Scalable Video Browsing Techniques for Intranet Video Servers*

Shahram Ghandeharizadeh, Roger Zimmermann, Jaber Al-Marri,
Weifeng Shi, Seon Ho Kim

Computer Science Department
University of Southern California
Los Angeles, California 90089

January 14, 1998

Abstract

Servers that employ scalable techniques prevent the formation of bottlenecks in order to allow the system to support a higher number of clients as function of additional hardware. They are desirable because they enable a growing organization to satisfy the increased demand imposed on its hardware platform by increasing its size (instead of replacing it with a new one which is almost always more expensive [Sto86]). This study presents the design, implementation, and evaluation of scalable techniques in support of video browsing, i.e., VCR functions such as fast-forward and fast-rewind.

1 Introduction

With the convergence of computers and consumer electronic goods, e.g., digital cameras and camcorders, digital video has become as pervasive as any other data type such as text, records, objects, etc. An Intranet video server provides for storage, delivery, and distribution of video. These servers might contain training video for the employees of an organization, archive broadcasted speeches, video conferences between several people collaborating on a project, etc. In order to be effective, these servers must support browsing techniques for video. Both fast-forward and fast-rewind (with scan) are two such techniques. To illustrate,

---

*This research was supported in part by the National Science Foundation under grants IRI-9258362 (NYI award) and ERC grant EEC-9529152, and a Hewlett-Packard unrestricted cash/equipment gift.
once a video conference is archived, one of the participants may want to browse the stored clip to recall a specific demonstration or discussion. This is realized using a fast-forward display of a clip to the point of interest. In addition, using this functionality, the user may construct marked indexes for future recall. Even with these marked indexes, browsing continues to be useful because it enables a user to study a segment carefully by moving back and forth in that segment.

This study investigates the design and implementation of scalable techniques in support of fast-forward (FF) and fast rewind (FR). By scalable, we imply that there are no inherent bottlenecks and the number of active displays that can invoke FF/FR increases as a linear function of the available bandwidth. These designs minimize: (1) the amount of memory and processing overhead incurred at both the client and the server, (2) the communications overhead between the client and the server, and (3) the delay incurred when VCR functions are invoked. We start with a taxonomy of different approaches to implement VCR functions in order to compare our work with the previous literature. Subsequently, we detail our design.

Figure 1 distinguishes two basic paradigms\(^1\) to provide VCR functions from a video server: either (1)
a single, normal-speed movie is processed accordingly in real-time during playback time\(^2\) (this paradigm is termed *online* processing), or (2) a clip is pre-processed by the content provider with separate files for FF and FR viewing (termed *offline* processing). The tradeoff between these two paradigms is I/O processing versus storage space. While online techniques require extra bandwidth (disk, network), offline techniques require additional disk storage. Each method has other advantages and disadvantages that might make it more appropriate for a specific application. We describe each method in turn.

## 2 Related Work

- **Online Processing**

  Sub-sampling at the client side [DSSJT94] allocates \(n\) times the playback bandwidth for \(n\) times fast viewing. This approach is associated with statistical quality-of-service (QoS) guarantees. With a low system load, its probability of providing immediate access to full-resolution FF and FR is high. When bandwidth is scarce due to a high system load, service is either delayed or provided by sacrificing resolution. This approach is conceptually simple but may need a complex decoder for high speed decompression or real-time frame reduction. Moreover, it may waste network and disk bandwidth.

  Sub-sampling at the server side is conceptually similar to sub-sampling at the client side, except that the real-time frame reduction is performed at the server side. This approach does not increase the network bandwidth, however it does waste disk bandwidth. More sophisticated variations of this method can reduce this overhead (see [CKLV95, SV95]).

  Segment skipping [CKY94, ORS96] skips a fixed number of blocks to achieve the desired playback rate by controlling the placement of blocks on disk drives. For example, to provide five times fast forward, this

---

\(^2\)The processing may be done at the server side, the client side, or both.
method displays one block out of five consecutive blocks. The advantage of this approach is its scalability because it requires no extra network and disk bandwidth to support various fast rates. However, it results in unusual previewing because it skips blocks, not frames.

- **Offline Processing**

[And96] proposed a method to implement separate fast-forward and fast-rewind files by selecting and pre-processing frames, either before or after compression. For example, to create a five times fast-forward file, every fifth frame is selected from the original movie before compression. This collection of frames is encoded in the regular manner (e.g., using MPEG) and stored in a separate file. Cross-references among the different files are maintained to enable a user to switch between versions. This method requires neither additional network nor extra disk bandwidth. Moreover, video quality is not degraded. However, it requires extra storage space for additional files. In Section 3 we detail our design and state our contributions.

The rest of this paper is organized as follows. In Section 3 we introduce the design of our technique, which is then detailed for Motion JPEG compressed videos in Section 4. Section 5 extends our approach for MPEG compressed videos. The performance of the proposed scheme is described in Section 6. We conclude with the future research directions in Section 7.

