Multimedia Tools and Applications 6, 61–83 (1998)
© 1998 Kluwer Academic Publishers. Manufactured in The Netherlands.

Dynamic Buffer Management for Near Video-On-Demand Systems

WEN-JIIN TSAI AND SUH-YIN LEE wjtsai@info1.csie.nctu.edu.tw
Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

Abstract. Advances in networking and storage technology have made it possible to deliver on-demand services over networks such as the emerging video-on-demand (VOD) applications. A variety of studies have been focused on designing a video server suitable for VOD applications. However, the number of concurrent on-demand services supported by the server is often limited by the I/O bandwidth of the storage systems. This paper describes a *discrete buffer sharing model* which uses batching and buffer sharing techniques in video servers to support a large number of VOD services. Two operations, splitting and merging, enable the model to fully utilize system resources such as buffers and disk bandwidths. Moreover, this paper also introduces the concept of *imprecise video viewing* which assumes that a limited amount of quality loss is acceptable during video playback. Based upon this assumption, three shrinking strategies are explored to reduce buffer requirements. Finally, the results of experiments show that our methods perform better than traditional buffer management techniques for VOD systems.

Keywords: video-on-demand, buffer management, buffer sharing, scheduling

1. Introduction

Advances in networking and storage technology have made it possible to deliver on-demand services, such as catalog shopping, distance learning, and general information browsing [4, 5], over networks. Recently, emerging video-on-demand (VOD) applications have received enormous attention from telecommunications, entertainment, and computer industries. A variety of studies have focused on the design of high-performance VOD systems capable of handling large numbers of services simultaneously [7, 9]. A typical VOD system consists of a *video archive* and a set of *video servers* [10, 15]. The video archive maintains a collection of all available video files and the video servers maintain a small set of frequently requested videos. This paper focuses on the design of a video server suitable for VOD applications.

The characteristics of digital video files and video traffic differ substantially from those of conventional applications with regard to *continuity* and *high bandwidth* requirements. Continuity means that client stations must acquire the needed video data on time, namely, the components of a VOD system, including the server storage and the interconnection network, must support enough bandwidths to accommodate all the video streams so that they can be continuously displayed. Due to finite I/O bandwidths in the storage and the network, the number of concurrent on-demand services supported by the VOD system is often limited. The bottleneck in network bandwidth can be reduced by employing multicasting techniques which is available in most current networks. If a single video stream is being accessed by multiple viewers, the server retrieves the data exactly once from the server buffer and multi-casts it to all the viewers. This technique enables networks to support more services for more users. To cope with the I/O bottleneck in storage, a variety of studies have been focused on different techniques of data placement and scheduling policy in the storage system [1, 12, 13, 17]. However, the I/O problem has not been solved yet.

Recently, the studies of solving I/O problems tend to reduce I/O demands on the video server through *batching*, *adaptive piggy-backing*, and *buffer sharing*. *Batching* delays the start time of video playback in order to service multiple viewers by using a single I/O stream [2]. *Adaptive piggy-backing* adjusts display rates of the videos in progress, for the purpose of merging their respective I/O streams into a single stream [8]. *Buffer sharing* techniques reduce I/O demands by preserving the recently used data in memory such that it can be reused by subsequent viewers [3, 14, 16]. Although these approaches may incur problems when providing VCR functionality, several approaches have been proposed. Dan et al. solved the problem in the context of batching by using *reserved channels* [2], while Yu et al. solved the similar problem by using *look-ahead scheduling* [18].

This paper describes a *discrete buffer sharing model* which uses batching and buffer sharing techniques in VOD systems to support a large number of services. Two operations, splitting and merging, used in the model enable a video server to fully utilize system resources such as buffers and disk bandwidths. This paper also introduces the concept of *imprecise video viewing* which assumes that a limited amount of quality loss is acceptable during the playback of a video. The quality loss can be resulting from inserting advertisements or skipping some video contents during the playback. Based on this assumption, three shrinking strategies which include *backward shrinking*, *forward shrinking* and *two-way shrinking*, are explored to reduce buffer requirements in the system. The idea behind the shrinking strategies is similar to that used in adaptive piggy-backing. Finally, the results of experiments show that our methods perform better than traditional buffer management techniques for VOD systems.

