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 evaluating 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.

KEYWORDS

Video servers, scan operations, disk arrays

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, product announcements, customer support, video on-demand, etc.). In its simplest configuration, the architecture of such multimedia services will comprise of multimedia information 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.

Amongst all these data types, since video is the most demanding (with respect to data rate and real-time performance requirements), several research groups have investigated techniques for meeting the real-time playback requirement of video streams [1, 7, 11, 13, 15]. However, methods for efficiently supporting interactive scan operations (fast-forward and rewind) have not been adequately investigated.

Relation to Previous Work

In general, schemes that support fast-forward and rewind operations for video either display frames at a rate higher than normal playback [5], or 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 bandwidth available at the server is not sufficient to meet the increased requirements, then the client request must be delayed until the necessary resources become available. To minimize the waiting time, the server may 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 is dependent on the probability of clients requesting a transition from playback to fast-forward, as well as the duration for which a client may remain in the fast-forward mode [5].

In schemes that skip frames, on the other hand, fast-forward at n-times the normal playback rate is achieved by displaying every n^{th} frame at the normal playback rate. 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 to be independently decoded.

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*, consisting of group of frames that can be independently decoded; and

(2) fast-forward at n-times the normal playback is achieved by accessing and displaying 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 require any additional 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 impose a 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 during fast-forward (and hence, minimize the overhead on the disk and network subsystems), and (3) provide acceptable video quality. The design and evaluation of a scheme that meets the above requirements is the subject matter of this paper. Since fast-forward and rewind impose similar performance requirements, the techniques for fast-forward presented in this paper can be easily extended to rewind.

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 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. Each frame in the base sub-stream is then partitioned into a *low-resolution* and a *residual* component. Whereas frames contained in all three components 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. Furthermore, 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 both 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 time 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 them for carrying out extensive trace-driven simulations. Our results demonstrate that the overhead of interactive scan operations (e.g., fast-forward and rewind) can be substantially reduced by exploiting the characteristics of video streams and human perceptual tolerances, and by integrating encoding techniques with policies for block placement on disk,

The rest of the paper is organized as follows: First, we present our techniques for supporting fast-forward operation in a video server. Second, we present a model for evaluating the effects of a fast-forward operation on the load on a disk array. We will then evaluate our scheme through extensive simulations, and finally, summarize our results.

SUPPORTING FAST-FORWARD OPERATION

General Methodology

Consider a multimedia server that supports 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. However, if we assume that information is accessed from disk in terms of fixed-size blocks, then, in the worst case, the server will be required to access all the frames from the disk prior to selectively transmitting every n^{th} frame to the client sites, thereby incurring an n fold increase in the load on disks. To address this limitation, the server can partition each video stream into two sub-streams in the temporal dimension, 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, 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 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 1). The applicability of these approaches, however, is dependent on the compression algorithm. In intra-frame compression algorithms (e.g., JPEG [10]), since successive frames can be decoded independently, the pre- and the post-compression partitioning techniques are logically equivalent. Moreover, they do not have any adverse effects on the compression efficiency. In inter-frame compression algorithms (e.g., MPEG [6]), on the other hand, the degree of compression 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 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 constituting 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 point out in the next section, supporting fast-forward operation using the base stream generally yields 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.

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 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 for the MPEG compression algorithm.

Supporting Fast-Forward in MPEG

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 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).

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). DCT un-correlates 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 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 2 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 [4].

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. Specifically, an application can control the encoding pattern of an MPEG stream 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 the compression efficiency.