### 3 Design Overview

The design presented here is based on an offline approach, i.e., it implements the fast forward and fast rewind functionalities by maintaining separate, pre-processed versions of the normal-speed clip. Figure 2 shows the terminology that is used throughout the rest of this paper. A presentation consists of one normal clip and several eXPressS (XPRS) clips. Each XPRS clip corresponds to a specific speed-up factor and direction (forward or reverse). If a user invokes fast-forward during an active display, the system
switches from the normal clip to the corresponding XPRS clip to provide the effect of on-demand fast-forward. For example, a presentation (say Batman) might have a fast-forward version with five times speed (Batman_{FF5X}), a fast-forward version with ten times speed (Batman_{FF10X}), and a fast-rewind version with five times speed (Batman_{FR5X}), as shown in Figure 2.

In order to switch between the different clips of a presentation based on user activities, a method of jumping to the appropriate location in each clip must be provided. These locations are referred to as Random Access Points (RAPs) [And96]. Their cross-references, which map between the frames of different clips, must be maintained. In the example of Figure 2, four mappings are necessary. One maps the frames of Batman\textsubscript{N} to those of Batman\textsubscript{FF5X}, Batman\textsubscript{FF10X}, and Batman\textsubscript{FR5X}. This enables a user to switch to any one of the available FF or FR versions during normal display. The remaining three map the frames of each XPRS clip to one another and the normal clip. An example use of these mappings is to enable a user to switch to fast-rewind directly from fast-forward.

The focus of this study is how a Presentation Manager (PM), which controls the display at the client side (e.g., an extended TV set-top box or a PC-TV) gains access to RAPs. Previous designs have suggested to maintain RAP index files, presumably at the server side [And96]. There are several disadvantages to this approach. First, the server must maintain a separate database of RAP index files that can be accessed very
quickly (e.g., a real-time or main memory database). This resource could become a bottleneck and limit the scalability of a server. Second, because of client buffering and networking delays the server may not know exactly which frame the client is displaying at the very instant a VCR function is invoked. Therefore, the transition to the new clip may not be very smooth\(^3\).

Our design and implementation techniques enable a server to scale to thousands of streams by avoiding the formation of bottlenecks. The two key ideas that facilitate scalability are as follows. First, the RAP mappings are not maintained centrally (e.g., at the server). Instead, they are interleaved between frames of clips in support of distributed processing. The mappings are organized such that the relevant information is available when the user invokes a VCR functionality. The existence of these mappings does not impact a hardware decoder (e.g., MPEG, MJPEG) to display a presentation: e.g., with an MPEG-2 decoder that utilizes the standard, the interleaved records can be represented as user data which is ignored by the decoder. Alternatively, a simple process (either hardware or software based) can filter these mappings from the data stream and redirect them to the VCR functions circuits, thereby avoiding their transmission to the decoder. This means that the system reads mappings on behalf of a display even though the user might not invoke a VCR functionality. The wasted bandwidth (and storage space) attributed to retrieving this mapping information is negligible, less than 0.2\% in our prototype implementation, see Table 4(b).

The second key idea is as follows. A PM interprets the interleaved mappings on demand when a user invokes a VCR function. This off-loads the processing of mappings to the PMs in order to avoid the formation of bottlenecks at the server. Furthermore, the transition between different clips will be precise because the client can determine precisely the last decoded frame when a VCR function is invoked. **In essence, with our technique, a small fraction of the mapping information is staged at the right place (a PM) at the right time (when the user invokes a VCR functionality) to facilitate**

\(^3\)This problem may be avoided by encoding frame numbers into each stream or counting the frames at the client side and submitting this information together with a VCR-command to the server.
distributed information processing to realize a scalable server.

4 Methodology with Motion JPEG

The design described in Section 3 depends to some extent on the data format assumed to encode a digital video clip. In this section, we will first outline our design and implementation based on a Motion JPEG data format\textsuperscript{4}. In Section 5 we will subsequently outline our technique for the popular MPEG standard.

4.1 MJPEG Data Format

The Motion JPEG (MJPEG) format as implemented by the Parallax XVideo700\textsuperscript{TM} system supports digital video based on the Joint Photographic Experts Group (JPEG) image compression standard. Each frame is encoded based on the JPEG algorithm with no inter-frame dependencies. A MJPEG movie file starts with a fixed length header data structure that describes critical attributes of the movie such as height and width of video frames, quality factor of the JPEG images, desired playback rate, audio specifications, etc. A linear sequence of frames follows the header data. Each individual frame is structured as shown in Figure 3. Image data is preceded by a field that specifies its length. A LOAD_JPEG delimiter indicates the start of the image data while a LOAD_AUDIO delimiter precedes the fixed-size audio slice. Both, audio and video, components may be present or optionally one of them may be omitted. A frame always ends with an END_FRAME tag. The length of the image data varies depending on its complexity and the compression factor attained by JPEG. The variable data length in combination with a fixed number of frames displayed as a function of time explains the variable bit rate requirement of MJPEG. Assuming a presentation consists of $x$ frames, the last frame of a clip is followed by a Frame Offset Array (FOA) that

\textsuperscript{4}Motion JPEG is not as well standardized as MPEG. We base our discussion on the format specified by Parallax Graphics Inc. [Par95].
Figure 3: Parallax MJPEG movie format

consists of \( x \) elements. Each element \( i \) of FOA contains the offset of frame \( i \) relative to the beginning of the file. The offset of the FOA itself is stored in the header data structure.

To display an MJPEG video, the client application must read the header data and set up the video display window according to the specified attribute values. Next, it reads and plays the frames in sequence at the frame rate specified in the header.

