Providing Interactive Functions for Staggered Multicast Near Video-On-Demand Systems

(Extended Abstract)

Zongming Fei † * Ibrahim Kamel ‡ Sarit Mukherjee ‡ Mostafa H. Ammar †*

† *Networking and Telecommunications Group*
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
U.S.A.
{fei,ammar}@cc.gatech.edu

‡ *Panasonic Information and Networking Technology Laboratory*
Panasonic Technologies, Inc.
Two Research Way
Princeton, NJ 08540
U.S.A.
{ibrahim,sarit}@research.panasonic.com

Abstract

In this paper we propose a scheme to provide interactive functions through active buffer management at the client side for staggered multicast near VOD systems. The scheme uses client side buffering in a novel fashion that relies on the simultaneous availability of "past", "present" and "future" parts of a video and lets the client selectively prefetch segments from multicast channels based on the observation of the play point 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 the interactive functions with the contents of the local buffer is achieved. We perform several simulations and find that the active buffer management is an effective method to provide interactive functions in these systems.

1 Introduction

A Video-On-Demand (VOD) service provides subscribers with a set of videos and sends a specific video to customers upon request. In unicast VOD systems customers are serviced individually by allocating and dedicating a transmission channel and a set of server resources to each customer. This leads to an expensive-to-operate, non-scalable system. It has been recognized that most of the requests are for a small group of videos and the popularities of the videos follow the Zipf distribution [1, 2]. Using multicast to send these videos has been demonstrated by many researchers to be an efficient way for delivery.

There are two basic approaches to provide multicast video-on-demand services. One is on-demand batching and the other is continuous broadcast. In conventional on-demand batching multicast VOD schemes [1, 2, 3, 4, 5] the server allocates channels to one client or a group of clients and sends a video over that channel. A group of pending clients requesting the same video is called a batch. There is a waiting time (called *startup latency*) before the clients can be serviced.

Continuous broadcast VOD systems change the allocation strategy of the multicast channels. In these systems one or several channels are allocated to one video and each channel sends the whole video or a part of it in cycles. These systems do not need upstream channels and are most suitable for broadcasting popular videos. In the staggered multicast (the pay-per-view model) several channels broadcast a video periodically with staggered start times. In such a system the maximum startup latency equals the length of the video divided by the number of the channels allocated to this video. Some recently proposed schemes [6, 7] improves the startup latency by dividing a video into segments of unequal sizes and letting each channel broadcast one segment. We focus on how to provide interactive functions in staggered multicast near VOD systems. We relegate consideration of how to provide interactive functions for segmented video broadcast systems to future work.

*The authors work is supported in part by a research grant from NSF under contract number NCR-9628379.

Several schemes [3, 4, 5, 8, 9] have been proposed to deal with the problem of providing interactive functions in multicast near VOD systems. However, most of them have addressed interactive functions (sometimes called 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 continuous broadcasting "pay-per-view" model of video delivery, but interactive channels are not available in this model. 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 a scheme for providing VCR functions in staggered multicast near VOD systems. The scheme uses client side buffering in a novel fashion that relies on the simultaneous availability of "past", "present" and "future" parts of a video and 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]. Our simulations show that our scheme can achieve continuous VCR actions with a high probability in a wide range of user interaction levels.

2 Conventional Buffer Scheme for Providing VCR Functions

Staggered multicast near VOD can be described as follows. Assume that there are K channels dedicated to a video of length L. The K channels are labeled as $0, 1, 2, \cdots, K-1$. Each channel repeatedly multicasts the whole video as shown in Figure 1. Assume channel 0 begins sending the video at time t_0. Channel k begins at s time units later than channel $k-1$, where $s = L/K$.

We consider following VCR functions in this paper. 1)*Jump Forward/Backward*: Immediate jump to the destination, specified in terms of forward/backward distance relative to the current play position. 2)*Pause*: Stop the playback for a specified duration of time. 3)*Fast Forward/Backward*: Playback the video at a faster speed than the normal playback rate in forward or backward direction. 4)*Slow Forward*: Playback the video at a slower speed than the normal playback rate.

[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.

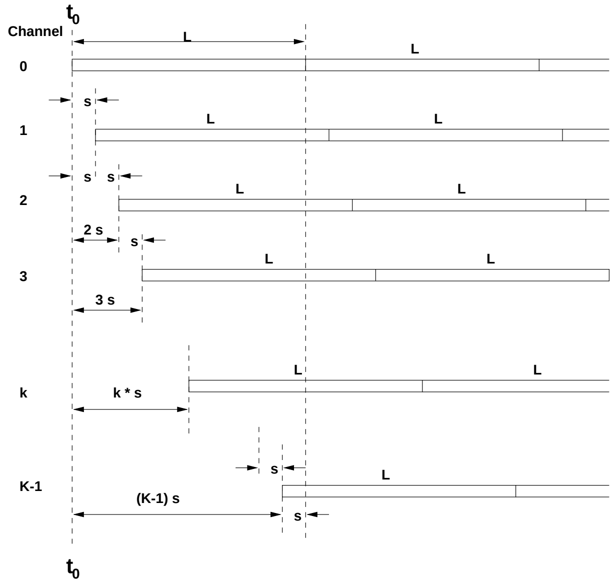

FIGURE 1: **Staggered Multicast**

Using client buffer is an intuitive solution to the problem of providing VCR functions. It has been discussed in several video multicast schemes [3, 4, 5, 8, 9]. The problem with the 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 the 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 the 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 provides VCR functions with a greater flexibility.