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

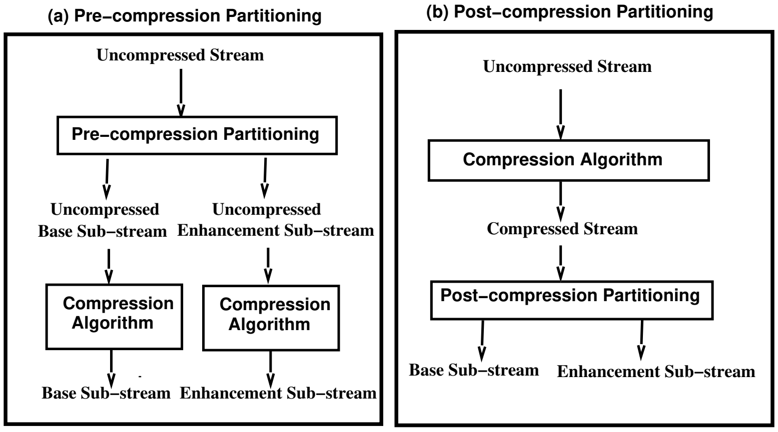

Figure 1 : Temporal partitioning techniques

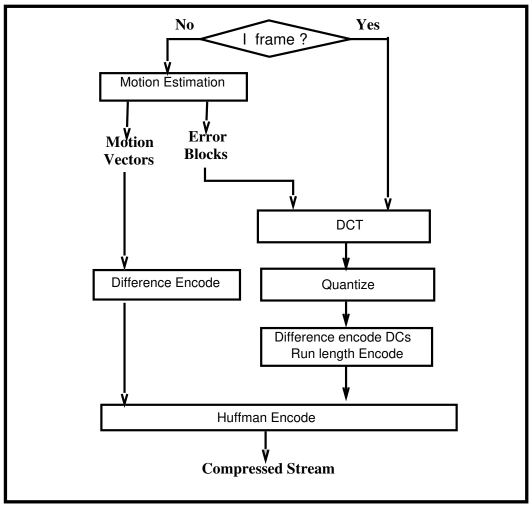

Figure 2 : The MPEG compression algorithm

dependencies on 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 (i.e., all of the I and P frames must belong to the base sub-stream). To construct a base sub-stream that meets this requirement and can support fast-forward at n-times the normal playback rate, the encoding pattern must be constrained such that:

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

Observe that 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. 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 $n = 2$), selecting $N = 2$ and $M = 6$ (i.e., the encoding pattern of $IBPBPBI...$) and then assigning alternate frames to base and enhancement sub-streams, respectively, yields a base stream containing all the I and P frames and an enhancement stream with all the B frames.

Observe that, appropriate choice of encoding pattern enables temporal partitioning to be performed *after* the spatial and temporal redundancies have been exploited by the DCT and motion-estimation stages of the MPEG compression algorithm, and thereby completely eliminates any adverse effects of temporal partitioning on compression efficiency. However, it yields a base sub-stream with a higher relative frequency of I and P frames than 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 [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. Figure 3 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.

Although only the low-resolution component of the base

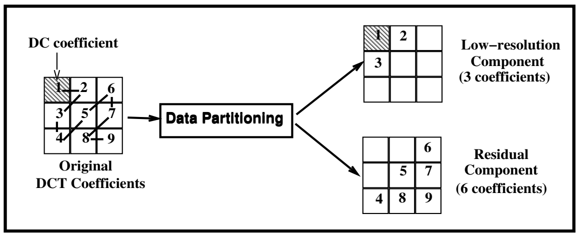

Figure 3 : The data partitioning technique

sub-stream is utilized 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 their display at the client. The main steps involved in the compression and decompression algorithm are illustrated in Figure 4. Notice that 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.

EFFICIENT PLACEMENT OF SUB-STREAMS ON DISK ARRAYS

To effectively utilize the bandwidth of a disk array (by accessing only as much data as needed and no more), the placement algorithm should interleave the storage of the enhancement sub-stream as well as both components of the base sub-stream 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. In this section, we present a placement algorithm which meets this requirement, and then present an analytical model to evaluate the impact of this placement policy on the performance of the disk array.