4.2 Interleaved RAP Mappings

In our approach, the RAP mapping information is represented as *records*. Each record consists of a set of pointers to its corresponding RAPs. We interleave records between the frames of clips to enable the system to switch between clips of the same presentation. This section specifies the structure, content, and location of these records.

Assume that a content provider desires to offer a presentation (say Batman) with \( n \) different VCR
The first pass is as follows. The algorithm determines the maximum scan speed $y_{max}$ ($y_{max} = 10$ in our example due to $FF_{10X}$) among $n$ speeds. Then the original MJPEG file is scanned sequentially and $n + 1$ clips with empty interleaved records are generated: one XPRS clip for each of the $n$ scan functionalities (Batman$_{FF_{5X}}$, Batman$_{FF_{10X}}$, and Batman$_{FR_{5X}}$), and one clip for normal display (Batman$_{N}$).

The size of RAP records is fixed to hold $n$ RAP pointers (e.g., $n$ integers or $4n$ bytes assuming a 32 bit architecture). For the normal clip (Batman$_{N}$), each pointer corresponds to one of the scan functionalities. The order of the pointers within a record is $\langle \text{Ptr}_{FF_{aX}}, \ldots, \text{Ptr}_{FF_{bX}}, \text{Ptr}_{FR_{cX}}, \ldots, \text{Ptr}_{FR_{dX}} \rangle$, with $a < \ldots < b$ and $c < \ldots < d$. For example, in Figure 5, the pointer order is as follows: the first pointer maps to $FF_{5X}$, the second to $FF_{10X}$, and the third to $FR_{5X}$ (see the arcs labeled 1, 2, and 3). This order is important

---

Figure 4: Two pass algorithm for inserting RAP mappings

---

5 File sizes are not shown to scale. A FF$_{5X}$ version of a clip would be approximately 20% the size of the $N$ version (or less if the audio portion is removed).
Figure 5: File layout for different versions of an MJPEG presentation

and must be maintained by the system (for example in the FileVersions relation which will be detailed in Section 4.4). For a given XPRS version the pointer field that would normally map to its own clip is instead used to refer back to a frame in the normal speed version (N). Each of the other pointers is as before. For example, with Batman_FF_{sX}, the second and third pointer of each record continue to be offsets to Batman_FF_{10X} and Batman_FR_{sX}, respectively. However, the first pointer is an offset to Batman_N (see arcs 4, 5, and 6 in Figure 5). Similarly, for Batman_FF_{10X}, the second pointer is an offset into Batman_N while the other two are offsets into Batman_FF_{sX} and Batman_FR_{sX}, respectively. Note that a record precedes each frame in an XPRS clip. In the normal clip (Batman_N), a record precedes the first frame and a record is inserted after every $\frac{10}{n_{max}}$ frame (tenth in our example).

The detailed construction of each of the $n + 1$ clips of a presentation is as follows. A new header is generated followed by the first RAP record. In addition to the information listed in Section 4.1, the header is extended to maintain: (1) the size of each record, and (2) the number of frames that appear
between two consecutive records. The algorithm retrieves each frame \( i \) of the original MJPEG movie and inserts it into the normal clip. After every \( y_{\max} \) frames, it inserts a record into the normal clip that corresponds to the next sequence of \( y_{\max} \) frames. For each of the \( n \) XPRS clips corresponding to a scan functionality with speed \( z \), it inserts frame \( i \) only if \( i \) modulo \( z \) equals zero. A record is inserted before each such frame. For the fast-rewind XPRS files, the frames and records are stored in reverse order. In our example scenario, the size of each record for Batman\(_N\), Batman\(_{FF_{5X}}\), Batman\(_{FF_{10X}}\), and Batman\(_{FR_{5X}}\) is twelve bytes long. A record appears after every ten frames in Batman\(_N\). For the remaining three XPRS files, one frame separates two adjacent records. While Batman\(_{FF_{5X}}\) contains one out of every fifth frame, Batman\(_{FF_{10X}}\) contains one out of every tenth frame. One out of every fifth frame is inserted in reverse order into Batman\(_{FR_{5X}}\).

We eliminate the audio portion of each frame for XPRS files because its display is non-continuous and would translate into a collection of incomprehensible noises. This optimization saves disk space, disk and network bandwidth, and memory at the Presentation Manager (PM) side when the user invokes a scan functionality.

The first pass is complete when the last frame of the normal clip has been processed. At this point, an FOA with modified offsets is appended to the end of each of the \( n + 1 \) clips. Recall that the FOA contains the offset of each frame in a clip and because numerous RAP records have been inserted during the processing of pass one the offsets of frames have changed. For frames that are preceded by a RAP record, the offset of the beginning of the record is used (see Figure 5).

