Providing Interactive Functions through Active Client-Buffer Management in Partitioned Video Multicast VoD Systems

Zongming Fei[1], Mostafa H. Ammar[1], Ibrahim Kamel[2], and Sarit Mukherjee[2]

[1] Networking and Telecommunications Group, College of Computing,
Georgia Institute of Technology, Atlanta, GA 30332-0280, USA
{fei,ammar}@cc.gatech.edu,
[2] Panasonic Information and Networking Technology Lab,
Panasonic Technologies,Inc., Two Research Way, Princeton, NJ 08540, USA
{ibrahim,sarit}@research.panasonic.com

Abstract. Multicast delivery is an attractive approach to the provision of a video-on-demand service because it scales well to a very large number of clients. The problem is how to provide interactive functions to individual clients within the multicast framework without compromising the scalability of the multicast paradigm. In this paper, we propose an active buffer management scheme to provide interactive functions in partitioned video broadcast. Our scheme lets the client selectively prefetch segments from broadcast channels based on the observation of the play point in its local buffer. We introduce the concept of feasible points which can guarantee the continuity of playback after resuming normal play following VCR actions. Our simulations show that the active buffer management scheme can implement interactive actions through buffering with a very high probability in a wide range of user interaction levels.

1 Introduction

A Video-on-Demand (VoD) service provides subscribers with a set of videos and sends a specific video to customers upon request. Using multicast to send popular videos has been demonstrated by many researchers to be an efficient way for their delivery [1–6]. There are two basic approaches to provide multicast video-on-demand services. One is on-demand batching, in which the server allocates a channel to a group of pending clients requesting the same video and sends it over that channel. These systems are *on-demand* in the sense that the server allocates a channel to send a video only if there are requests sent over upstream channels by the clients requesting the video.

Another approach is continuous broadcast VoD systems in which the server allocates one or several channels to one video and each channel sends the whole video or a part of it in cycles. These systems are not on-demand because they keep broadcasting videos even if there are no requests for them. They do not need upstream channels and are most suitable for broadcasting popular videos. A simple form of this approach is the pay-per-view model (or *staggered broadcast*) in which several channels broadcast a video periodically with staggered start times. In such a system, the maximum startup latency is equal to the length of the video divided by the number of channels allocated to this video. Pyramid broadcasting [7] divides a video into segments of exponentially increasing sizes and lets one channel broadcast each segment repeatedly. Segments are broadcast in the channels at a faster speed than the playback rate. The scheme greatly decreases the startup latency at the expense of requiring client buffering. Skyscraper broadcasting [8] modifies the distribution of segment sizes in the pyramid broadcasting scheme and keeps the broadcast speed the same as the playback rate. It also places an upper bound on the weight of maximum segment size to reduce the storage required at the clients. The greedy disk-conserving broadcasting scheme [9] generalizes the skyscraper scheme to the case where each client can download from more than 2 channels at the same time. We call these schemes *partitioned video broadcast* and will describe them in more detail in Section 2.1.

The multicast approach to video-on-demand satisfies requests of several users at one time and thus, to some extent, sacrifices the special requirements of each individual user. The startup latency is a tradeoff of some degradation of service quality for scalability. The desirable interactive functions (sometimes called VCR functions) for video-on-demand service are also difficult to provide in multicast systems. Some recent VoD work begins to deal with this problem. The work by Almeroth and Ammar [3–5] incorporates the interactive functions into a multicast delivery VoD system and introduces the concept of discontinuous VCR actions. The system uses client buffering to provide interactive functions and proposes *emergency interactive channels* be used when the client buffer contents are not sufficient for the desired interaction. The SAM protocol proposed by Liao and Li [10] uses synch-buffer and special interactive channels (called I-streams) to provide VCR functions. The work by Abram-Profeta and Shin [11] improves the SAM protocol by changing the shared synch buffers to separate buffers at each client and making it more scalable. Most of the work has addressed VCR functions in the environment of an on-demand batching video delivery system and depends on both client buffering and "interactive" channels to provide VCR functions. The client side buffering technique can be extended to apply to the continuous broadcast model of video delivery (including partitioned broadcast), but interactive channels are not available in this model. Even if we have interactive channels, it is not preferable because using interactive channels to provide VCR functions compromises the scalability of the multicast paradigm. As an alternative to interactive channels in a continuous broadcast system, interactive functions that cannot be handled through client buffering can be accomplished by switching broadcast streams and incurring a possible discontinuity.

In this paper, we propose an *active buffer management* scheme for providing VCR functions in continuous partitioned video broadcast systems. It is a generalization of our previous work on staggered multicast near VoD systems [12]. The active buffer management scheme uses client side buffering in a novel fashion that relies on the simultaneous availability of "past", "present" and "future" parts of a video. It lets the client selectively prefetch segments from broadcast channels based on the observation of the play point in its local buffer. We let clients make the decision of what to prefetch and when to prefetch from broadcast channels according to the current status of their buffers, because in such systems no backward channel is available from the client to the server [1].

The challenging part is how to deal with the unequal sizes in the partitioned broadcast scheme. After a VCR action moves the play point to a new place in the video stream and we resume normal playback, we may face a situation of discontinuity of playback because we cannot get the frames before their scheduled playback times due to the unequal segment sizes. To deal with this problem, we introduce the concept of *feasible points* that restricts a client to move only to those points within the broadcast stream that can preserve the continuity of playback. While the existing broadcasting schemes [8, 9] focus on reducing the startup latency and client storage, they do not deal with the problem of providing interactive functions for clients. We design a new video partitioning scheme that is more suitable for the interactive behavior of clients. We perform extensive simulations to explore the effects of various parameters of our scheme at different user interaction levels. Our simulations show that our scheme can implement VCR actions through buffering with a high probability in a wide range of user interaction levels.

The rest of this paper is structured as follows. In Section 2 we describe the schemes of the partitioned video broadcast and summarize the existing work of using client buffering scheme to provide VCR functions in video multicast. In Section 3 we propose an active client buffer management technique to provide VCR functions in partitioned video broadcast and in Section 4 we give the rules for deciding feasible points. In Section 5 we perform an extensive evaluation of the performance of the proposed scheme. The paper is concluded in Section 6.