Placement of Base and Enhancement Sub-streams

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 [2, 9, 14]. 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 from one block to another. A detailed comparison of these policies is presented in [14]. In this section, we assume that the multimedia server employs a variable-size block placement policy. The techniques presented in this section, however, 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 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 [13]. In such a scenario, to minimize the seek time and rotational latency incurred during a round: (1) each block associated with the sub-streams should contain the number of frames needed in a round, and (2) blocks of the sub-streams must be stored contiguously on disk (See Figure 5). Moreover, to ensure that each sub-stream (as well as their components) are independently accessible, blocks for each component must be stored at disk-access unit boundaries (e.g., a sector). Finally, consecutive blocks from each stream should be stored on successive disks in a round-robin manner. Such a placement scheme yields efficient utilization of the disk bandwidth during fast-forward as well as normal playback.

As per this placement scheme, 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. However, the seek time and rotational latency overhead can be significantly different. 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.

Determining the Overhead of Fast-forward Operation

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. Let each stream S_i consist of several sub-streams (say $S_i^1, S_i^2, ..., S_i^k$). During normal playback, let us assume that 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. Blocks from the sub-streams are interleaved on a disk array consisting of D asynchronous

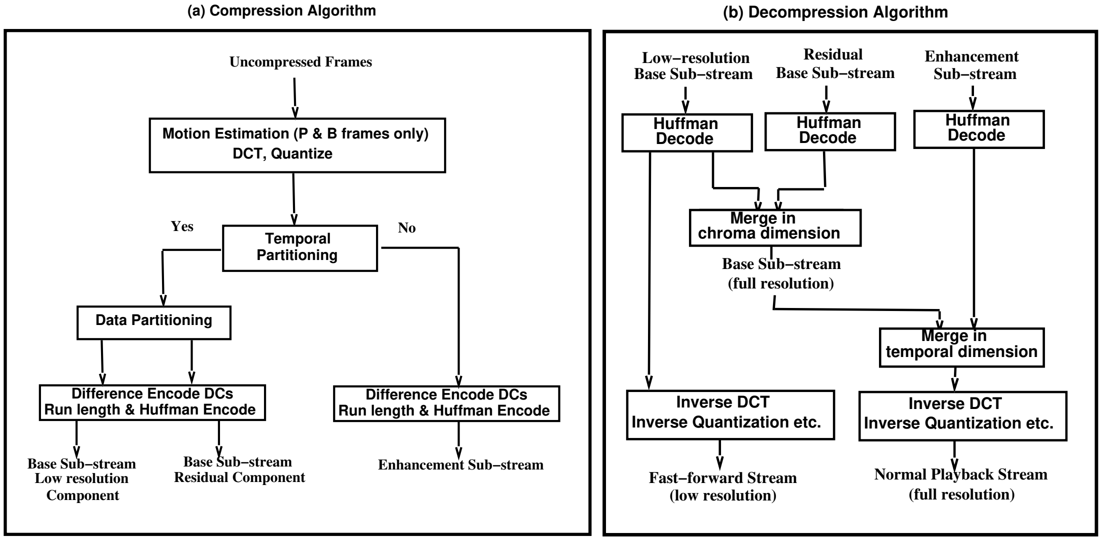

Figure 4 : The compression and decompression algorithms

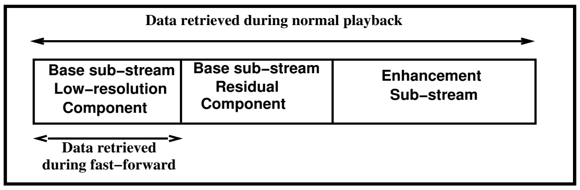

Figure 5 : Contiguous placement of sub-stream blocks

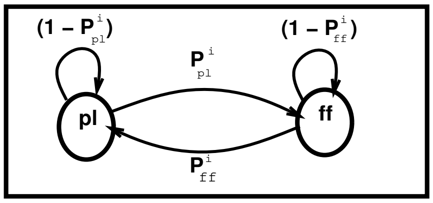

Figure 6 : The Markov model for the behavior of a client

disks using the variable-size block placement policy.