The second pass consists of two stages. During stage one, the algorithm fills up the contents of the records interleaved in the normal clip (e.g., Batman\(_N\)). This processing is as follows. For each frame \( X \) in Batman\(_N\), its corresponding frame numbers within the XPRS clips are computed using the equations of Table 2. These frame numbers (denoted \( Y \) in Table 2) are then converted to offsets (i.e., RAP pointer
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>Frame number for the current version</td>
</tr>
<tr>
<td>$Y$</td>
<td>Frame number for the new version</td>
</tr>
<tr>
<td>$S_X$</td>
<td>Speed of the current version</td>
</tr>
<tr>
<td>$S_Y$</td>
<td>Speed of the new version</td>
</tr>
<tr>
<td>$F_N$</td>
<td>Number of frames in the normal clip</td>
</tr>
<tr>
<td>$F_Y$</td>
<td>$[F_N/S_Y] - 1$</td>
</tr>
<tr>
<td>$F_I$</td>
<td>Number of frames between two records in the normal clip</td>
</tr>
</tbody>
</table>

Table 1: Parameters and their Definitions

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>from normal clip to FF clip</td>
<td>$Y = \lfloor X/S_Y \rfloor$</td>
</tr>
<tr>
<td>from FF clip to normal clip</td>
<td>$Y = \lfloor X * S_Y / F_I \rfloor * F_I$</td>
</tr>
<tr>
<td>from normal clip to FR clip</td>
<td>$Y = F_Y - \lfloor X/S_Y \rfloor$</td>
</tr>
<tr>
<td>from FR clip to normal clip</td>
<td>$Y = \lfloor (F_N - 1) - X * S_X / S_Y \rfloor * F_I$</td>
</tr>
<tr>
<td>between two similar(^a) XPRS clips</td>
<td>$Y = \lfloor X * S_X / S_Y \rfloor$</td>
</tr>
<tr>
<td>between two dissimilar(^b) XPRS clips</td>
<td>$Y = F_Y - \lfloor X * S_X / S_Y \rfloor$</td>
</tr>
</tbody>
</table>

\(^a\)Similar XPRS clips have the same type but different speeds, e.g., $FF_{5X}$ and $FF_{10X}$.
\(^b\)Different XPRS clips have different types, e.g., $FF_{5X}$ and $FR_{5X}$.

Table 2: Mapping equations between frames among different clips

values) by looking up each element $Y$ in the FOA of its clip.

**Example 4.1:** Assume $Batman_N$ consists of 4000 frames with 10 frames between two records. Given frame 99 in $Batman_N$, its corresponding frame number in $Batman_{FF_{5X}}$ is $Y = \lfloor X/S_Y \rfloor = \lfloor 99/5 \rfloor = 19$, in $Batman_{FF_{10X}}$ it is $Y = \lfloor X/S_Y \rfloor = \lfloor 99/10 \rfloor = 9$, and in $Batman_{FR_{5X}}$ it is $Y = F_Y - \lfloor X/S_Y \rfloor = \lfloor (4000/5) - 1 \rfloor - \lfloor 99/5 \rfloor = 799 - 19 = 780$. The offset of a frame (e.g., 19) in $Batman_{FF_{5X}}$ is determined by looking up its element (i.e., number 19) in the FOA of $Batman_{FF_{5X}}$. □

At the end of stage one, the records interleaved in $Batman_N$ contain the appropriate offsets into XPRS clips.

During the second stage, the algorithm visits each XPRS clip and computes the contents for the empty pointers of its records, one at a time. Specifically, for each frame $X$ in an XPRS clip (say $Batman_{FF_{5X}}$), the algorithm determines the corresponding frames in the other clips ($Batman_N$, $Batman_{FF_{10X}}$, and
Batman_{FR,5X} using the equations of Table 2. By fetching the offsets from the FOAs of the clips, the content of a record in the clip is completed.

**Example 4.2:** With the assumption made for Example 4.1, frame 19 in Batman_{FF,5X} now corresponds to frame 90 in Batman_{N}, frame 9 in Batman_{FF,10X}, and frame 780 in Batman_{FR,5X}, respectively. By fetching the 90th offset in the FOA of Batman_{N}, 9th offset in the FOA of Batman_{FF,10X}, and 780th offset in the FOA of Batman_{FR,5X}, the algorithm determines the proper offsets for frame 19 in Batman_{FF,5X}. Note that frame 90 in Batman_{N} is referenced by frame 19 in Batman_{FF,5X} (not 95, as derived from 19 × 5). This is because we want to jump to a frame that is prefixed with a record. □

The above procedure is repeated for all the frames in a XPRS clip and sub-sequentially for all the XPRS clips. At the end of stage two, construction of all the clips is complete.

### 4.3 Record Placement

A record must be placed in front of a frame that we want to have direct access to. For example, if a record precedes every frame in a clip then we can potentially jump from and to every frame in that clip. If, on the other hand, records exist only every 10th frame then we can only access every 10th frame directly and any access that would fall in between would need to be rounded up or down to the closest indexed frame. More generally the tradeoff is access granularity vs. wasted disk space and bandwidth. (Access granularity translates into time accuracy in our applications.)

When interleaved records are used to switch between different versions of a presentation the following lower bound can be put on the interleaving factor in the normal clip: insert a record only every \( y \)th frame where \( y \) is the smallest speed-up factor of any of the XPRS versions we want to support for this presentation. The reasoning behind this lower bound is as follows. When switching from the normal to the \( y \)-times fast
version, the frame number must be rounded such that it is divisible by $y$ (since only those frames exist in the $y$-times fast version). Similarly, when switching from the $y$-times fast to the normal version the current frame number must be multiplied by $y$ to find the correct normal speed frame number. In either case, switching information is only needed before every $y^ih$ frame in the normal version and inserting more records would not improve the switching accuracy.