2 Background

2.1 Partitioned Video Broadcast

Partitioned video broadcast divides a video into segments and sends each segment over one channel in cycles. Assume the bandwidth allocated to a video is B Mbits/sec[2]. The bandwidth is divided into equal-bandwidth channels and the bandwidth of each channel is equal to the playback rate, b Mbits/sec. The number of channels is $K = \lfloor \frac{B}{b} \rfloor$.

A video of length L is divided into K segments and channel n periodically broadcasts segment n, where $1 \leq n \leq K$. The relative weights of segments distinguish one broadcasting scheme from another. Usually a function $f(n) (1 \leq n \leq K)$ is given and the size of segment n is $\frac{f(n)*L}{\sum_{j=1}^{K} f(j)}$. The maximum startup latency is $\frac{f(1)*L/b}{\sum_{j=1}^{K} f(j)}$. The function $f(n)$ is designed to minimize the startup latency. One way to achieve this is to make $f(n)$ increase as fast as possible, subject to the continuity condition that guarantees the clients can download each segment of the video in a timely fashion with a given number of loaders

[1] If such a backward channel were available, then on-demand batching is a preferable option to continuous delivery of a static set of videos.

[2] How the server bandwidth is allocated among videos depends on the strategy of the server and is not considered here.

in order to playback without jitter. The function $f(n)$ can generate a series specifying the relative weights of each segment.

The skyscraper broadcasting scheme [8] assumes that each client has two loaders (can download from two channels at the same time as the client plays back from the buffer) and the series generated by the function $f(n)$ is $[1, 2, 2, 5, 5, 12, 12, 25, 25, \ldots]$. There is an upper bound parameter W, which sets $f(n)$ to W if $f(n) > W$. The greedy disk-conserving broadcasting scheme [9] generalizes the skyscraper scheme to the case in which each client can have more than 2 loaders. A series generated by function $f(n)$ with $4b$ disk bandwidth requirement for each client is $[1, 2, 4, 8, 14, 24, 40, 70, \ldots]$ [3].

2.2 VCR Actions

We consider a video as a continuous sequence of frames. The *play point* of a client is a point in the video which the client is currently playing out. *The destination point* of a VCR action is the point in the video when the client finishes the VCR action. A VCR action can be described as a pair *($\Delta t, \Delta \ell$)*, where Δt is the time it takes for the VCR action to complete and $\Delta \ell$ is the relative position of the destination point with respect to the current play point (assume it is p). Obviously $\Delta t \geq 0$ and the destination is $p + \Delta \ell$. The destination is in the forward direction if $\Delta \ell > 0$ and in the backward direction if $\Delta \ell < 0$. We assume that the part of the video (of length $\Delta \ell$) is evenly distributed in the time interval Δt. Assuming the VCR action is issued at the play point p and the normal playback rate is b, we consider following VCR actions.

1. Jump Forward/Backward (JF/JB): $\Delta t = 0, \Delta \ell \neq 0$.
2. Fast Forward/Backward (FF/FB): $\Delta t > 0, \frac{|\Delta \ell|}{\Delta t} > b$.
3. Slow Forward/Backward (SF/SB): $\Delta t > 0, 0 < \frac{|\Delta \ell|}{\Delta t} < b$.
4. Pause: $\Delta t > 0, \Delta \ell = 0$.
5. Play: $\Delta t > 0, \frac{\Delta \ell}{\Delta t} = b$, and Play Backward: $\Delta t > 0, \frac{\Delta \ell}{\Delta t} = -b$.

Note that *Play* is also described as a pair $(\Delta t, \Delta \ell)$ because we can denote the time from the moment we resume normal play until the moment we issue the next VCR action as Δt and the video length played as $\Delta \ell$. It is interesting to note that VCR actions can be determined by one parameter x defined as $x = \frac{\Delta \ell / \Delta t}{b}$.

2.3 Conventional Client Buffering Scheme to Provide VCR Functions for Video Multicast

Using client buffering is an intuitive solution to the problem of providing VCR functions [3, 10, 11]. We illustrate the conventional client buffering scheme in Figure 1 and focus on how the relative position of play point in the buffer is changed by each VCR action.

In Figure 1, we show a client buffer of size 100. The buffer can be looked as a sliding window over a usable portion of a video. It is composed of video

[3] After downloading the first segment and sending it directly to the display, the client only has to download from 3 channels ($3b$) at the same time. $4b$ disk bandwidth is required at this time because one b is used for sending data from disk to the display.

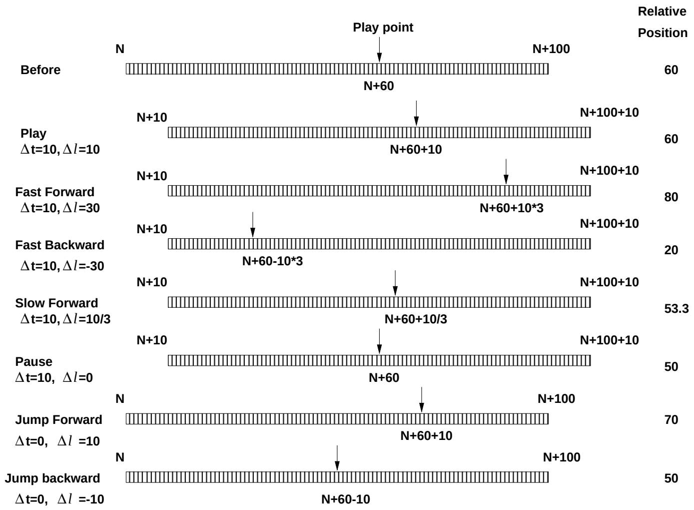


Fig. 1. The effects of VCR functions on play point position in the buffer

between the most recently downloaded frame and the oldest frame still held in the buffer. When a client begins to download a video, the buffer is empty. As the frames are downloaded from the multicast group, they are played back and the buffer is filled up at the same time. If there is no VCR function, the play point is the same as the most recent frame. When the buffer becomes full, the oldest frames are discarded to allow more recent frames to come in. Let us focus on the configuration labeled as *Before* in Figure 1. We assume the frame immediately before the oldest frame is N and the buffer is full. So the most recent frame is $N + 100$. This is also the frame that is currently being downloaded from the multicast channel. If some VCR functions have been performed, then the play point can be frames other than the most recent frame. In the example we assume the play point is $N+60$. We show how each VCR function will change the relative position of the play point in the buffer.

