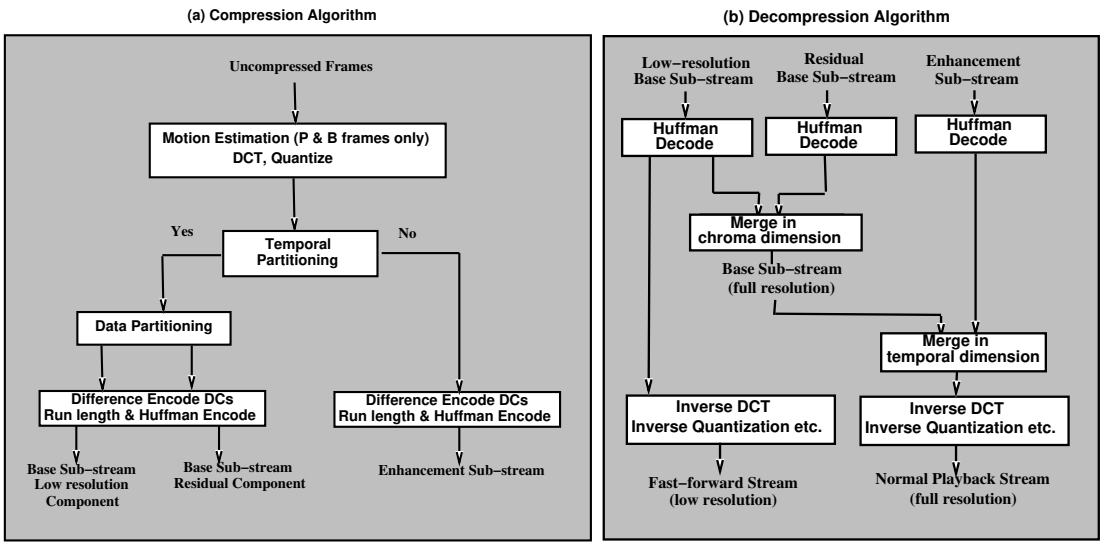

Figure 3 : The Compression and Decompression Algorithms

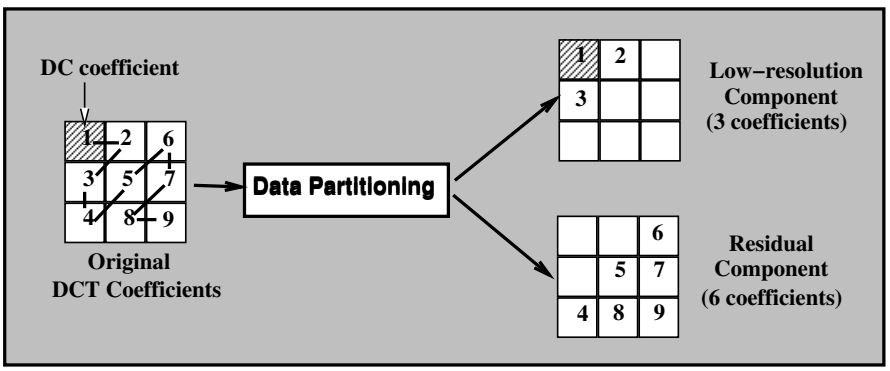

Figure 4 : The Data Partitioning Technique

To illustrate the compression algorithm, consider an application which requires support for fast-forward at twice the normal playback rate. In such a scenario, the encoder can choose $N = 2$ and $M = 6$ (i.e., $IBPBPBP...$) as the encoding pattern for the video stream, and assign the I and P frames to the base sub-stream and the B frames to the enhancement sub-stream. Alternatively, the encoder can choose $N = 4$ and $M = 12$ (i.e., $IBBBPBBBP...$) to define the encoding pattern for the video stream. 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 all the B_1 and B_3 frames. In either case, the base sub-stream is subjected to partitioning in the chroma dimension to produce the low-resolution and the residual components. On the other hand, if the needs of an application do not permit any flexibility in the choice of the encoding pattern, then the encoder must resort to pre-compression temporal partitioning of the video stream, and encode each sub-stream independently. However, this leads to a degradation in compression efficiency due to the reduction in correlation between successive frames within each sub-stream. The compression and decompression algorithms may be implemented by either using a MPEG-2 compliant codec (which supports scalable compression), or by suitably extending a MPEG-1 codec to incorporate partitioning in the temporal and chroma dimensions.