When a presentation supports multiple XPRS speeds there are various ways to choose the interleaving factor in the normal and the XPRS versions. Again, the tradeoff is between switching accuracy and space (bandwidth) overhead. In our implementation, we chose to insert a record before every frame in all the XPRS versions and before every $\max(y_1, y_2, \ldots, y_n)^{th}$ frame in the normal version, where $y_i$ is the speed-up factor of the $i^{th}$ XPRS clip.

Because frame numbers sometimes need to be rounded we have a choice of rounding up or down. From an algorithmic point of view either way will work fine. Even visually there will not be any noticeable difference in most cases. Only when the rounding involves several tenth of frames the choice might become important. In this study, we are always using the floor function (i.e., rounding down) to simplify the discussion.
4.4 A Multi-disk Server and its Presentation Managers

This section outlines how the methodology of interleaved RAP records can be supported by an actual implementation of a video server. This will be done in the context of Mitra [GZS+97], a video server prototype that was developed at the database laboratory of the University of Southern California, Los Angeles. The functionalities needed at both the PM and the server sides will be discussed.

A Mitra server supports three relations to maintain the identity of different presentations and their clips: FileVersions, FileOrg, and MediaTypes (see Figure 6). FileVersions is partially cached at a Presentation Manager (PM) in support of both VCR functionalities and to enable a user to identify the available titles. We start with a description of these relations. Subsequently, we describe the PM, the server, and how they interact.

FileVersions maintains each presentation known to Mitra and its corresponding clips. Each entry in this relation is of variable length, consisting of the presentation name, the number of scan-functionalties supported for this presentation, and for each functionality a triple consisting of type, speed, and field-id. Valid types are N (normal), FF (fast-forward), and FR (fast-rewind). The speed field specifies the speed-up factor for a given FF or FR functionality. (Speed is one for type N.) Finally, field-id corresponds to a specific field of the record inserted between the frames of the clips. It identifies which field of this record offsets into the other clips corresponding to this VCR functionality. In our Batman example with three scan functionalities, the entry for this presentation in FileVersions would be as follows:

(Batman, 4, N, 1, -, FF, 5, 1, FF, 10, 2, FR, 5, 3).

The MediaTypes relation contains a listing of all the media types known to the system. For each media type, this relation maintains its block size, i.e., at what granularity a clip of this media type should

---

Note that, except for FileVersions, these relations are necessary for the normal operation of a Mitra continuous media server even without the FF/FR support.
Figure 7: Example of version switching (normal to FF_{5,X})

be retrieved.

At the server, each clip (both normal and XPRS clips) is represented as a unique file. The name of a file is a concatenation of its presentation-name, type, and speed. The four files that represent Batman and its XPRS clips are named: Batman.N.1, Batman.FF.5, Batman.FF.10, and Batman.FR.5. With a multi-disk server that consists of D disks, a clip is partitioned into fix-sized blocks. The size of each block is determined by the clip’s media type. The blocks of a clip are assigned to the disks in a round-robin manner (starting with an arbitrary disk) to distribute the load of a display evenly across the disks. FileOrg maintains the name of each presentation, its media type (to determine its block size) and the disk that contains its first block. Section 4.4.2 details how this information is used.
4.4.1 Presentation Manager

The Presentation Manager (PM) represents the display station of a user connected to Mitra. Through the PM the user can request audio and video clips from the Mitra server. When the server starts to stream the data to the user workstation, the PM is responsible for decoding and displaying the data. Other (VCR-like) functions provided by the PM include: pausing and resuming, fast-forwarding, fast-rewinding, and stopping a stream. We will focus on how the VCR functions are supported.

Figure 8 shows the internal structure of a PM process. At the beginning of a session the PM will retrieve a list of the available program material (i.e., presentations) from the server. Specifically, the PM caches a portion of the FileVersions relation which contains vital information about the available titles.

When a user requests the display of a presentation (say Batman), the PM enables the fast-forward and/or fast-rewind buttons of the user interface based on the version types applicable for this clip. A pull down menu is configured to enable a user to choose between the available fast-forward (and fast-
rewind) speeds available. Next, a control message with the following format \(<\text{PLAY}, \text{‘Batman.N.1’}, 0\>\) is dispatched to the server. The third field identifies the offset within the clip that the display should start with. The server sends the blocks of Batman.N.1 to the PM starting with the block that contains offset 0 (the computation of the block number is a function of the offset and described in Section 4.4.2). Upon the arrival of this block, the PM extracts the header to compute the window size, the rate at which the frames should be displayed, length of the records interleaved between the frames (12 bytes in our example) and the number of frames between two records (10 in our example). This information is the context for the active display. This context remains valid until the user invokes the Stop button or the display terminates.