When a *play* action is issued, the play point moves forward at the same rate as the buffer boundaries. So the relative position of play point in the buffer does not change. When a *fast forward* (specified as $(\Delta t = 10, \Delta \ell = 30)$[4]) is issued, the boundaries move 10 units while the play point moves 30 units. So the relative position of the play point in the buffer moves forward 20 units. For the same reason, the *fast backward* moves the play point in the buffer backward 40 units. Though *pause* and *slow forward* do not play in the backward direction, the relative positions of the play point in the buffer do move backward because the boundaries of the buffer move forward. *Jump forward* and *jump backward* are implemented instantaneously, so the boundaries of the buffer do not change. *Jump forward* moves the play point forward and *jump backward* moves the play point backward. We can see that *play* does not change the relative position of the play

[4] The unit of time is a frame. $\Delta t = 10$ represents a time interval for 10 frames of video to be played back at the normal playback rate (=channel transmission rate). We use 3 as the ratio of the FF speed over play speed in this example.

point, *fast forward* and *jump forward* move the play point forward (called *forward VCR actions*), and all other actions move the play point backward (called *backward VCR actions*).

Problems with the scheme: At the beginning, no forward VCR actions can be performed because the relative position of play point is the same as the most recent frame (right boundary). After some backward VCR actions are performed moving the play point to the middle of the buffer, we can begin to perform forward VCR actions. The problem with the above buffer scheme is that the relative position of the play point is determined by the history of the VCR actions. The effects of VCR actions in the same direction are cumulative. When consecutive VCR actions in the same direction are performed, the play point will ultimately move to a boundary of the buffer. At this time, VCR functions in this direction cannot be implemented any more. It is at the mercy of VCR actions in the opposite direction to reverse the effects. Whether a VCR action can be implemented or not depends on what previous VCR actions are. The lack of self-adjustment is an inherent problem of the above buffer scheme. In the next section, we propose a new buffer scheme that takes advantages of the characteristics of partitioned video broadcast and provides VCR functions with a greater flexibility.

3 Providing VCR Functions in Partitioned Video Broadcast

3.1 The Idea of Active Buffer Management

As we know, the probability that a VCR function can be implemented with local buffers depends on many factors, including buffer size, play point within the buffer, buffer contents and management. The key is how to keep this probability high given a fixed buffer size. Our intuition is to keep the play point in the middle of the buffer so that there is a lower probability that VCR actions will move the play point out of the buffer. However, when a VCR action is performed, the relative position of the play point in the buffer will change. Our scheme is based on the use of a *buffer manager* that adjusts the contents of the buffer after VCR actions so that the relative position of the play point in the buffer stays in the middle [5]. We employ a policy of selectively downloading the segments based on the current position of play point in the buffer. This is possible in partitioned video broadcast because all the segments are repeatedly broadcast in the channels ready for downloading and therefore, a video's "past", "present" and "future" are being broadcast simultaneously at any point in time.

Let us explain the basic idea by a simplified example. Assume that the client buffer can hold three segments. At some point, the buffer holds segments $z, z+1, z+2$ and the play point is in the middle segment $z+1$. Now we observe the situation after one segment time. 1) Case 1: If there is no VCR function, the

[5] We may prefer a different position if the probabilities of forward and backward VCR actions are skewed. For example, if we have 90% of VCR actions in the forward direction we will keep the relative position of the play point in the left part of the buffer.

play point will be in the segment $z+2$. During this period, the loader finishes downloading the segment $z+3$ and the segment z is discarded. The buffer now holds segments $z+1, z+2, z+3$. The play point is still in the middle segment (now $z+2$). 2) Case 2: If a *fast forward* action is issued during this period and moves the play point to a position within segment $z+3$. We observe the play point is no longer in the middle segment. The idea is to let the buffer manager select the segments to be downloaded according to the position of play point. After observing that the play point is in the last segment $z+3$ of the three segments $z+1, z+2, z+3$ in the buffer, the buffer manager will download segments $z+4, z+5$ at the same time. After one segment time, the buffer will hold segments $z+3, z+4, z+5$ and the play point will move to segment $z+4$. Therefore, the play point is moved back to the middle segment.

We see from this example that by letting the buffer manager selectively prefetch segments based on the position of play point in the buffer, it can compensate the effects of VCR actions. Therefore, our scheme can tolerate consecutive VCR actions in the same direction and solve the problem of conventional client buffering scheme.

3.2 Destination Adjustment for VCR Actions

An important property of the destination of any VCR action is that *once the client resumes normal play from this destination, it should be able to play the rest of the video to the end without interruptions, if there are no further VCR actions.* We call all those destinations satisfying the above property *feasible points.*

Since we cannot predict user behaviors, there always exists a case in which the destination point of a user action is outside of the client buffer. Although we let the user specify an arbitrary destination for VCR actions, the algorithm adjusts it to the nearest feasible point automatically, if necessary. This is what we called *discontinuous interactive function* implementation, for the reason of the discontinuity between the specified destination and the adjusted destination [6]. By doing this, we can guarantee the continuity after we resume normal playback.

When the destination is not in the buffer, only those frames that are being broadcast at each channel are available immediately. Naively, we can select the nearest frame among them as the adjusted destination. This, however, will not always work because, in the unequally segmented schemes, they do not always satisfy the above property.

If all the segments are of equal size, the points being currently broadcast in each channel are feasible. It is complicated in partitioned video broadcast because of unequal segmentation. Not all those frames being broadcast are good candidates. If we consider buffering at the clients, the frames in the buffer can also be feasible points even if they are not being broadcast in the channels. In Section 4, we will give a detailed description of the rules for deciding whether a point is feasible and how to find the nearest feasible point with regard to a given destination.