2.3 Placement of Base and Enhancement Streams on Disk-Arrays

The disk-array placement algorithm interleaves the storage of the enhancement sub-stream as well as the low-resolution and residual components of the base sub-stream on disk-arrays and ensures that: (1) each sub-stream as well as its components are independently accessible, and (2) no additional overhead (i.e., seek and rotational latency) is incurred while accessing these components during normal playback. A multimedia server may employ a fixed-size block placement policy, or a variable-size block placement policy to store compressed video streams on a disk-array [15, 2, 9]. In the fixed-size block placement policy, each media block consists of a fixed number of storage units (i.e., bytes). Since, each media stream stored on the array is compressed using a variable bit rate (VBR) compression algorithm, each block contains variable number of frames. On the other hand, in the variable-size block placement policy, each media block consists of a fixed number of media units (i.e., frames) , and hence, the size of the media block varies due to VBR compression.

The main disadvantage of the fixed-size block placement policy is its higher buffer space requirements as compared to the variable-size block placement policy. Assuming that a multimedia server services clients by proceeding in terms of periodic rounds, the set of fixed-size media blocks accessed during a round generally contain a larger amount of data than what is consumed, especially for blocks belonging to the low-resolution and residual components of the base sub-stream. Hence, the server must provide sufficient buffer space to maintain such additional data across round boundaries. Whereas buffer space requirements can be decreased by reducing the media block size, this leads to an increase in the seek latencies incurred at the disks, thereby decreasing the throughput of the disk-array. The variable-size block placement policy, on the other hand, retrieves exactly as much data as is consumed in each round, and hence, does not suffer from the above drawback. Consequently, in this paper we assume that the multimedia server uses a variable-size block placement policy. However, the techniques presented in this paper can be easily extended for a fixed-size block placement policy.

Consider a multimedia server that interleaves the storage of video streams on a disk array in terms of variable-size blocks, each containing a fixed number of frames. Since the multimedia server services multiple clients simultaneously by proceeding in terms of periodic rounds, during each round, the server will access a fixed number of frames for each client. Thus, if $f_1, f_2, ..., f_N$, and $r_1, r_2, ..., r_N$ denote the the number of frames accessed during a round and the playback rate for each client, respectively, then to ensure continuity of playback, the duration of a round must be bounded by

$$\tau = \min_i(\frac{f_i}{r_i})$$

In such a scenario, to minimize the seek time and the rotational latency incurred during a round, each block should contain f_i frames. Moreover, blocks from each sub-stream that are retrieved in a round must be stored in a contiguous manner on a disk. This ensures that there is no additional overhead (i.e seek and rotational latency) incurred during normal playback. Whereas blocks from the enhancement sub-stream as well as the two components of the base sub-stream are retrieved during normal playback, during fast-forward, however, blocks from only the low-resolution component of the base sub-stream are retrieved and transmitted to clients (See Figure 5). This placement policy enables the server to access large chunks of data each time a disk-head is repositioned, thereby improving the throughput of the disks.