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 [12] (see Figure 6). In Figure 6, P^i_{pl} denotes the probability of switching from playback to fast-forward, and P^i_{ff} denotes the probability of switching from fast-forward to playback mode for client i. If $P^i_{pl,ss}$ and $P^i_{ff,ss}$ denote the steady state Markov probability of client i being in playback and fast-forward, respectively, then $P^i_{pl,ss} + P^i_{ff,ss} = 1$.

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 or the fast-forward mode, it accesses either a single block or n blocks from the array in each round. Let the random variable $\mathcal{X}^j_i$ denote the number of blocks accessed by client i from disk j in a round. Then assuming that $n \leq D$,

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

Clearly, $\mathcal{X}^j_i = 1$ 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 media blocks being accessed by the stream. However, since transitions from fast-forward to playback and vice-versa occur at random instants in time, after a large number of such transitions, a media block is equally likely to be accessed from any of the disks in the array. Consequently,

$$P(\mathcal{X}^j_i = 1) = p_i = P^i_{pl,ss} \cdot \frac{1}{D} + P^i_{ff,ss} \cdot \frac{n}{D}$$

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

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

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]. 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:

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

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

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

Given the distribution of $\mathcal{B}^j$, the expected number of blocks accessed from the most heavily loaded disk is given by $\mathcal{B}_{\max}^{exp} = E(\mathcal{B}_{\max})$, where $\mathcal{B}_{\max} = \max(\mathcal{B}^j)$. Assuming that the server employs SCAN disk scheduling algorithm, the service time (i.e., the total seek time, rotational latency, and data transfer time) incurred while accessing $\mathcal{B}_{\max}^{exp}$ blocks can be modeled as:

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

where t_{seek}^{exp} and t_{rot}^{exp} denote the expected seek time and rotational latency incurred while accessing a block from disk, respectively. Assuming that the $\mathcal{B}_{\max}^{exp}$ blocks are equally spaced across the $\mathcal{C}$ cylinders of a disk, we define $t_{seek}^{exp} = t_{seek}\left(\lfloor \frac{\mathcal{C}}{\mathcal{B}_{\max}^{exp}} \rfloor\right)$, 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_{rot}^{exp}, is defined to be half of the maximum rotational latency. The transfer time t_{xfr}^{exp}, on the other hand, can be 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_{xfr}^{pl} and t_{xfr}^{ff} denote the expected transfer time of a playback block and a fast-forward block, respectively (obtained from trace data), then

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

Thus, given the server configuration (i.e., disk array characteristics, number of clients, and their data rate requirements) and the steady-state 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 changes in the service time values indicate the overhead of supporting the fast-forward operation.

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

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 with $N = 4$ and $M = 12$ as the encoding pattern, and stored on disks in a round-robin manner using a variable-size 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. 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.

Determining Parameters for Data Partitioning of the Base Sub-stream

The video quality obtained by decoding the low-resolution component of the base sub-stream depends upon the number of coefficients retained in each DCT block. To quantify

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

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:

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

Figure 7(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 7(b)). For efficient fast-forward, the encoder must ensure that: (1) the bit rate during fast-forward is not significantly greater than that during normal playback, and (2) the video quality is acceptable (generally about 30dB). For the streams under consideration, this criteria was met when the low-resolution component contained 18-20 DCT coefficients of each transform block. As a consequence, the average size of a block retrieved during fast-forward is about half of that retrieved during normal playback (see Figure 8). Since the server retrieves twice the number of blocks in each round during fast-forward, the bit rate of the media stream remains unchanged.