[6] Another way to implement is that we may delay the VCR action until the buffer is filled with the contents we want. We do not need to adjust destination in this mode. However, the user has to wait before a VCR action can be implemented. It is also possible that a hybrid mode may give a good tradeoff between the destination adjustment and the response time.

In summary, our scheme uses an active buffer management technique to maintain the play point in the middle of the buffer. If the destination point of a VCR action is a point already in the buffer and it is also a feasible point at the finishing time, then the action can be provided by the frames from the buffer. If not, the destination is adjusted to the nearest feasible point and the continuity of the playback of the rest of video is guaranteed if there are no further VCR actions. We use the percentage of VCR actions implemented by the contents in the client buffer (abbreviated as *percentage of buffer VCR actions*) as a performance measure of our scheme. The higher this percentage, the better the interactive performance of the system will be. We also measure the difference between the specified destination and the adjusted destination for those VCR actions with destination changed for the feasibility reason. We use the *percentage of destination shift* (defined as the percentage of the difference over the length of VCR actions) as another performance measure. The smaller this percentage, the better the performance.

3.3 A VCR-oriented Broadcasting Series

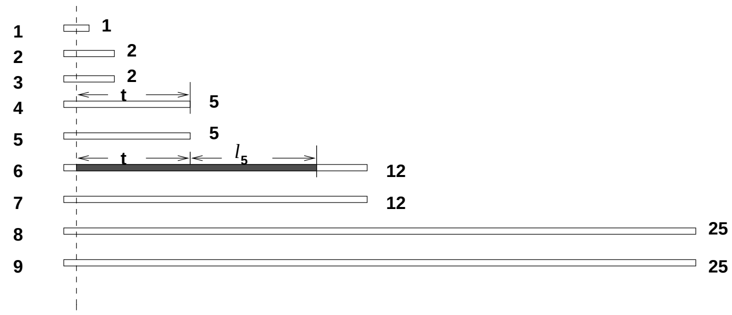


Fig. 2. A scenario in the skyscraper scheme

The design of a new broadcasting series is motivated by the problems we encountered during the process we explored the possibility of providing VCR functions in the skyscraper scheme [8]. Consider a scenario in the skyscraper scheme as in Figure 2. Assume the channels are broadcasting the frames at the positions identified by the dotted line. The length of the segment i is l_i. At this time, the client wants to jump to somewhere in segment 4. Assume segment 4 is not in the client buffer. The available frames are those being broadcast at each channel. The nearest one among them is what is being broadcast in channel 4. If we let the client jump to this destination and resume normal playback, after time $t+l_5$, we finish playing the rest part of segment 4 and the whole segment 5. Now we need the beginning part of segment 6. However, the channel 6 is now beginning to broadcast the last two units of segment 6. During the above $t + l_5$ period, we cannot get the beginning part of segment 6, either. This is because only the shaded part of segment 6 is broadcast in the channel during this period. We face the interruption of playback. If we let the client jump to channel 5, the interruption situation occurs even sooner (after t). We may consider adjusting the destination point to other channels. The problem is that jumping to any even number channel will lead to the same situation as to channel 4 and jumping to any odd number channel will lead to the same situation as to channel 5. The only feasible points are in those equal segments whose length is restricted by W. A similar situation of discontinuity can occur in the greedy scheme [9].

We design a VCR-oriented broadcast series based on two observations from the above analysis.

1. If a destination segment is not in the buffer and we still require that we can always select the frame being broadcast at this channel as the destination, its size must be equal to the size of the next segment to guarantee that the client can playback the next segment when it finishes playing the current segment.
2. The sizes of all segments beginning from the latter of the two above equal segments must satisfy the continuity condition to guarantee smooth playback of the rest of video, assuming the client can download from m channels simultaneously. To make the requirement small at the clients, we set m to 3.

We give a function with segments satisfying these two conditions as follows.

$$f(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 & \text{if } n = 2 \\ 4 & \text{if } n = 3 \\ f(n-1) & \text{if } f(n-1) \neq f(n-2) \text{ and } f(n-2) \neq f(n-3) \\ f(n-2) * 2 & \text{otherwise} \end{cases}$$

It generates the following segment sizes: $1, 2, 4, 4, 8, 16, 16, 32, 64, 64, \cdots$.

We use a parameter u to limit the maximum value of the series. We set $f(n) = f(n-1)$ if $n > u$. So all $f(n)$ with $n \geq u$ are equal to $f(u)$. This restriction also limits the buffer requirements for clients.

The loading strategy for the clients is simple if there are no VCR functions. At the beginning three loaders are allocated to segment 1,2 and 3. Each will load the assigned segment as soon as it begins. The downloaded segments are put into the buffer. The display gets the frames from the buffer or from the channel at the same time when they are sent to the buffer. The playback of the video begins as soon as the loader begins downloading the first segment.

When a loader finishes loading segment j, it will be assigned to segment $j + 3$. The loader downloads the segment when it is broadcast in the channel at the latest cycle before its playback time. The reasoning about the continuity of playback, and the generalization of the series can be found in [13].

3.4 VCR Function Implementations with the Active Buffer Management Scheme

When clients download or playback video segments, they go through two phases. A segment k is in the pyramid phase (unequal segments) if $1 \leq k \leq u - 1$ and in the equal segment phase if $u \leq k \leq K$. Clients are required to have at least three loaders and three buffers. The size of each buffer is the same as that of the maximum segment. It can hold either one equal phase segment or several pyramid phase segments. During pyramid phase, we may have more than 3 segments in the buffers at the same time.

We distinguish two components that work together to provide VCR functions. One is the *player*, which accepts user interaction commands and plays back the video as requested. The other is the *loader/buffer manager* (or simply manager), which manages the loaders/buffers and decides which channels the client should listen to and download segments.

Player The player keeps playing back the contents from the buffers until it accepts a VCR command. It then makes the decision based on the availability of the contents in the buffer and the broadcasting position of relevant channels.