2. Buffer management for continuous video data

For a video server, buffers are used as an intermediate to cache and deliver video data from the storage to the network interface. In order to support continuous playback, the technique of *double-buffering* is often employed in a typical video server for buffer management. With double-buffering, each video stream requires two buffers: a *consuming buffer* and a *producing buffer*. A consuming process empties the consuming buffer by transferring video data from the buffer to the network. A producing process fills the producing buffer with video data retrieved from the storage. The two operations are performed in parallel. Whenever the consuming buffer is empty and the producing buffer is full, the two buffers exchange their roles. The process is repeated until the end of the playback. Let a *session* represent the disk bandwidth necessary to support the real-time delivery of a single video. With double-buffering, the maximum number of concurrent streams supported in the system is often limited to the number of sessions offered in the storage subsystem. The following *discrete buffer sharing model* is proposed to accommodate more concurrent streams in the system.

2.1. Discrete buffer sharing model

In discrete buffer sharing model, n buffers ($n \geq 2$) will be reserved whenever a new session is created. Among n buffers, one buffer is used for the producing process and the other $n - 1$ buffers are used for the consuming process, that is, $n - 1$ consuming buffers will share a single producing buffer for data refreshing. Such a technique is called *n-buffering*. With n-buffering, each of the n allocated buffers serves, in turn, as the producing buffer in a different cycle. For example, if the kth buffer acts as the producing buffer in the ith cycle, then the $((k + 1)\ \mathbf{mod}\ n)$th buffer will serve as the producing buffer in the $(i + 1)$th cycle. In this way, the video data in each buffer would remain unchanged for $n - 1$ cycles so that they can be reused by subsequent streams that display the same video. By using such a technique, called *buffer sharing*, multiple streams can be served by a single disk session. We also distinguish *streams* from *viewers* because multiple viewers can participate in a single stream. If a stream is being accessed by multiple viewers, the video data is fetched from the server buffer exactly once and broadcasted to all the viewers by using multi-casting techniques in the network. The relationship between the viewer, stream and session in our model is illustrated in figure 1. Let n_{Viewer}, n_{Stream} and n_{Session} respectively denote the number of viewers, streams, and sessions supported in a VOD system. Figure 1 indicates that we have advantages of $n_{\text{Viewer}} > n_{\text{Stream}}$ by employing multi-casting techniques in the network and $n_{\text{Stream}} > n_{\text{Session}}$ by using buffer sharing techniques in the video server.

We operate buffer sharing in a *gated* manner which means the retrieval of the data in each consuming buffer must be time-aligned and be completed before the end of each cycle. The gated operations prevent starvation in any consuming process such that continuity is guaranteed for each video stream. Figure 2 illustrates the buffer sharing scenario in the proposed model, where six buffers ($b_1, b_2, \ldots, b_6$) are reserved for a single session. In this figure, buffer b_i serves as the producing buffer, represented by shaded rectangles, in the cycle ($i\ \mathbf{mod}\ 6$), and all other buffers serve as consuming buffers which are represented by plain rectangles. For the purpose of gated operations, the viewers that arrive within a cycle must be grouped together (called a *viewer group*) to share a single stream. In figure 2, the viewers that arrive within cycle i form the viewer group V_i. This group will share stream

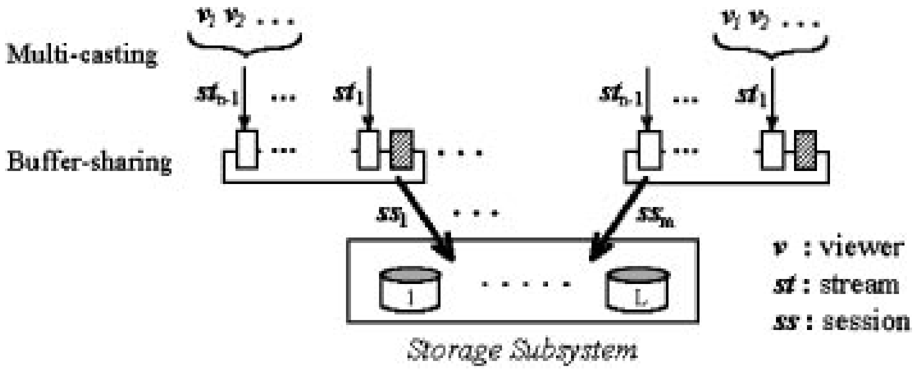

Figure 1. The relation between viewer, stream and session.