The PM retains the RAP record that precedes each sequence of frames that are displayed (step a in Figure 7). In our example, the PM caches the first RAP record of Batman.N.1 when displaying its first ten frames. This record is replaced with the next record that precedes the display of the next ten frames. With no manual intervention this process repeats until all frames of the clip have been displayed. However, when the user invokes fast-forward (say \(\text{FF}_{5X}\)), the current display is terminated and the PM issues a new play command such as \(<\text{PLAY}, \text{‘Batman.FF.5’}, 525,000\>\). The server extracts the second field and interprets it as the name of the XPRS clip that is required by the display. The third field specifies that the display of the XPRS file should start at offset 525,000 (in this example). It is determined by the PM as follows. First, the entry corresponding to the active clip (in the cached FileVersions relation) is employed to compute the field-id \(f\) of the record interleaved between every ten frames that would offset into the frame of the needed XPRS file (field-id = 1 for Batman\(F_5\)) (step b in Figure 7). Next, using the record that corresponds to the sequence of frames currently being displayed, the PM looks up the \(f^{th}\) field within this record to retrieve the byte-offset of 525,000 (step c in Figure 7). After the PM sends the \(<\text{PLAY}, \text{‘Batman.FF.5’}, 525,000\>\) command, the server will respond by transmitting the block that contains byte-offset 525,000 of the Batman\(F_{5X}\) clip (step d in Figure 7). The details of the server-side computation are outlined in Section 4.4.2. The PM is aware of the transmission block size used for the new
clip and can therefore determine the correct starting offset within the first data block. For example, with a block size of 256 Kbytes, the offset of the record just before the first frame to be displayed is \((525,000 \mod 262,144) = 712\).

### 4.4.2 Server Operation

With any PLAY command the server receives (1) the name of the file representing that clip, and (2) the offset of the frame at which the display should start. Using this information the server goes through the following steps to start the new display. From the FileOrg relation it determines the media type of the referenced clip and the disk that contains its first block (termed \(d_{start}\)). Using the MediaType relation, the server determines the size of the blocks for this clip (termed \(blocksize\)). The block (denoted \(b_j\) that contains the offset (and therefore the frame) referenced by the display command is computed as:

\[
 b_j = \left\lfloor \frac{offset}{blocksize} \right\rfloor
\]

The disk \((d_i)\) that contains this block is identified by:

\[
 d_i = (b_j + d_{start}) \mod D
\]

where \(D\) denotes the total number of disks of the server. Once the correct starting block \(b_j\) has been located on the correct disk \(d_i\), the server streams the data to the PM (step \(d\) in Figure 7). A control message precedes the transmission of this block to inform the PM of the size of each block. Using this information and the offset, the PM can determine the starting address of the frame that should initiate the display. To illustrate, consider a simple example. Assume \(D = 8\), \(offset = 525,000\), \(d_{start} = 3\), and \(blocksize = 256\) Kbytes. It follows that block \(b_j = \left\lfloor \frac{525,000}{262,144} \right\rfloor = 2\) contains the required frame. This block
resides on disk \( d_i = (2 + 3) \mod 8 = 5 \). Subsequent blocks are retrieved in the usual round-robin manner.

### 4.5 Online Creation of Normal and FF Versions

The normal clip of a presentation and its corresponding fast-forward XPRS clips can be constructed online while it is recorded. This is because, at the time a record needs to be interleaved in all versions, the system has the necessary information that should be stored in the interleaved records (both the normal clip and its XPRS clips). For example, since the addresses of the first frames in all the versions are known before recording, the first record of all versions can be constructed. Because the records in the normal version are interleaved according to the fastest FF version, this guarantees the availability of next addresses that are needed to build the next record of all fast-forward XPRS clips. For obvious reasons, online construction of FR versions is impossible.

### 5 Extension to MPEG

Our approach can be adapted to other video formats such as MPEG (Motion Picture Experts Group). When compared with MJPEG, MPEG provides for both inter-frame and intra-frame compression. MPEG’s inter-frame compression results in several different frame types in the compressed domain: I-frames, P-frames, and B-frames (for details see [ISO92]).

Andersen [And96] proposed to use only the I-frame type in the XPRS files. Our approach allows the system to use all types of frames when compressing XPRS files. This results in great reduction in the size of XPRS files since I-frames typically contain about 3 times as many bits as P-frames and about 15 times as many bits as B-frames. Also, it results in a low network bandwidth requirement due to the smaller sizes of XPRS files. However, it restricts access to a group of pictures (GOP). GOP denotes a concept in the
MPEG syntax that denotes a sequence of pictures that contains exactly one I-frame which must also be the first frame of this sequence.

5.1 Switching Strategies

I-frames are the only independently encoded frames that allow random access to the compressed files. Hence, when the system switches from one file to another, it must start from an I-frame in the target file. Since a GOP contains a single I-frame which is also the first frame, our technique essentially maps from one GOP to another. Two strategies are proposed to handle the mapping from one file to another: GOP and Skip, see Figure 9. They are explained in turn.

- **GOP Strategy**

  ![GOP Strategy](image1)

  (a) GOP Strategy

- **Skip Strategy**

  ![Skip Strategy](image2)

  (b) Skip Strategy

Figure 9: Switching process with GOP and Skip strategies

- **GOP Strategy**

  In this strategy, whenever a user switches from normal display to XPRS clips or vice versa, the system must begin from an I-frame and display the next frames sequentially as shown in Figure 9(a). When the system switches from one file to another, the target frame might be a P-frame or a B-frame. In this case, the system must start from the I-frame that precedes the target frame. As a result, this strategy may suffer from an accuracy problem in the switching process to the target frame. This accuracy problem, on the average, might lead to a delay of $\frac{\text{GOP Size} + 1}{2 \times \text{fps}}$ seconds before displaying the target frame, where fps is
the number of frames that get displayed per second. The GOP size refers to the total number of pictures that are contained in a GOP. If the GOP size is 15 and the fps is 30, then the average delay time before displaying the target frame is approximately 0.26 second. One way to minimize the average delay is to decrease the GOP size.