When the player accepts a *Fast Forward/Fast Backward/Slow Backward/Play Backward* command, it checks whether the contents to be played are in the buffer and whether the destination point is a feasible point at the finishing time. When a command passes these two tests, it can be implemented smoothly. Otherwise we simply jump to the nearest feasible point with respect to the specified destination.

When the player accepts a *Jump Forward/Backward* command, it checks whether the destination is a feasible point. If it is, it moves to that point and resumes normal playback. Otherwise, the player selects a nearest feasible point with respect to the destination and resumes normal playback.

If the player accepts a *Pause or Slow Forward* command, it will implement them for the duration specified. This is because *Pause* or *Slow Forward* can be implemented smoothly from any feasible point. So for *Pause*, the player stops moving forward for a specified period of time. For *Slow Forward*, it plays back the video from the buffer at a specified lower speed.

After a VCR action is completed that has a destination outside of the buffer, the player checks the allocation of loaders/buffers and reallocates them, if necessary. Assume the segment containing the current play point is k. During the pyramid phase, if three segments $k, k+1, k+2$ are in the buffer or being downloaded, no actions are taken. Otherwise the player will reallocate loaders to these segments. During the equal segment phase, if segments $k-1, k, k+1$ are in the buffer or being downloaded, no actions are taken. Otherwise the player will reallocate loaders to these segments. The downloading begins immediately after the loaders are allocated to them.

Loader/buffer manager It allocates loaders/buffers at the channel boundary. By channel boundary we mean the point of time when the channel finishes one cycle of broadcasting the segment and begins the next cycle of broadcasting.

During the pyramid phase, if any of the segments k+1,k+2,k+3 are not in the buffer and not being downloaded, we allocate loaders for them. Startup allocation is considered as a special case. When a client starts, we assume the current segment is 0. After the startup latency, we move to the boundary of the first channel. The manager allocates loaders to segments $k+1, k+2, k+3$, i.e., 1,2,3. Downloading of the first segment begins immediately. Downloading of the second and the third segments may begin later, depending on channel positions. During the equal segment phase, if the play point is in the earlier half of the current segment k, the three loaders are assigned to segments $k-1, k, k+1$. If the play point is in the later half of the current segment k, the three loaders are assigned to segments $k, k+1, k+2$. If the contents of a segment are in the buffer, no loading will occur. The downloaded segments will overwrite the earlier segments already in the buffer. However, if a VCR function is ongoing, all the buffer contents for the action cannot be overwritten. If a loader is downloading a segment when we want to allocate a loader to the segment, no action is taken and we assume one loader is already allocated to the segment.

Our scheme has the flexibility for clients with more than three loaders and buffers. In this case, extra loaders/buffers are allocated to unallocated segments based on the priorities of these segments. We define the priority of a segment as its distance to the current segment. Smaller number means higher priority. If two segments have same priority number, the segment in the forward direction is given a higher priority.

4 Feasible Points

We introduce the concept of feasible points to deal with problems that occur in providing VCR functions in partitioned video broadcast. If we do not check the property of the destination point of a VCR action, we may face discontinuity of playback after we resume normal play following the VCR action, even if no further VCR actions are issued. We first describe some notations before we illustrate the problem by another example and give the rules for deciding whether a destination point is feasible or not.

Let us assume the start position and end position of segment i in a video (e.g., time from video starting point) are s_i and e_i, respectively. At any time, each channel is broadcasting a specific frame in the segment. It is called the channel point denoted as c_i. We have $s_i \leq c_i \leq e_i$ and $e_i = s_{i+1}$. We use $[y_1, y_2]$ to represent the part of video between two points y_1 and y_2. Assume the destination point d is in segment j. Note that d can be different from c_j.

4.1 Another Infeasible Case

In Section 3.3, we showed that an unrestricted jump can lead to discontinuity of later playback, even if there are no further VCR actions. In this section, we presents another example in which the video contents required by a VCR action are in the buffer, but performing it leads to the discontinuity in the future after we resume normal play.

In Figure 3, we assume the current play point is s_k and the *fast forward* command is specified as $(t, [s_k, e_k])$. We know that channel broadcast point is c_i for channel i. Assume the contents of channel k ($[s_k, e_k]$) are in the buffer and contents of channel $k+1$ are *not* in the buffer. Because all the contents ($[s_k, e_k]$) required by the VCR action are in the buffer, we can perform the *FF* action to the full length to the position e_k. It takes time t for this fast forward action. During this period, we can only download $[c_{k+1}, h]$ to fill in the buffer, but we need $[s_{k+1}, c_{k+1}]$ to continue the normal play, which is not available.

Therefore, we show a case in which some VCR action cannot be performed to the full length for the continuity reason, even if the contents required by the VCR action are in the buffer.

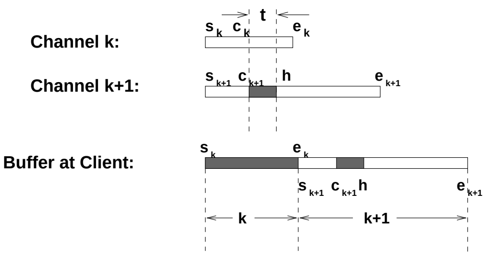


Fig. 3. Another infeasible case

4.2 Rules

We now give the rules for deciding whether a given point is feasible or not, and how to find the nearest feasible point with respect to a given destination. We distinguish three cases based on the relative sizes of segments involved.

If the destination point $d(= p + \Delta\ell)$ is in segment j and d is located at or before the channel point c_j (i.e., $d \leq c_j$), the segment containing d is designated as the current segment. We use k to denote the current segment number and have $k = j$. We now give the rules for these destination points. Later we will consider the rules for destination point d located after the channel point c_j.