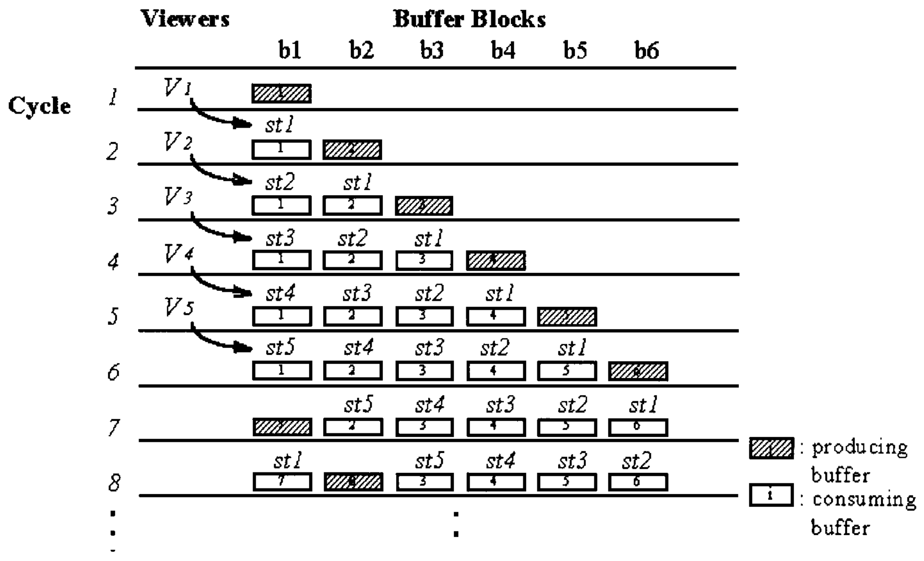

Figure 2. Buffer sharing scenario for a six-buffering system.

st_i and join the session in cycle $i + 1$. If no viewer arrives within cycle i, then st_i will not be created. Each video stream starts from the consuming buffer b_1 and all the streams *synchronously* move to the next buffer at the beginning of each cycle.

With n-buffering, only the viewers that subscribe to the same video within the first $(n - 1)$th cycle can share these n buffers and use the common disk session. If the cycle time is T_{cycle}, then only the viewers who arrive within time interval $[t_c, t_c + (n - 1)T_{\text{cycle}}]$ are admitted to *join* the session created at time t_c. To "join" a session means to share common disk bandwidth for video data delivery. In figure 2, the joining process can last through cycle 5 because the viewers that arrive during cycle 6 must join the session in cycle 7. Unfortunately, in cycle 7, buffer b_1 serves as the producing buffer again and the data in b_1 gets flushed. Thus, at most five streams are allowed to share the session in figure 2. To save buffer resources, if there are no viewers arriving during cycle 4 and cycle 5, and streams st_4 and st_5 are not created, then buffer b_3 (instead of b_1) can serve as the producing buffer in cycle 7 and two buffers b_1 and b_2, can be released.

2.1.1. Admission control. Many types of videos may be archived in a video database in terms of their playback rates [6]. In general, the video server should be capable of offering concurrent sessions to support videos with distinct playback rates. Assuming that a video file is made up of *segments* where each segment (a sequence of video frames) denotes the data required for playback per cycle, then when a common cycle time T_{cycle} is used, the segment size for a video with playback rate r_d is equal to $r_d \times T_{\text{cycle}}$. Choosing the buffer block size for a session means determining the segment size of the video served by this session. However, due to variable bit rates in compression, the segment size in a video file may not be constant. To accommodate the data retrieved by a session per cycle, the buffer block size used by this session must be equal to the maximum segment size. Let b_i denote the buffer block size used by session i, and r_t denote the disk transfer rate (bytes/sec). To ensure continuous playback for k concurrent sessions in the system, the following inequality must be satisfied:

$$\frac{\sum_{i=1}^{k} b_i}{r_t} + k \times t_s^{\max} \le T_{\text{cycle}} \tag{1}$$

where $\frac{\sum_{i=1}^{k} b_i}{r_t}$ means the time for transferring data to fill the k producing buffers, and $t_s^{\max}$ represents the maximal seek and latency times on the disk. Assuming that n_i denotes the number of buffer blocks used by session i, the following inequalities must also hold to meet the buffer requirement:

$$\sum_{i=1}^{k} n_i b_i \le M \tag{2}$$

where $n_i \ge 2$ for $\forall i$ and M is the total buffer size in the system. Note that, inequality (1) corresponds to the constraints imposed by the disk bandwidth and inequality (2) by the available buffer resources in the system. In the following, we shall address how to determine a proper n_i for a session i.

2.2. *Resource adjusting operations*

A simple way for sizing n_i is to give all sessions a fixed, equal number of buffer blocks. Assuming that the server storage is capable of offering n_{session} sessions, the number of buffer blocks assigned to each session is measured as $n_i = \lfloor M/(\sum_{j=1}^{n_{\text{session}}} b_j) \rfloor$ for $\forall i$. This method, called *static buffer sharing*, while simple and easy to implement, wastes system resources as illustrated in figure 3. In this figure, each session is assigned 15 buffers which are represented by rectangles with denoted segment numbers. All the sessions display the same video. In figure 3, nine buffers are unused per cycle in session 2 because viewers subscribe the video sporadically. In session 3, there are eight free buffers because viewers only subscribe during the first six cycles. These two cases indicate buffer is wasted due to fixed buffer assignment. Moreover, the streams which are created nearly of the same time and served by different

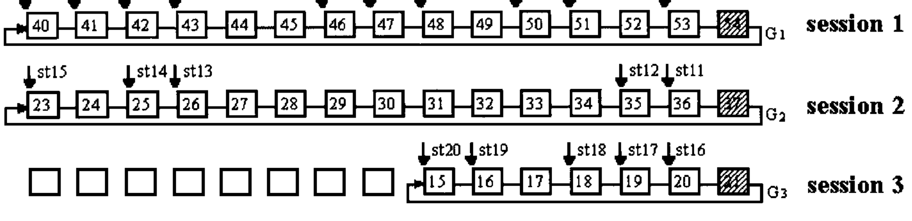

Figure 3. A snapshot of static buffer sharing.

sessions waste disk bandwidths. For example, st_{15} and st_{16} do not share a common session, even though there are only three cycles between them. The inadequate utilization of buffers and disk sessions may prevent the system from supporting more services. To overcome the problem, two operations, *splitting* and *merging*, are explored to dynamically adjust n_i to improve resource utilization.

2.2.1. Splitting. Splitting subdivides a stream group into several smaller groups in order to release the unused buffers. We first define the segment that stream st_i consumes in the current cycle as the *playback offset* of st_i, denoted by $\delta(st_i)$. In figure 3, for example, $\delta(st_1)$ is equal to 53. The set of streams that share a session are called a *stream group*, denoted by G. In figure 3, we can subdivide the stream group $G_2 = \{st_{11}, st_{12}, st_{13}, st_{14}, st_{15}\}$, which is served by session 2, into two subgroups, $G_{21} = \{st_{11}, st_{12}\}$ and $G_{22} = \{st_{13}, st_{14}, st_{15}\}$. Since each subgroup needs a producing buffer for data refreshing, seven buffers (filled with segments 28, 29, ..., 34) can be released. Let G_x denote a stream group and sort all the streams in G_x in an ascending order of their playback offsets, i.e., $\delta(st_i) > \delta(st_j)$ for $i < j$. Let Spare_{G_x} denote the maximal playback distance between any two consecutive streams in G_x, which is defined by $\text{Spare}_{G_x} = \max(|\delta(st_i) - \delta(st_{i+1})| - 1)$ for any $st_i, st_{i+1} \in G_x$. In figure 3, for example, $\text{Spare}_{G_1} = 2$, $\text{Spare}_{G_2} = 8$ and $\text{Spare}_{G_3} = 1$. Note that, to perform a splitting operation on a stream group G_x , at most $\text{Spare}_{G_x} - 1$ buffer blocks can be released, where the “1” refers to the producing buffer. Thus, to release more buffers in splitting, the stream group G_i with larger Spare_{G_x} is preferred. More buffers can be released, of course, if we perform splitting operations recursively on each of the resulting subgroups. However, splitting, while releases buffers, consumes more disk sessions. In figure 3, although seven buffers can be released by splitting session 2, four sessions (instead of three) are required because one additional producing buffer must be filled per cycle. Thus, to split a stream group G_x, inequality (3) must be satisfied to meet disk bandwidth requirements. b_x denotes the buffer block size used for G_x.