3 Providing VCR Functions with Active Buffer Management Scheme

3.1 The Idea of Active Buffer Management

The aim is to design a novel buffer scheme to implement VCR functions within the staggered multicast VOD framework. 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 challenge 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 [2]. We will refer to a video part from $k*s$ to $(k+1)*s$ as segment k in our description below, where s is the staggered interval. We employ a policy of selectively downloading the segments based on the current position of play point in the buffer. This is possible in staggered video multicast because 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 scenario. We 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. If there is no VCR function, the 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, i.e., segment $z+2$. Now consider a case when a *fast forward* action is issued during the period and the play point is moved to within segment $z+3$. We observe the play point is no longer in the middle segment $z+2$. The idea is to let the buffer manager select the segments to be downloaded according to the position of the play point. After observing that the play point is in the last segment, the buffer manager will download segments $z+4$ and $z+5$ 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.

Our scheme can tolerate consecutive VCR actions in the same direction because the effects of any VCR action on the relative position of play point in the buffer is compensated by our buffer management scheme.

[2] 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.

3.2 VCR Function Implementation in Staggered Multicast Near VOD systems

Assume that there are K channels $(0, 1, 2, \cdots, K-1)$ dedicated to a video of length L. Assume that channel 0 begins sending the video at time t_0 and channel k begins at $s = L/K$ time units later than channel $k-1$. We know channel k should begin at time $t_0 + k*s$. We further assume that a client should be able to get the parameters k, L, t_0 and the client has a buffer of size $3*s$.

Based on the notation that segment k in a video represents the video part from $k*s$ to $(k+1)*s$, we can derive several functions that will be used in our description. The segment sent by channel k at time t is

$$seg(t,k) = \lfloor \frac{[t-(t_0+k*s)] \bmod L}{s} \rfloor$$

and the offset of the broadcasting point in the segment is

$$offset(t,k) = [t-(t_0+k*s)] \bmod s = (t-t_0) \bmod s,$$

which is independent of channel number k and can thus be abbreviated as

$$offset(t) = (t-t_0) \bmod s.$$

The play point p in a video should be in segment

$$playseg(p) = \lfloor p/s \rfloor$$

with offset

$$playoffset(p) = p - playseg(p) * s.$$

At time t, segment j should be downloaded from channel [3]

$$channel(t,j) = (\lfloor (t-t_0)/s \rfloor - j) \bmod K.$$

Two concepts will be used in our description of providing VCR functions through client buffering. One is *next channel position* of a play point $p (0 \le p \le L)$ at time t. We know at time t channel k is sending segment $seg(t,k)$ and the offset in the segment is $offset(t)$. If $playoffset(p) \le offset(t)$, the next channel position of p is the position in segment $playseg(p)$ with offset $offset(t)$. If $playoffset(p) \ge offset(t)$, the next channel

[3] At time t, channel 0 is sending segment $s_0 = \lfloor (t-t_0)/s \rfloor \bmod K$. Channel k is sending segment $s_k = (s_0 - k) \bmod K$. Let $s_k = j$, we get $(s_0 - k) \bmod K = j$. So $k = (s_0 - j) \bmod K = (\lfloor (t-t_0)/s \rfloor \bmod K - j) \bmod K = (\lfloor (t-t_0)/s \rfloor - j) \bmod K$.

position of p is the position in segment $playseg(p)+1$ with offset $offset(t)$. Another notion is *the nearest channel position* of a play point $p (0 \leq p \leq L)$ at time t. It is the nearest point from p among three candidates, positions with offset $offset(t)$ in segments $playseg(p)-1$, $playseg(p)$ and $playseg(p)+1$.

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

When a client comes at time t, it will begin to download and play at $start\_time = \lceil (t-t_0)/s \rceil * s$. It can download segment 0 from channel $\lceil \frac{(t-t_0) \bmod L}{s} \rceil$. Instead of downloading only from one channel as in the conventional scheme, the client downloads segment 1 from channel $(\lceil \frac{(t-t_0) \bmod L}{s} \rceil - 1) \bmod K$ and segment 2 from channel $(\lceil \frac{(t-t_0) \bmod L}{s} \rceil - 2) \bmod K$ at the same time. By doing this, it prefetches some contents for potential forward VCR actions.

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 multicast position at each channel.