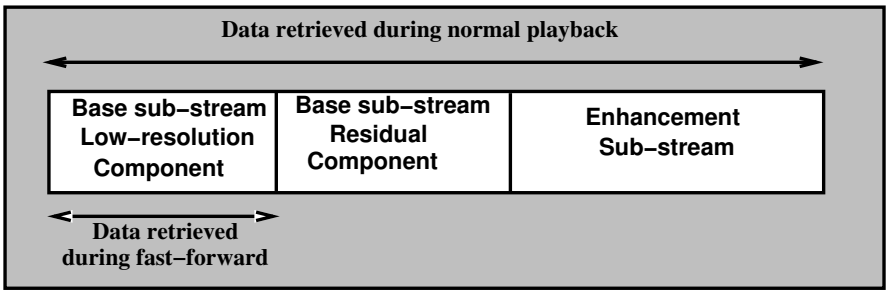

Figure 5 : Contiguous Placement of Sub-stream Blocks

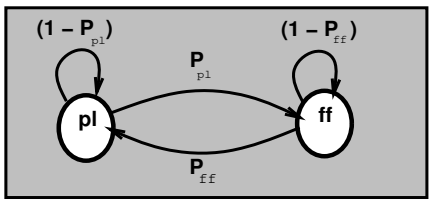

Figure 6 : The Markov model for the behavior of a client

The most appealing feature of this placement policy is that, regardless of the playback rate requirements, the retrieval of each individual video stream from disk proceeds in *lock-step*. That is, each client accesses exactly one disk during each round in the playback mode, and that consecutive disks in the array will be accessed by the same set of clients during successive rounds. However, during fast-forward at *n*-times the normal playback rate, each client accesses exactly *n* disks in each round, and consecutive sets of *n* disks will be accessed by the same set of clients in successive rounds. Thus, depending upon whether a client is in the playback mode or the fast forward mode, the server accesses either a single block or n blocks per client in each round. Since the data partitioning technique ensures that the bit rate of the stream during fast forward is nearly the same as that during playback, the aggregate size of n fast forward blocks is same as that of a normal playback block. In what follows, we present an analytical model which evaluates the impact of accessing n smaller blocks in the fast forward mode, instead of a single large block in the playback mode, on the performance of the server. The derivation of the analytical model is similar to that of the model presented in [15] for determining optimal media block sizes in case of a fixed-size block placement policy.

3 Determining the overhead of the fast forward operation

3.1 Modeling Client Behavior

A client in the playback mode can switch to the fast forward mode at any random instant, and vice versa. However, since a multimedia server services clients by proceeding in rounds, we assume that such transitions occur only at round boundaries. This behavior can be modeled using a two state discrete-parameter Markov chain [12] (see Figure 6). In figure 6, P_{pl} denotes the probability of switching from playback to fast-forward, and P_{ff} denotes the probability of switching from fast-forward to playback mode. Due to the memoryless property of a Markov chain, the times spent by a client in each mode are geometrically distributed.

3.2 Analyzing the Impact of Playback to Fast-Forward Transitions

Consider a multimedia server that is servicing 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 during each round from streams $S_1, S_2, ..., S_N$, respectively. Each stream S_i, encoded using the algorithm presented in Section 2, consists of several sub-streams (say $S_i^1, S_i^2, ..., S_i^k$). During normal playback, the server retrieves $f_i^1, f_i^2, ..., f_i^k$ frames of sub-stream $S_i^1, S_i^2, ..., S_i^k$ respectively in each round, which are then merged to obtain f_i frames of the original stream S_i. However, during the fast forward operation, the server retrieves f_i frames of only the low resolution component of the base sub-stream (say S_i^1) in each round.

Let each sub-stream S_i^j be partitioned into variable size blocks containing f_i^j frames each. As described in Section 2.3, blocks from each sub-stream that are retrieved in a round are stored in a contiguous manner on disk to optimize on the seek latency incurred during normal playback. Let us assume that the disk-array consists of D asynchronous disks. For client i, let P_{pl}^i denote the probability of switching from playback to fast-forward , and let P_{ff}^i denote the probability of switching from fast-forward to playback . If $P_{pl,ss}^i$ and $P_{ff,ss}^i$ denote the steady state Markov probability of client i being in playback and fast-forward respectively, then $P_{pl,ss} + P_{ff,ss} = 1$.

Given that media blocks of each video stream have been stored on the array using the round robin placement algorithm, depending upon whether client i is in the playback mode or the fast forward mode, it accesses either a single block or n blocks from the array in each round. 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 $n \leq D$,

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

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. Assuming transitions from fast-forward to playback and vice-versa occur at random instants in time, a media block is equally likely to be accessed from any of the disks in the array. Consequently,

$$P(\mathcal{X}_i^j = 1) = p_i = P_{pl,ss}^i \cdot \frac{1}{D} + P_{ff,ss}^i \cdot \frac{n}{D}$$

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

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

Since clients can switch between the playback and the fast forward modes at random instants in time, even if multiple clients begin accessing the same media stream simultaneously, the set of media blocks accessed by the streams become independent of each other within a small number of rounds. That is, $\mathcal{X}_i^j$'s are independent. Hence, we get:

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

where $Z(\mathcal{B}^j)$ and $Z(\mathcal{X}_i^j)$ are the z-transforms of the random variables $\mathcal{B}^j$ and $\mathcal{X}_i^j$ respectively [12]. Since $Z(\mathcal{X}_i^j) = (1-p_i) + p_i z$, we get $Z(\mathcal{B}^j) = \prod_{i=1}^{N}((1-p_i) + p_i z)$. 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:

$$P(\mathcal{B}^j = x) = \begin{pmatrix} N \\ x \end{pmatrix} p^x (1-p)^{N-x}, \qquad 0 \leq x \leq N$$

Similarly, the distribution function $F_{\mathcal{B}^j}(x) = P(\mathcal{B}^j \leq x)$ is given by:

$$F_{\mathcal{B}^j}(x) = \sum_{k=0}^{x} \begin{pmatrix} N \\ k \end{pmatrix} p^k (1-p)^{N-k}, \; 0 \leq x \leq N$$

The expected number of blocks accessed from the most heavily loaded disk can be determined using the distribution function of $\mathcal{B}_{\max} = \max(\mathcal{B}^j)$. Since the distribution functions of $\mathcal{B}^j$'s yielded by the round-robin allocation algorithm are dependent, we use the following result derived in [15] :

$$\max\{0, D * F_{\mathcal{B}^j}(x) - (D-1)\} \leq F_{\mathcal{B}_{\max}}(x) \leq F_{\mathcal{B}^j}(x)$$

Figure 7 illustrates the variation of the lower and the upper bound of $F_{\mathcal{B}_{\max}}(x)$ with x. As is evident from Figure 7, the lower bound of $F_{\mathcal{B}_{\max}}(x)$ yields an upper bound on $\mathcal{B}_{\max}$. Since the objective is to determine the load on the most heavily loaded disk, in the rest of the paper we will use the lower bound of $F_{\mathcal{B}_{\max}}(x)$ to approximate $F_{\mathcal{B}_{\max}}(x)$.

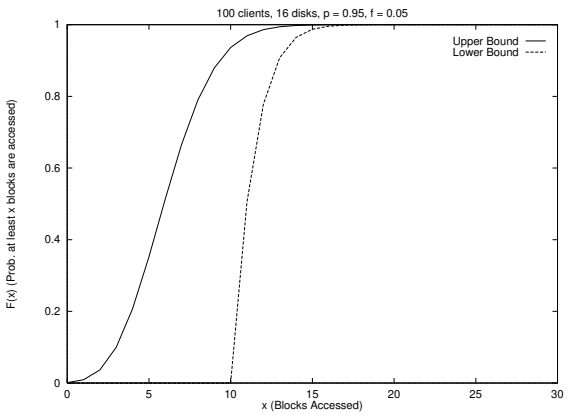

Figure 7 : Variation of $F_{\mathcal{B}_{\max}}(x)$ with x

Since $\mathcal{B}^j$ denotes the total number of media blocks accessed from disk j, the expected number of blocks accessed from the most heavily loaded disk is $\mathcal{B}^{exp}_{\max} = E(\mathcal{B}_{\max})$. Assuming that the server employs SCAN disk scheduling algorithm, the service time for $\mathcal{B}^{exp}_{\max}$ blocks can be modeled as:

$$\tau(\mathcal{B}^{exp}_{\max}) = \mathcal{B}^{exp}_{\max} * (t^{exp}_{seek} + t^{exp}_{rot} + t^{exp}_{xfer}) \tag{1}$$

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

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

and a, b, and c are constants (determined using physical characteristics of a disk) [8]. The rotational latency t^{exp}_{rot}, is defined to be half of the maximum rotational latency. The transfer time t^{exp}_{xfr}, is computed as a weighted average of the expected transfer times of a playback block and a fast-forward block, where the weights are the expected number of blocks retrieved during playback and fast-forward, respectively. Hence, if t^{pl}_{xfr} and t^{ff}_{xfr} denote the expected transfer time of a playback block and a fast-forward block respectively (obtained from trace data), then

$$t^{exp}_{xfr} = \frac{P_{pl,ss} \cdot t^{pl}_{xfr} + P_{ff,ss} \cdot n \cdot t^{ff}_{xfr}}{P_{pl,ss} + P_{ff,ss} \cdot n}$$

Thus, given the server configuration (i.e., disk array characteristics, number of clients, and their data rate requirements) and the client Markov probabilities, Equation (1) derives the expected service time for the most heavily loaded disk. The service time τ computed by the model is for a particular value of the probability pair (P_{pl}, P_{ff}) of the Markov chain. By varying the steady-state Markov probability of being in the fast-forward mode, we can compute the variation in the service time of the most heavily loaded disk as predicted by the model. The resultant change in the service time values, obtained by increasing the probability of a client in fast-forward, indicates the overhead of supporting the fast-forward operation.