$$\frac{\left(\sum_{i=1}^{k} b_i\right) + b_x}{r_t} + (k+1)t_s^{\max} \leq T_{\text{cycle}} \tag{3}$$

2.2.2. Merging. Merging combines multiple stream groups into a single group in order to save disk sessions but can only be performed when these groups display the same video. Let G_x and G_y denote two stream groups of the same video with G_x leading G_y, i.e., all the playback offsets in G_x are greater than that in G_y. Merging G_x and G_y can be achieved by preserving the data of G_x in memory until it can be reused by G_y. Namely, we need another buffers to bridge the gap between G_x and G_y such that continuous playback will not be disturbed after the disk session used by G_y is released. As an example, to merge G_1 and G_2 in figure 3, we need to allocate another two buffers to G_1 to hold the segments 55 and 56 retrieved in the next two cycles. After session 2 has filled its producing buffer with segment 39, G_1 and G_2 are merged and then served by session 1. Afterwards session 2 is released to complete the merging process. Let Gap_{G_x,G_y} denote the minimal playback distance between G_x and G_y, which is defined by $\text{Gap}_{G_x,G_y} = \min(|\delta(st_i) - \delta(st_j)| - 1)$ for $\forall st_i \in G_x$ and $\forall st_j \in G_y$. Merging G_x and G_y needs additional $\text{Gap}_{G_x,G_y} - 1$ buffers to bridge the gap between G_x and G_y, where the “1” refers to the producing buffer of G_y.

The merging algorithm for two stream groups G_x and G_y is described as follows:

1. Allocate $\text{Gap}_{G_x,G_y} - 1$ free buffers to G_x, where each buffer acts in turn as the producing buffer of G_x and holds the segments retrieved in the next $\text{Gap}_{G_x,G_y} - 1$ cycles.
2. At the end of the $(\text{Gap}_{G_x,G_y} - 1)$th cycle, the links in G_x and G_y are updated as follows, where $b^i_{\text{producing}}$ represents the producing buffer of G_i.

$$\begin{aligned}&\text{temp} = b^x_{\text{producing}} \to \text{next}\\&b^x_{\text{producing}} \to \text{next} = b^y_{\text{producing}} \to \text{next}\\&b^y_{\text{producing}} \to \text{next} = \text{temp}\end{aligned}$$

3. The session reserved for G_y is released and the merging process is completed.

To merge stream groups G_x and G_y in the system, inequality (4) must be satisfied to meet the buffer requirements, where b_y stands for the buffer block size used in G_y. Note that, to release a session by sacrificing less buffers in merging, two stream groups with smaller playback distances are preferred.

$$\sum_{i=1}^{k} n_i b_i + \left(\text{Gap}_{G_x,G_y} - 1\right) \le M \tag{4}$$

2.2.3. Admission control with splitting and merging. Both splitting and merging dynamically alter the amount of buffers used for a session to fully utilize buffers and disk bandwidths, such that more services can be provided within the system. Figure 4 shows the benefits yielded from performing both splitting and merging on the stream groups from figure 3. In figure 4, a splitting operation is performed to subdivide G_2 into two subgroups, G_{21} and G_{22}. Then two merging operations are performed on the resulting four groups, G_1, G_{21}, G_{22} and G_3, to combine them into two stream groups, G'_1 and G'_2. As a result as figure 4 shows, only two sessions are required. That is, one session is saved in this case. Moreover, since seven buffers are released through splitting and three buffers are required for merging (two for the merging of G_1 and G_{21} and one for the merging of G_{22} and G_3), four buffers can be released in total. The result in figure 4 reveals that, by using splitting