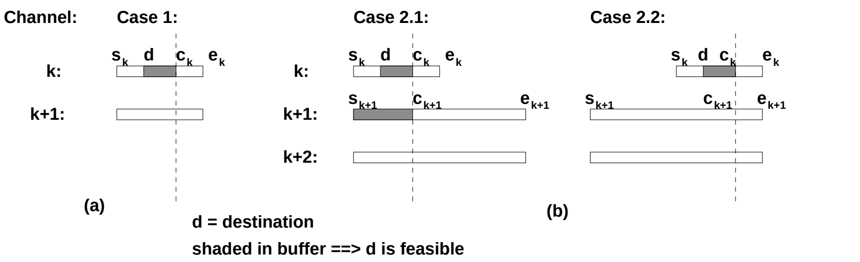


Fig. 4. Rule for feasible points

- Case 1 (Figure 4(a)): (A,A) case. This is the case in which the size of current segment is equal to the size of the next segment, i.e., $f(k) = f(k+1)$.
  - If $[d, c_k]$ is in the buffer, then d is feasible; otherwise, the later nearest feasible point is the point q such that $[q, c_k]$ are in the buffer with the smallest q value.
- Case 2 (Figure 4(b)): $(A, 2A, 2A)$ case. This is the case in which the size of current segment is half of the size of the next segment and the next two segments are of equal size, i.e., $f(k) = 1/2 * f(k+1)$ and $f(k+1) = f(k+2)$.
  - case 2.1 The segments in channels k and $k+1$ are left aligned, i.e., $c_k - s_k = c_{k+1} - s_{k+1}$.
    If $[d, c_k]$ and $[s_{k+1}, c_{k+1}]$ are in the buffer, then d is feasible; otherwise, if $[s_{k+1}, c_{k+1}]$ are in the buffer, the later nearest feasible point is the point

q such that $[q, c_k]$ are in the buffer with the smallest q value; otherwise, the later nearest feasible point is the point q such that $[q, c_{k+1}]$ are in the buffer with the smallest q value;

- case 2.2 The segments in channels k and $k+1$ are right aligned, i.e., $c_k - s_k \neq c_{k+1} - s_{k+1}$.
  If $[d, c_k]$ are in the buffer, then d is feasible; otherwise, the later nearest feasible point is the point q such that $[q, c_k]$ are in the buffer with the smallest q value.

– Case 3: $(A, 2A, 4A)$ case. This is the case in which the size of current segment is half of the size of the next segment and the size of next segment is in turn half of the size of its next segment, i.e., $f(k) = 1/2 * f(k+1)$ and $f(k+1) = 1/2 * f(k+2)$. We distinguish four subcases based on the relative positions of relavant broadcasting channels and give rules for each case. Interested readers are referred to [13].

Now we consider the case in which the destination point d in segment j is located after the channel point c_j (i.e., $d > c_j$). If we designate segment $j+1$ as the current segment, we can use the above rules. If we still use k to denote the current segment number, then we have $k = j+1$. In this situation, $[d, c_k]$ means $[d, e_{k-1}]$ and $[s_k, c_k]$.

5 Simulation Results

5.1 Experimental Settings

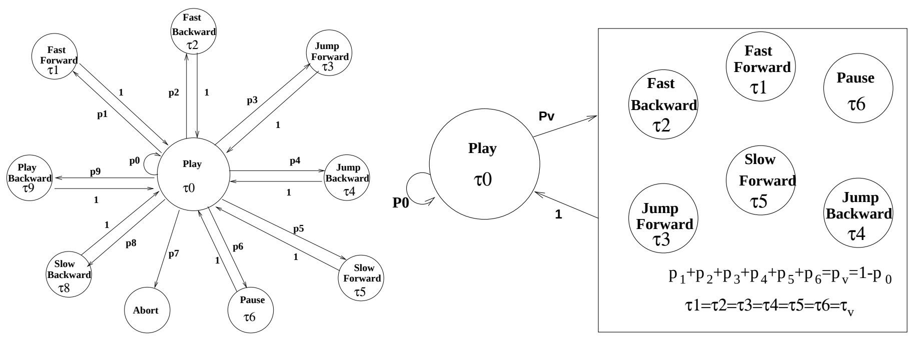
In our experiments, we assume the video length is 120 minutes and we have 30 channels for broadcasting the video. The video is divided into 30 segments and first 8 segments are of unequal size with the weights defined by the series in Section 3.3. Segments from 9 to 30 are of equal size. The size of the first segment is 0.0805 minutes or 4.83 seconds. So maximum startup latency is 4.83 seconds and average is half of it, i.e., 2.42 seconds. The size of largest segment is 5.15 minutes. The size of buffer required at the client side is a multiple (at least 3) of the size of the largest segment.

5.2 User Interaction Model

The user interaction model is shown in Figure 5(a), which gives the probability of issuing a specific VCR action and the mean duration [7]. We assume the duration of a VCR action is exponentially distributed with the mean τ_i given in the user interaction model. The durations τ_i and transition probabilities p_i determine the degree of interactivity of user actions. A similar model is also used in [11]. Finding typical values for these parameters is beyond the scope of this paper. However, rather than selecting a specific set of values for these parameters, we experiment with a range of values.

The model is further simplified to reduce the number of experimental parameters. We set $p_7 = 0$ to prevent unexpected leave from the middle of a video. We

[7] For *Jump*, the mean duration means average jump steps measured as $\frac{|\Delta \ell|}{b}$ since $\Delta t = 0$.



(a) (b)

Fig. 5. User interaction model

set $p_8 = p_9 = 0$ because *slow backward* and *play backward* are not used so often as other VCR functions. Also we set $p_i = p_j$ and $\tau_i = \tau_j$ for $1 \le i < j \le 6$. We use p_v to represent the probability of issuing VCR actions after a *play* action and get $p_v = \sum_{i=1}^{6} p_i$ and $p_0 = 1 - p_v$. We use τ_v to represent those τ_i's $(1 \le i \le 6)$. Therefore, $p_i = p_v/6$ and $\tau_i = \tau_v$ for $1 \le i \le 6$. After the simplification, we get a user interaction model as shown in Figure 5(b). In this model, we have only three parameters p_v, τ_0 and τ_v. We call $\frac{\tau_v}{\tau_0}$ *duration ratio*.

We experiment with two ways to change the user interaction level.