4 Experimental Evaluation

In this section, we demonstrate the viability of our fast-forward scheme using a prototype encoder built at the Distributed Multimedia Computing Laboratory, UT Austin. We also validate the analytical model through extensive trace-driven simulations. The simulation environment consists of an asynchronous array of 16 disks. The characteristics of each disk in the array are shown in Table 1. Each video stream is assumed to be encoded using the algorithm presented in Section 2 with $N = 4$ and $M = 12$ as the encoding pattern, and stored on disks in a round-robin manner using a variable block placement policy. The normal playback rate of each video stream is assumed to be 30 frames/second. For the purpose of simulations, we assume a fast forward speed of twice the normal playback rate.

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

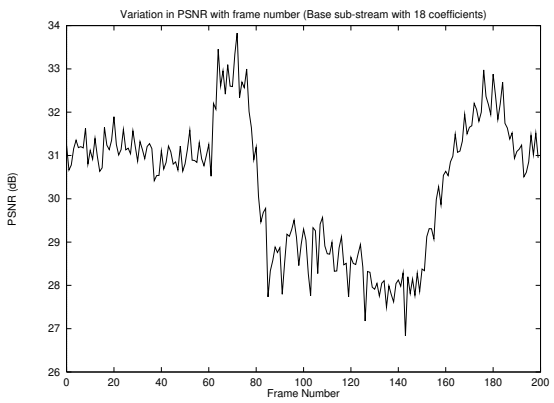

Figure 8 : Variation of PSNR with frame number

The trace data for simulations was generated by digitizing several video clips, the largest one of which was a ten minute sequence of a Frasier episode [1] which yielded 18,000 frames. These frames were then encoded using our prototype encoder, and the trace data was generated for the resulting compressed stream.

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

The output of the MPEG compression algorithm is a variable bit rate (VBR) stream. Due to the variable nature of the compression ratios of frames yielded by MPEG, the average number of DCT coefficients required to pack all the information contained in an 8x8 pixel block varies from frame to frame. The data partitioning technique stores a fixed number of DCT coefficients in the low-resolution component of the base sub-stream. Consequently, the video quality obtained by decoding only the low-resolution component of the base sub-stream varies from frame to frame.

To quantitatively measure the quality of an image, we use the *Peak Signal to Noise Ratio (PSNR)* as our metric. For an AxB image of resolution r bits/pixel, let $p(x,y)$ and $p'(x,y)$ denote the pixel values at location (x,y) in the original and the uncompressed images respectively. We define the PSNR of the image by the equation

$$PSNR = 10.log_{10}(\frac{(2^r-1)^2}{\sigma^2}) \text{ dB}$$
$$\text{where } \sigma = \sum_{x=1}^{A}\sum_{y=1}^{B}(p(x,y) - p'(x,y))^2$$

Figure 8 shows the variation in the video quality obtained by decoding frames from only the low-resolution component of the base sub-stream, where the low-resolution component was encoded using the lower 18 DCT