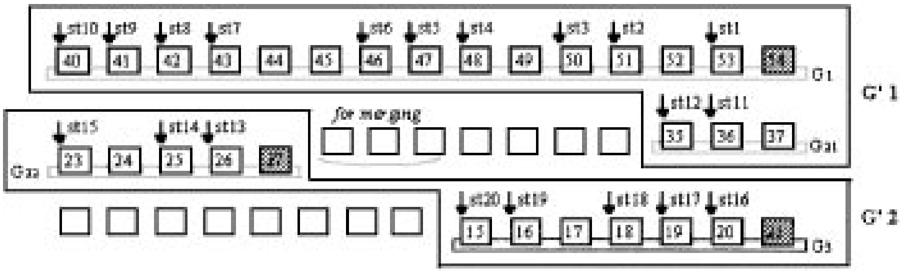

Figure 4. Dynamic buffer sharing with splitting and merging.

and merging, one session and four buffers are saved and these resources can be utilized to create more streams offer more services.

Performing splitting and merging operations involves buffers and sessions trade-offs. Splitting releases buffers by using more disk sessions, while merging saves disk sessions at the cost of consuming more buffers. This means that, it is beneficial to perform splitting when buffer space is insufficient in the system, i.e., inequality (2) is not satisfied for creating a new session. Similarly, it is beneficial to perform merging when disk sessions are insufficient, i.e., inequality (1) is not satisfied for creating a new session. The following admission control algorithm for creating a new session is capable of supporting more services through splitting and merging. It is assumed that $k-1$ sessions are already in progress.

```
 1. Admission Control with Splitting and Merging
 2. Begin
 3.     Case both inequality (1) and (2) hold:
 4.         return session $k$ is admitted;
 5.     Case both inequality (1) and (2) are not satisfied:
 6.         return reject session $k$;
 7.     Case inequality (1) holds, but inequality (2) is not satisfied:
 8.         if ( there is a $G_x$ in the system such that inequality (3) holds )
 9.             perform a splitting on $G_x$;
10.             return session $k$ is admitted;
11.         else
12.             return reject session $k$;
13.     Case inequality (2) holds, but inequality (1) is not satisfied:
14.         if ( there are $G_x$ and $G_y$ in the system so that inequality (4) holds )
15.             perform a merging on $G_x$ and $G_y$;
16.             return session $k$ is admitted;
17.         else
18.             return reject session $k$;
19. End
```

Although splitting and merging operations are performed only upon the arrival of new requests when the system lacks sufficient resources (buffers or disk sessions), they may impose large computation overheads on the system if we need to search the entire system to find the best candidate for performing such operations. One solution to reduce the overheads is to maintain candidate lists for both splitting and merging. The *splitting candidate list*, say S-list, maintains all stream groups in a descending order of their maximal playback distances (*Spare*), and the *merging candidate list*, say M-list, maintains all stream groups in an ascending order of their minimal playback distances from other groups (*Gap*). With S-list and M-list, we can easily find a stream group for splitting or merging, even when there are a large number of sessions and several videos in the system. The computation overhead by using this method is that, each time a new stream group is formed, we need to locate this stream group in both S-list and M-list. The new stream group can be the newly created group or the groups resulting from splitting and merging.

3. Imprecise buffer management

In this section, we further reduce buffer requirements in the discrete buffer sharing model by introducing a concept called *imprecise video viewing* which assumes that a limited amount of quality loss resulting from inserting advertisements (*external data*) or skipping video segments (*optional data*) is acceptable during the playback of a video. Based on this assumption, three *shrinking strategies* were explored in Section 3.2 which dynamically handle data during the video playback to reduce buffer requirements.