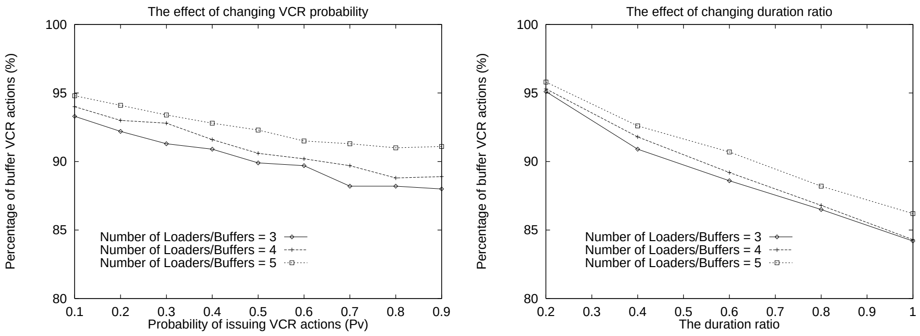
1. Change probability. In this case, we let $\frac{\tau_v}{\tau_0} = 0.5$ and vary p_v from 0.1 to 0.9.
2. Change duration ratio $\frac{\tau_v}{\tau_0}$. In this case, we set $p_v = 0.6$ and change the duration ratio $\frac{\tau_v}{\tau_0}$ from 0.2 to 1.0.

5.3 Numerical Results

Recall that we measure the performance of our scheme by the percentage of buffer VCR actions and the percentage of destination shift (Section 3.2).

In our first experiment (Figure 6(a)), we examine the effect of frequency of issuing VCR actions on the percentage of buffer VCR actions. We change the probability of issuing VCR actions (p_v) from 0.1 to 0.9. We set τ_0 to be about half of maximum segment size (2.58 minutes). We experiment with different number of loaders and buffers at the clients. As expected, the percentage of buffer VCR actions decreases as the probability of issuing VCR actions increases. However, the percentage of buffer VCR actions stays above 87% even if the probability of issuing VCR actions is as high as 0.9. Though this high interactive degree rarely happens, the simulation results in this setting make us believe our scheme can deal with a very high level of user interactions.

We change the duration ratio in Figure 6(b). We let it change from 0.2 to 1.0. The percentage of buffer VCR actions decreases when the ratio increases. The degradation of performance is larger than the case in which we change probability. However, even when the ratio is 1.0 we still only need 3 loaders/buffers



(a) Change probability (b) Change the duration ratio

Fig. 6. Percentage of buffer VCR actions

to keep the percentage of buffer VCR actions above 84%. In most practical situations, the ratio is usually low. We can achieve 95% buffer VCR actions when the ratio is 0.2.

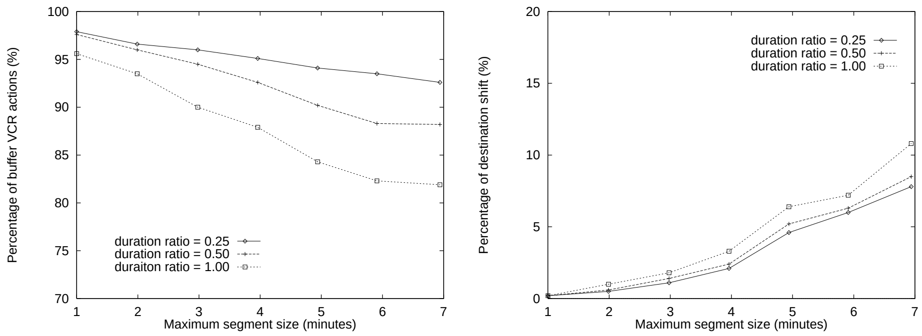
Number of channels	127	67	47	37	31	27	24
Maximum segment size (min)	1.0	2.0	3.0	4.0	4.9	5.9	6.9

Table 1. Relations of the number of channels and maximum segment size

Next we explore the effects of the maximum segment size. We set $p_v = 0.6$ and $\tau_0 = 2.58$ minutes. We let the duration ratio be 0.25, 0.50 and 1.0. We vary the number of the channels so that the size of maximum segment changes from 1.0 minute to 6.9 minutes. The actual numbers are listed in Table 1. The client has three loaders. When the size of the maximum segment is 6.9 minutes, each client needs three 6.9 minute buffers. So the total buffer size in this case is 20.7 minutes. We let the clients in the cases with a smaller maximum segment size have a buffer of the same size, so they may have more than 3 buffers of the maximum segment size.

Figure 7(a) shows the percentage of buffer VCR actions when the maximum segment size changes from 1 minute to 6.9 minutes. As the maximum segment size increases, the percentage of buffer VCR actions decreases. However, even when the segment size is 7 minutes, the percentage of buffer VCR actions is still above 82%. We can see from the plot that we can achieve better performance by increasing the number of channels and thus decreasing the maximum segment size. When the buffer size is 2 minutes, the percentage of buffer VCR actions is 93% for the duration ratio of 1.0.

Next we measure the percentage of destination shift for those adjusted VCR actions. Figure 7(b) shows the percentage of destination shift. In all cases, the percentages of shift are always less than 11%. With the total buffer size being the same, the client can achieve smaller shift if the segment size is smaller. This



(a) Percentage of buffer VCR actions　　(b) Percentage of destination shift

Fig. 7. The effects of maximum segment size

implies that the increase of the number of channels not only reduces the startup latency, but leads to better performance for VCR functions as well.

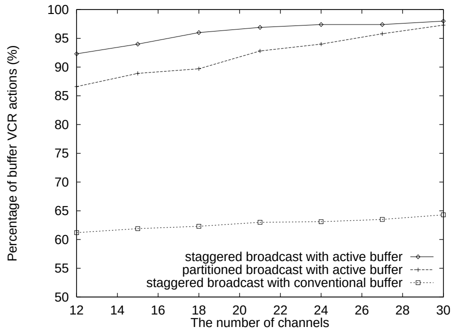
5.4 Comparison with Staggered Broadcast

In this section, we compare the performance in three cases:

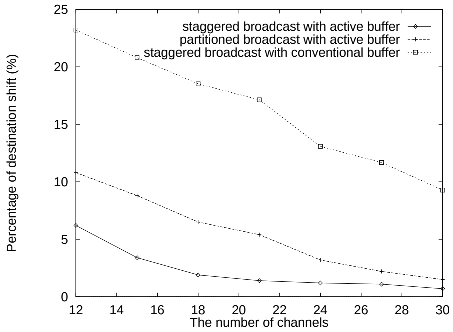
1) staggered broadcast with conventional buffer scheme;
2) partitioned broadcast with active buffer management scheme;
3) staggered broadcast with active buffer management scheme.