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

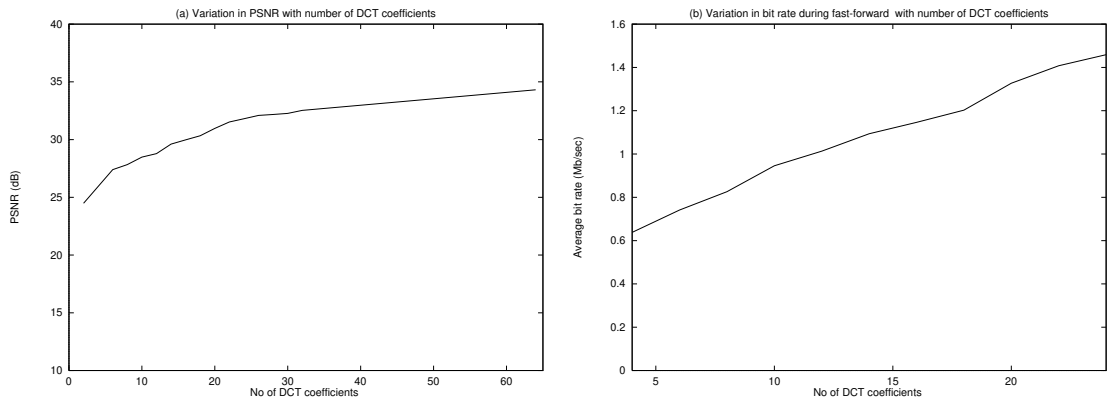

Figure 9 : Variation of PSNR and fast-forward bit rates with number of DCT coefficients.

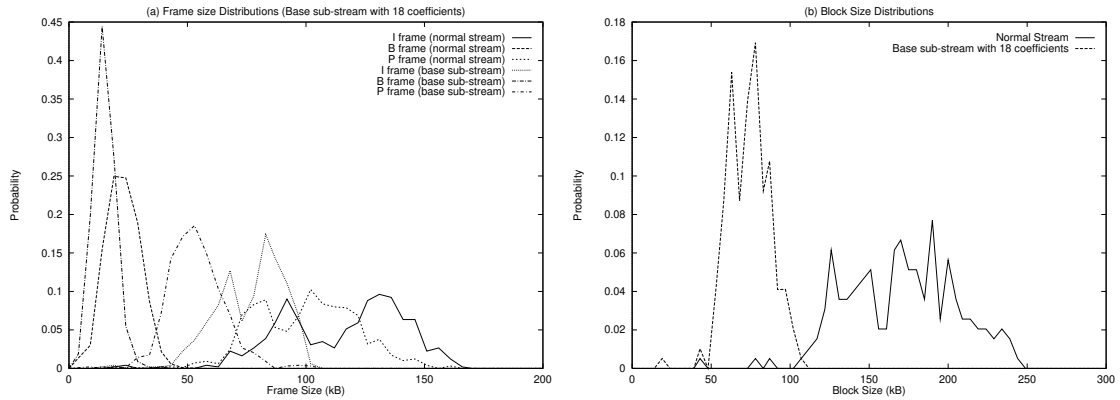

Figure 10 : Comparison of block and frame size distributions of the base sub-stream with the normal stream

coefficients. As is evident from the figure, even though the video quality of the encoded frames shows variations over relatively small sequences, the average video quality is acceptable during fast-forward.

Figure 9(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 of the base sub-stream. The figure shows that an acceptable video quality is obtained when the low-resolution component includes at least 18 to 20 lower DCT coefficients. On the other hand, Figure 9(b) shows the variation in the bit rate of the media stream during fast-forward obtained by varying the number of DCT coefficients contained in the low-resolution component. The figure demonstrates that increasing the number of DCT coefficients contained in the low-resolution component leads to an increase in the bit rate of the stream during fast-forward. Since the bit rate of the Frasier sequence during normal playback is about 1.35 Mbits/s, the low-resolution component must contain no more than 20 DCT coefficients to ensure that the bit rate of the media stream remains unchanged during fast-forward. Based on these tradeoffs, we conclude that the low-resolution component of the base sub-stream should include 18-20 DCT coefficients for acceptable video quality with negligible increase in the bit rate requirement during fast-forward.

Figure 10(a) shows the efficacy of the data partitioning technique in reducing the stream bit rate during fast-forward. The figure compares the frame size distributions of the *I, P*, and *B* frames of a data-partitioned stream to those belonging to a normal MPEG stream. Clearly, the data partitioning technique reduces the average frame size significantly, thereby reducing the average stream bit rate during fast-forward. Figure 10(b) compares the block size distributions of blocks retrieved during fast-forward to those retrieved during normal playback. As is evident from the figure, the average size of a block retrieved during fast-forward is about half of that retrieved during normal playback. Since the server retrieves two blocks per client in each round during fast-forward, the bit rate of the media stream remains unchanged.