- **Skip Strategy**

  This strategy is similar to the GOP strategy. However, with this strategy, the system attempts to reduce the average delay time by only displaying the reference frames until it reaches the target frame; then it displays the frames sequentially as depicted in Figure 9(b). A reference frame is either an I-frame or P-frame. This enables the system to eliminate the overhead time required to display B-frames that arrive prior to the target frame. With this strategy, the average delay time due to displaying the reference frames is \( \frac{\text{Number of Reference Frames in GOP} + 1}{2 \times \text{fps}} \) seconds. If the GOP size is 15, as shown in Figure 9(b), and fps is 30, then the average delay time is 0.10 second. The system can minimize the average delay time by minimizing the number of the reference frames in a GOP.

5.2 **Mapping of Frames**

Assume that a user invokes FF while displaying a clip. The system must be able to compute which frame is being displayed from the normal playback and find its corresponding frame in the FF file and vice versa.

To enable the system to compute a mapping of frames from one file to another, we derive the equations of Table 3. Table 4.2 defines the parameters used in these equations. The speed of a file refers to its speed up rate compared to the normal playback file. For instance, a ten times FF file displays the presentation at a rate that is ten times faster than that of normal previewing. Note that since we are taking the ceiling for \( Y \), which is the target frame within a GOP, it might be equal to \( L \), which is the GOP size of the target file. If this happens, then the value of \( Y \) should be reset to zero and \( J \) should be incremented by one. When \( Y \)
<table>
<thead>
<tr>
<th>Mapping</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>from normal file to FF file</td>
<td>( J = \left\lfloor \frac{S_Y}{S_X} x \right\rfloor )</td>
</tr>
<tr>
<td></td>
<td>( Y = \left\lfloor \frac{x \mod (S_Y \times L)}{S_Y} \right\rfloor )</td>
</tr>
<tr>
<td>from FF file to normal file</td>
<td>( J = \left\lfloor \frac{x \times S_X}{S_Y} \right\rfloor )</td>
</tr>
<tr>
<td></td>
<td>( Y = \left(\left\lfloor \frac{X \times S_X}{S_Y} \right\rfloor \right) \mod L )</td>
</tr>
<tr>
<td>from normal file to FR file</td>
<td>( J = \left\lfloor \frac{(T_Y - 1) \times S_Y - X}{S_Y \times L} \right\rfloor )</td>
</tr>
<tr>
<td></td>
<td>( Y = \left\lfloor \frac{(T_Y - 1) \times S_Y - X}{S_Y \times L} \right\rfloor \mod \left( S_Y \times L \right) )</td>
</tr>
<tr>
<td>from FR file to normal file</td>
<td>( J = \left\lfloor \frac{(T_X - 1) \times S_Y - (X \times S_X)}{S_Y \times L} \right\rfloor )</td>
</tr>
<tr>
<td></td>
<td>( Y = \left\lfloor \frac{(T_X - 1) \times S_Y - (X \times S_X)}{S_Y \times L} \right\rfloor \mod L )</td>
</tr>
<tr>
<td>from FF file to FR file</td>
<td>( J = \left\lfloor \frac{S_Y \times \left( X \times S_X \right)}{S_Y \times L} \right\rfloor )</td>
</tr>
<tr>
<td></td>
<td>( Y = \left\lfloor \frac{(T_Y - 1) \times S_Y - (X \times S_X)}{S_Y \times L} \right\rfloor \mod \left( S_Y \times L \right) )</td>
</tr>
<tr>
<td>from FR file to FF file</td>
<td>( J = \left\lfloor \frac{S_Y \times \left( X \times S_X \right)}{S_Y \times L} \right\rfloor )</td>
</tr>
<tr>
<td></td>
<td>( Y = \left\lfloor \frac{(T_X - 1) \times S_Y - (X \times S_X)}{S_Y \times L} \right\rfloor \mod \left( S_Y \times L \right) )</td>
</tr>
<tr>
<td>from FF file to FR file</td>
<td>( J = \left\lfloor \frac{S_Y \times \left( X \times S_X \right)}{S_Y \times L} \right\rfloor )</td>
</tr>
<tr>
<td></td>
<td>( Y = \left\lfloor \frac{(T_Y - 1) \times S_Y - (X \times S_X)}{S_Y \times L} \right\rfloor \mod \left( S_Y \times L \right) )</td>
</tr>
</tbody>
</table>

Table 3: Mapping equations between frames among different MPEG files

is equal to zero, it means that the system always starts from the I-frame pointed to by \( J \). Figure 10 shows an example of how to use the equations to map from one file to another.