We let the number of channels change from 12 to 30 and the number of loaders be 3. We let the case of staggered broadcast with 12 channels have 4 buffers of its maximum segments. For fairness, all other cases have a buffer of the same size as in this case. Thus each of these cases may have a different number (other than 4) buffers of its maximum segment size because its maximum segment may be smaller or bigger than that of the above-mentioned 12 channel case. For the partitioned broadcast cases, we can select an appropriate u value to guarantee that they have at least 3 buffers of their maximum segment size. We keep the user interaction level the same ($p_v = 0.6$ and the duration ratio 0.5). We know the startup latency decreases as the number of channels increases in both staggered broadcast and partitioned broadcast. The startup latency is much smaller in partitioned broadcast than in staggered broadcast with the same number of channels.

We are more interested in their interactive behavior. Figure 8(a) shows the percentage of buffer VCR actions in three cases. The percentage is almost constantly around 63% in the staggered broadcast with conventional buffer scheme, while the partitioned broadcast with active buffer management can achieve 86% with 12 channels and 97% with 30 channels. It is interesting to note that the staggered broadcast equipped with our active buffer management scheme has even better performance than the partitioned broadcast. This is because the maximum segment size of the staggered broadcast is smaller than that of partitioned broadcast, and thus it has a larger number of buffers of its maximum segment size and has more flexibility to arrange the contents in the buffer. The same



(a) Percentage of buffer VCR actions



(b) Percentage of destination shift

Fig. 8. Comparisons with staggered broadcast

performance improvement can also be seen from the measure of the percentage of the destination shift, as shown in Figure 8(b). The percentage in staggered broadcast with conventional buffer scheme changes from 23% to 9%. The partitioned broadcast has a smaller number, varying from 11% to 1.5%. The staggered broadcast with active buffer management enjoys a further improvement with the percentage decreasing from 6% to less than 1%. After comparing its performance with the conventional buffer management scheme, we can clearly see the advantage of our active buffer management scheme.

With the same number of channels, the staggered broadcast has better interactive performance than the partitioned broadcast, if both of them use active buffer management. On the other hand, we know the partitioned broadcast has a much shorter startup latency. An interesting observation here is the tradeoff between the startup latency and the interactive performance between these two schemes. The scheme with a larger startup latency has better performance for providing interactive functions.

6 Concluding Remarks

Multicast delivery is an attractive approach to the provision of a video-on-demand service because it scales well to a very large number of clients. The problem is how to provide interactive functions to individual clients within the multicast framework without compromising the scalability of the multicast paradigm.

In this paper, we propose a scheme to support interactive functions in partitioned video broadcast. The scheme lets the client selectively prefetch segments from broadcast channels based on the observation of the play point position in its local buffer. The contents of the buffer are adjusted in such a way that the relative position of the play point is kept in the middle part of the buffer and a high probability of providing interactive functions with the contents of the local buffer is achieved.

After analyzing the problems with providing interactive functions in the existing partitioned video broadcast schemes, we design a new broadcast series suitable for interactive behaviors of clients. We illustrate several cases in which VCR actions can lead to discontinuity of later playback in partitioned broadcast. We introduce the concept of feasible points to deal with these problems arising from unequal segmentation in the broadcast scheme. By restricting the destination of a VCR action to feasible points, we can guarantee the video can be played back without interruptions after the VCR action.

Using a simple user interaction model we experiment with various levels of user interactivity and explore the performance of our scheme. By using the percentage of buffer VCR actions and the percentage of destination shift as the performance measure, we show that our scheme can implement interactive actions through buffering with a very high probability in a wide range of user interaction levels. Compared with the conventional buffering scheme, our active buffer management scheme is shown to be able to improve the performance of interactive behavior with the same resource requirement.

References

1. A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. of ACM Multimedia'94*, pp. 15–23, 1994.
2. A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *Multimedia Systems*, vol. 3, pp. 112–121, June 1996.
3. K. C. Almeroth and M. H. Ammar, "A scalable interactive video-on-demand service using multicast communication," in *Proceedings of International Conference on Computer Communication and Networks*, pp. 292–301, 1994. San Francisco, CA.
4. K. C. Almeroth and M. H. Ammar, "On the performance of a multicast delivery video-on-demand service with discontinuous VCR actions," in *Proceedings of ICC'95*, pp. 292–301, 1995. Seattle, WA.
5. K. C. Almeroth and M. H. Ammar, "On the use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE Journal of Selected Areas in Communications*, vol. 14, pp. 1110–1122, August 1996.
6. M. Hofmann, E. Ng, K. Guo, S. Paul, and H. Zhang, "Caching techniques for streaming multimedia over the Internet," tech. rep., Bell Laboratories, April 1999. Technical Report 990409-04TM.
7. S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Systems*, vol. 3, pp. 197–208, May 1996.
8. K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proceedings of ACM Sigcomm'97*, 1997.
9. L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," in *Proceedings of NOSSDAV'98*, 1998.
10. W. Liao and V. O. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, vol. 4, pp. 51–62, October-December 1997. Also in Proceedings of Infocom'97.
11. E. L. Abram-Profeta and K. G. Shin, "Providing unrestricted VCR functions in multicast video-on-demand servers," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS'98), Austin, Texas*, 1998.
12. Z. Fei, I. Kamel, S. Mukherjee, and M. H. Ammar, "Providing interactive functions for staggered multicast near video-on-demand systems," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems '99*, 1999.
13. Z. Fei, M. Ammar, I. Kamel, and S. Mukherjee, "Providing interactive functions through active client buffer management in partitioned video broadcast," tech. rep., College of Computing, Georgia Institute of Technology, 1999. GIT-CC-99-09.