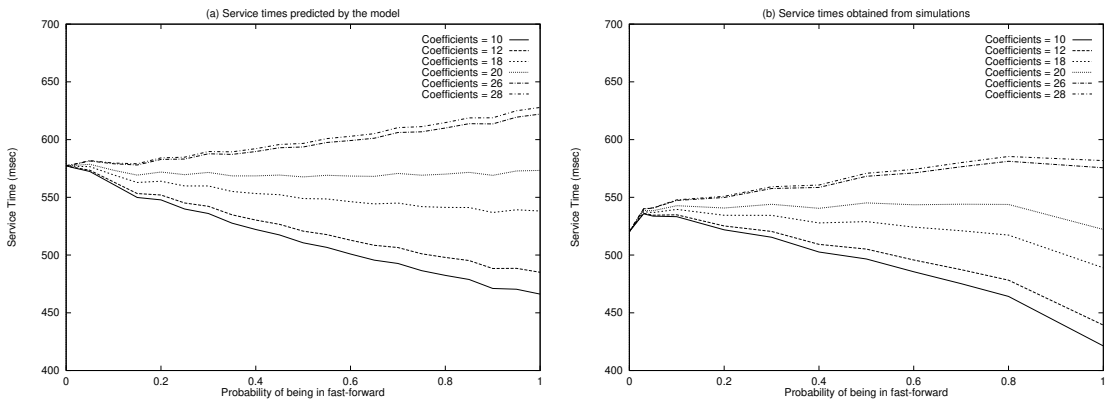

Figure 11 : Variation of service times with the probability of being in the fast-forward mode.

4.2 Validation of the Model

Figure 11(a) shows the variation in the expected value of the service time of the most heavily loaded disk obtained by varying the steady state Markov probability of a client in fast-forward as predicted by the analytical model. Figure 11(b) shows these values obtained from trace-driven simulations. Recall that, in Section 3, the probability distribution function for $\mathcal{B}_{\max}$ was chosen such that the model yields an upper bound on the expected value of the service time of the most heavily loaded disk. Consequently, the service times predicted by the model are around 5-8% larger than those obtained from simulations. Figure 11 shows that when the low-resolution component of base sub-stream contains 18 to 20 DCT coefficients, the expected value of the service time of the most heavily loaded disk is more or less independent of the probability of being in fast-forward. In other words, the media streams do not impose any additional overheads on the server during fast-forward. When the low-resolution component is encoded with fewer than 18 DCT coefficients, the bit rate of the stream during fast-forward is lower than that during normal playback (See Figure 9(b)). Consequently, the load imposed by such media streams during fast-forward is lower than that during normal playback. Since bit rate of video streams whose low-resolution component contains more than 20 DCT coefficients increases during fast-forward, this leads to a corresponding increase in the load on the most heavily loaded disk. However, this increase is much smaller than those assumed by most conventional schemes.

5 Concluding Remarks

In this paper, we have presented an encoding technique and a placement algorithm, which together efficiently support fast-forward of video streams. Our encoding technique combines scalability in the temporal and chroma dimensions supported by most compression algorithms to derive a video stream that can efficiently support fast-forward. By interleaving the storage of the sub-streams and their components on disk-array, the placement algorithm ensures that: (1) each sub-stream is independently accessible, and (2) no additional seek and rotational latency is incurred while accessing the sub-streams during normal playback. We have presented an analytical model for predicting the effects of making a transition from playback to fast-forward on the load and performance of the disk-array. The analytical predictions were validated by extensive trace-driven simulations.

Most conventional schemes which support fast-forward increase the bit rate requirements during fast-forward and/or increase the storage space overhead. By exploiting the characteristics of the compression algorithm and human perceptual tolerances, our scheme supports the fast-forward operation at acceptable picture quality without any increase in the bit rate and the storage space requirements.

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] Ming-Syan Chen, Dilip D. Kandlur, and Philip 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.

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

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

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

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

[8] 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.

[9] 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.

[10] William B. Pennebaker and Joan L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[11] 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.

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

[13] H. M. Vin, A. Goyal, A. Goyal, and P. Goyal. An Observation-Based Admission Control Algorithm for Multimedia Servers. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS'94), Boston*, pages 234–243, May 1994.

[14] 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.

[15] 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 (ICMCS'95)*, 1995.

[16] 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.