If the player accepts a *Jump Forward/Backward* command, it checks whether the part of the video from the destination to its next channel position is in the buffer. If it is, it moves to that point and resumes normal playback. Otherwise, the player jumps to the nearest channel position with respect to the specified destination.

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 part of the video from the destination point to its next channel position at the finishing time is in the buffer. When a command passes these two tests, it can be implemented smoothly. We call this case a continuous VCR function implementation. Otherwise we simply jump to the nearest channel position with respect to the specified destination at current time.

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. 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 the player checks the allocation of loaders/buffers and reallocates them, if necessary. Assume the current play point is p. If the play point is inside the buffer, no actions are taken. The relative position of play point will be adjusted later by the buffer manager. If the play point goes outside of the buffer, three loaders will be adjusted to download segment $playseg(p)-1, playseg(p)$ and $playseg(p)+1$ if the whole segment is not in the buffer. The downloading begins immediately after the loaders are allocated to these segments.

The buffer manager will adjust the channels from which to download segments every s time interval from *start_time*, or at time $start\_time + i * s$, where $i = 1, 2, \cdots$. Assume the play point is p, it should be in the segment $playseg(p)$ with offset $playoffset(p)$. If $playoffset(p) < s/2$, the buffer manager will allocate three loaders to download segments $playseg(p)-1, playseg(p)$ and $playseg(p)+1$ if the whole segment is not in the buffer. If $playoffset(p) \geq s/2$, the buffer manager will allocate three loaders to download segments $playseg(p), playseg(p)+1$ and $playseg(p)+2$ if the whole segment is not in the buffer. At selection time t, segment j should be downloaded from channel $channel(t,j)$.

The buffer management scheme can be extended to deal with the case in which the clients have more than 3 loaders and buffers. If there are more than three loaders and buffers, extra loaders/buffers are allocated to a candidate segment based on its priority defined by its distance to the current segment. Smaller number means higher priority. If two segments have the same priority number the segment in the forward direction is given a higher priority. Clients can have better performance with more loaders and buffers.

4 Simulation Results

We measure the performance of our scheme by the percentage of the continuous VCR actions and the percentage of the destination shift.

We compare the performance of providing interactive functions with conventional buffer scheme and our active buffer management scheme. In our experiments, we assume the video length is 120 minutes and let the number of channels change from 12 to 30. The buffer used for all cases are the same and the number of loaders are always 3. We keep the user interaction level the same in the experiment. Figure 2 shows the percentage of continuous VCR actions in two schemes. The percentage is almost constantly around 63% in the conventional buffer scheme, while it can achieve

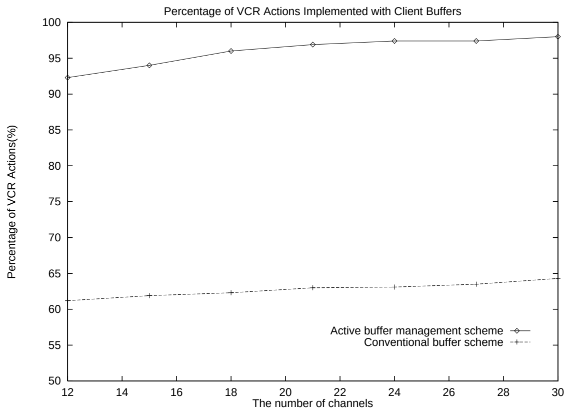

FIGURE 2: **Percentage of continuous VCR actions**

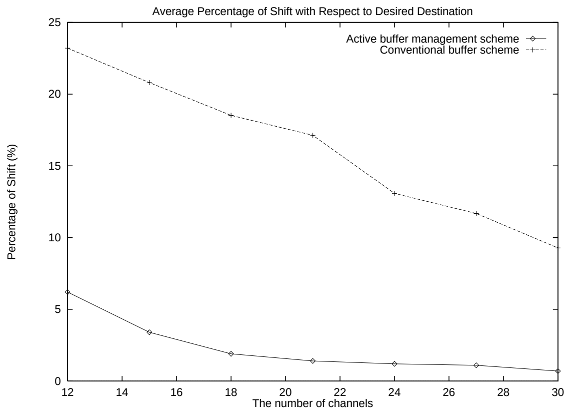

FIGURE 3: **Percentage of destination shift**

higher than 92% with our active buffer management scheme. As expected, the performance improves as the number of channels increases. The percentage of continuous VCR actions is as high as 98% when the number of channels is 30.

The same performance improvement can also be seen from the plot of the percentage of the destination shift, as shown in Figure 3. The percentage in staggered multicast with conventional buffer scheme changes from 23% to 9%. The staggered multicast with our active buffer management scheme enjoys an improvement with the percentage decreasing from 6% to less than 1%. From this we can clearly see the advantage of the proposed active buffer management scheme.

References

[1] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proceedings 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] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Systems*, vol. 3, pp. 197–208, May 1996.

[7] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," in *Proceedings of NOSSDAV'98*, 1998.

[8] 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.

[9] 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.