Validation of the Model

Figure 9 shows 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. Recall that, 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 9 shows that when the low-resolution component contains around 20 DCT coefficients, 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 fast-forward operation does not impose any additional overhead on the server. When the low-resolution component is encoded with fewer than 20 DCT coefficients, the bit rate during fast-forward is lower than that during normal playback (See Figure 7(b)). Consequently, such streams impose a lower load on the server at the expense of video quality. On the other hand, increasing the number of DCT coefficients in the low-resolution component beyond 20 yields an improvement in video quality with a corresponding increase in the load on the most heavily loaded disk.

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 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 bit rate and 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] 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.

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

[5] 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.

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

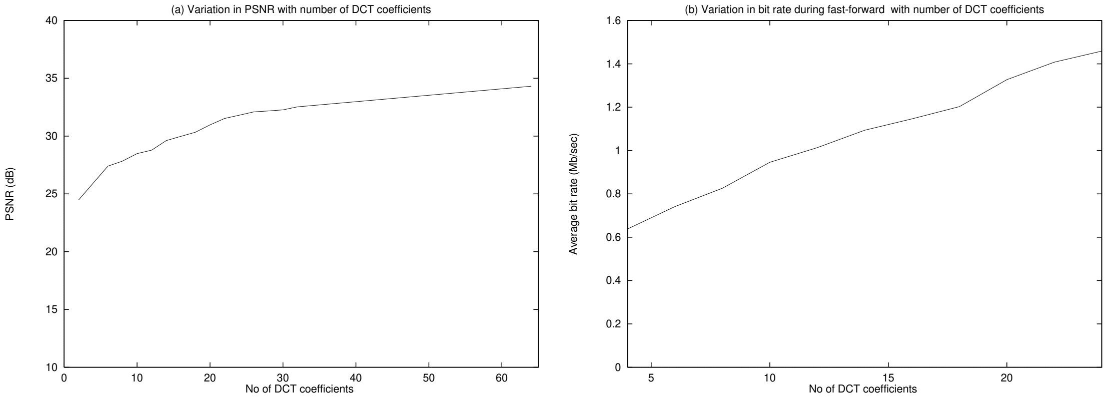

Figure 7 : Variation of PSNR and fast-forward bit rates with number of DCT coefficients.

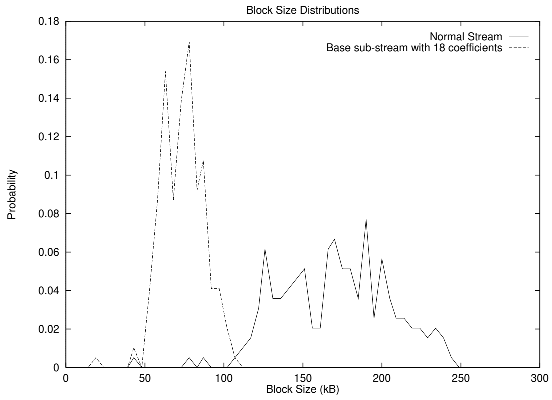

Figure 8 : Comparison of block size distribution of the base sub-stream with the normal stream

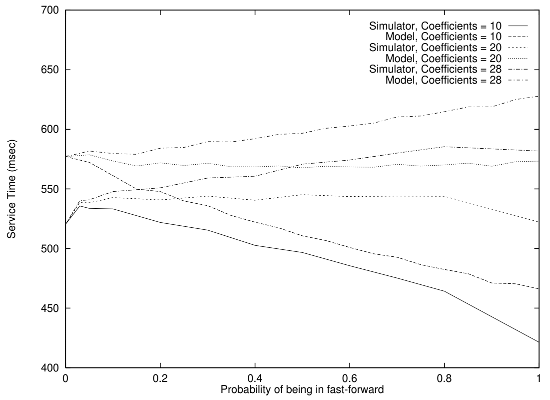

Figure 9 : Variation in service times obtained by changing the probability of fast-forward.

[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] W. B. Pennebaker and J. 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, 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.

[14] 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.

[15] 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.