5.3 Placement of Interleaved Records

The placement of interleaved records depends on the speed up rates and the GOP sizes of the files. Consequently, we can select the speed up rates and the GOP sizes to eliminate redundant information. Each interleaved record contains fields and each field contains an offset to a file. A new record may be added to a file whenever one of its fields is changed. The equation

\[
\text{Frequency of Changed Field (FCF)} = \left\lfloor \frac{L \times S_Y}{S_X} \right\rfloor
\]

can be used to detect how often a field gets changed. The definition of the parameters of this equation can be found in Table 4.2. The result of this equation might be zero because it is using the floor function. In
In this case, the value of FCF is adjusted to one because zero is an invalid result. This equation provides the number of frames after which a particular field in a record is modified.

For a particular file, we may have different FCFs for the different fields in a record. Redundancy can be minimized if these FCFs are multiples of each other. There must always be a record in front of each I-frame. Otherwise, if the system switches to an I-frame that does not have a record and it is immediately required to switch to a different file, then the system will fail to honor that request. Consequently, a record must be put in front of each I-frame even if this results in a redundancy. In this case, redundancy can be minimized if one of the FCFs is a multiple of the GOP size of the file.

To clarify the use of the equation, an example is provided in Figure 11. Even though this example shows how to calculate FCFs for only normal file and FF files, FR files can be computed in the same manner. After all, FR files are the same as FF files, except that they are ordered conversely.
Figure 11: An example for the placement of interleaved records

6 Performance Evaluation

To evaluate the performance, i.e., the switching time between different clip versions, of our design for VCR functionalities we implemented the necessary concepts in the Mitra server [GZS+97] and its PM clients. We then proceeded to impose a load on the system while at the same time exercising the VCR functions.

The experimental design was as follows. The system setup included one server consisting of two workstations with eight magnetic disks drives. A synthetic workload was imposed on the server with the arrival rates of user requests following a Poisson distribution. Four experiments were conducted with system loads of 45%, 60%, 70%, and 80%. A PM was modified to automatically generate requests for VCR functions. Specifically, the PM alternated between normal play and fast-forward of a clip until it reached
### Table 4: Performance evaluation results

<table>
<thead>
<tr>
<th>Version</th>
<th>Presentation</th>
<th>Switching Time</th>
<th>System Load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal</td>
<td>[sec]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Disney</td>
<td>Minimum</td>
<td>45%</td>
</tr>
<tr>
<td></td>
<td>Top Gun</td>
<td>0.391</td>
<td>48.5%</td>
</tr>
<tr>
<td></td>
<td>Hawaii</td>
<td>0.486</td>
<td>48.6%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum</td>
<td>3.621</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.204</td>
<td>58.03%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.803</td>
<td>58.03%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>1.587</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.675</td>
<td>20.56%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.056</td>
<td>23.61%</td>
</tr>
<tr>
<td>FF_{3X}, FR_{5X}</td>
<td>0.121%</td>
<td>FF_{10X}, FR_{10X}</td>
<td>0.121%</td>
</tr>
<tr>
<td></td>
<td>Disney</td>
<td>Minimum</td>
<td>45%</td>
</tr>
<tr>
<td></td>
<td>Top Gun</td>
<td>0.391</td>
<td>48.5%</td>
</tr>
<tr>
<td></td>
<td>Hawaii</td>
<td>0.486</td>
<td>48.6%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum</td>
<td>3.621</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.204</td>
<td>58.03%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.803</td>
<td>58.03%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>1.587</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.675</td>
<td>20.56%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.056</td>
<td>23.61%</td>
</tr>
</tbody>
</table>

(a) Storage space allocated for different clip versions

(b) Wasted storage space and network bandwidth attributed to RAP mapping information

(c) Clip switching time for different system loads

\[ \frac{2}{3} \] of the length of this clip. At that point it started to alternate between normal play and fast-rewind until it was back at \( \frac{1}{3} \) of the clip’s length. The normal play time was randomly chosen between 10 and 40 seconds and the FF/FR time between 10 and 30 seconds. The above sequence of version switches was repeated for two hours for each experiment to measure the switching time. The upper bound on the play times was deliberately chosen to be quite small to generate a significant number of switches.

Table 4(b) summarizes the results of our tests. As expected, the switching time increases with higher system loads. The minimum latency is approximately 0.5 seconds. The average delay experienced by a user is approximately 2.4 seconds under a heavy load of 80% on the server.

Other than quantitative performance different designs for VCR-like functions also exhibit different qualitative advantages and disadvantages. Because the frames in the XPRS clips are a subset of the normal version of a presentation, the digital FF and FR operations do not suffer from the artifacts that analog VCRs exhibit (streaks, increased picture noise). Moreover, the presented design results in a very smooth playback as compared with designs that skip whole blocks since exactly every \( y^{th} \) frame is displayed. A system manager may also choose different speed-up factors depending on the content of a movie. For example, a FF_{3X} version may be appropriate for a fast paced movie such as Top Gun whereas a FF_{10X}

\[ FF_{3X} \]

\[ FF_{10X} \]
clip may be better for a documentary. Compared to designs that increase the production rate at the server side to implement fast-forward or fast-rewind our approach does not increase the system load and hiccups due to load fluctuations are therefore avoided.

7 Conclusion and Future Research Directions

This study presented the design and implementation of scalable browsing techniques for Intranet video servers in support of high quality playback with low processing overhead. Using a prototype implementation of these techniques in Mitra [GZS+97], we validated the feasibility of our method as well as its performance. In the near future, we intend to implement our technique for MPEG.

References