3.1. Imprecise viewing model

In the *imprecise viewing model*, each video (V) on the system is characterized by {α-set, β-set, $MaxTotal_\alpha$, $MaxTotal_\beta$, $MinInterval_\alpha$, $MinInterval_\beta$}, where the description of each parameter is given in Table 1. In this table, we refer to α as an *insertable offset* where the insertion of external data is allowed, and refer to β as an *optional offset* where the skipping of optional data is allowed during video playback. Both α and β are called *imprecise offsets*. All these parameters are video dependent. We assume that all video files are preprocessed to find proper parameters, and external data are inserted only in the video of the same format to simplify the decompression in client stations. Let SIZE(α) and SIZE(β) respectively denote the amount of inserted data and skipped data at each imprecise offset. To simplify the model, both SIZE(α) and SIZE(β) are assumed to be constant and a multiple of segments, even though it is easy to release such constraints. In the imprecise viewing model, the streams that display the same video may have different *content progressing rates*. The content progressing rate of a stream is defined as the number of segments in a video file divided by the time for st_i to complete its playback. Note that, external data insertion will increase the playback time and hence lead to a slower content progressing rate, while optional data skipping will decrease the playback time and result in an accelerated content progressing rate. The content progressing rate is different from *playback rate* which is fixed during playback, e.g., 30 frames/sec, no matter whether insertion or skipping is performed or not.

Let ℓ_G denote the maximal playback distance in a stream group G, which is defined by $\ell_G = |\delta(st_{\text{first}}) - \delta(st_{\text{last}})| + 1$, where st_{first} and st_{last} respectively denote the earliest and the

Table 1. Parameters of the imprecise viewing model.

Notations	Descriptions
α-set	Set of insertable offsets (denoted by α) available in video V
β-set	Set of optional offsets (denoted by β) available in V
$\alpha_{\max}$	Maximum amount of external data which can be inserted into V
$\beta_{\max}$	Maximum amount of optional data which can be skipped in V
$\alpha_{\min}$	Minimum distance between any two consecutive insertions of external data during the playback of V
$\beta_{\min}$	Minimum distance between any two consecutive skips of optional data during the playback of V

latest streams created in G. In figure 4 for example, $\ell_{G'2} = 12$, $st_{\text{first}} = st_{13}$ and $st_{\text{last}} = st_{20}$. For the buffer sharing model proposed in Section 2.1, $\ell_G + 1$ buffers are required for G, i.e., ℓ_G consuming buffers and one producing buffer. Thus, the buffer requirements for G can be reduced if ℓ_G is reduced. In the following subsections, ℓ_G is reduced by using three different strategies.

3.2. Buffer shrinking strategies

This section describes three shrinking strategies for reducing buffer requirements. All the strategies aim at reducing ℓ_G by taking advantage of different content progressing rates.

3.2.1. Backward shrinking. Backward shrinking strategy slows down the content progressing rate of the streams created earlier in G to reduce ℓ_G. Slowing content progressing rate of a stream can be achieved by inserting external data during the playback. Let st_{first} denote the earliest stream created in G. With backward shrinking, the producing process starts to retrieve external data instead of video data for buffer refreshing at each α offset allowed for st_{first} and will resume to retrieve video data after external data of $\text{SIZE}(\alpha) - 1$ segments has been consumed by st_{first}. We refer to the state in which a stream is consuming external data as the *insertion mode*. Each of the other streams in G then, based upon the desired the content progressing rate, may enter insertion mode at the same α offset with st_{first} or proceed through the playback without insertion. To meet the proposed imprecise viewing model, the constraints defined in Table 1 must not be violated, and, to prevent stream starvation, we must ensure that the inequality $\delta(st_{\text{last}}) \leq \delta(st_i) \leq \delta(st_{\text{first}})$ is always satisfied for any stream in G during playback. Let $P(st_i)$ denote the set of the imprecise offsets allowed for stream st_i during its playback. With backward shrinking, we derive each $P(st_i)$ based upon the following rules:

Rule A-1: Let $e_1, e_2, \ldots, e_{n_b}$ denote the imprecise offsets in $P(st_{\text{first}})$, in an ascending order, and n_b denote the number of elements. For each e_i in $P(st_{\text{first}})$, the following conditions must hold:

C1: $e_i \in \alpha$-set
C2: $|e_i - e_j| + \text{SIZE}(\alpha) \geq \mathit{MinInterval}_\alpha$ for $i \neq j$
C3: $n_b \times \text{SIZE}(\alpha) \leq \mathbf{min}(\ell_G - \text{SIZE}(\alpha), \mathit{MaxTotal}_\alpha)$

Rule A-2: For a stream st_i in G, the corresponding $P(st_i) = P(st_{\text{first}}) - \{e_1, e_2, \ldots, e_{q_i}\}$, where $q_i = \min(\lfloor(\delta(st_{\text{first}}) - \delta(st_i))/\text{SIZE}(\alpha)\rfloor, n_b)$.

Example 3.1. Assume the imprecise viewing model of the video file in figure 4 to be: α-set $= \{26, 30, 44, 53, 58, 65, 72, \ldots\}$, β-set $= \{16, 22, 30, 46, 52, 65, 80, 88, \ldots\}$, $\mathit{MaxTotal}_\alpha = 15$, $\mathit{MaxTotal}_\beta = 15$, $\mathit{MinInterval}_\alpha = 18$, $\mathit{MinInterval}_\beta = 12$, $\text{SIZE}(\alpha) = 3$ and $\text{SIZE}(\beta) = 2$. All the parameters are measured in segments.

Consider the stream group G'_2 in figure 4 for backward shrinking, where $\ell_{G'_2} = 13$ and $st_{\text{first}} = st_{13}$. Note that in order to have more reduction in $\ell_{G'_2}$, as much external data must be

Table 2. The backward shrinking process.

Cycle	..	54	55	56	57	58	59	..	75	76	77	78	79	80	..	99	100	101	102	103	104	105	..
$\delta(st_{13})$	..	26	d_1	d_2	d_3	27	28	..	44	d_1	d_2	d_3	45	46	..	65	d_1	d_2	d_3	66	67	68	..
$\delta(st_{14})$	..	25	26	d_1	d_2	d_3	27	..	43	44	d_1	d_2	d_3	45	..	64	65	d_1	d_2	d_3	66	67	..
$\delta(st_{15})$	..	23	24	25	26	27	28	..	44	d_1	d_2	d_3	45	46	..	65	d_1	d_2	d_3	66	67	68	..
$\delta(st_{16})$	..	20	21	22	23	24	25	..	41	42	43	44	45	46	..	65	d_1	d_2	d_3	66	67	68	..
$\delta(st_{17})$	..	19	20	21	22	23	24	..	40	41	42	43	44	45	..	64	65	d_1	d_2	d_3	66	67	..
$\delta(st_{18})$	..	18	19	20	21	22	23	..	39	40	41	42	43	44	..	63	64	65	d_1	d_2	d_3	66	..
$\delta(st_{19})$	..	16	17	18	19	20	21	..	37	38	39	40	41	42	..	61	62	63	64	65	66	67	..
$\delta(st_{20})$	..	15	16	17	18	19	20	..	36	37	38	39	40	41	..	60	61	62	63	64	65	66	..
$\delta(st_{21})$	..	14	15	16	17	18	19	..	35	36	37	38	39	40	..	59	60	61	62	63	64	65	..
$\ell_{G'2}$		**13**					**10**							**7**								**4**	..

inserted into stream st_{first} as possible to slow down its content progressing rate. For example, we can choose $n_b = 3$ and $P(st_{\text{first}}) = \{26, 44, 65\}$ in this case based on rule A-1. According to rule A-2, we can easily derive $P(st_{14}) = P(st_{\text{first}}) - \phi = \{26, 44, 65\}$ since $q_{14} = \min(\lfloor(26-25)/3\rfloor, 3) = 0$ and $P_{15} = P(st_{\text{first}}) - \{26\} = \{44, 65\}$ since $q_{15} = \min(\lfloor(26-23)/3\rfloor, 3) = 1$. Similarly, all the other $P(st_i)$ are derived as follows: $P(st_{16}) = P(st_{17}) = P(st_{18}) = \{65\}$ and $P(st_{19}) = P(st_{20}) = P(st_{21}) = \phi$. Table 2 illustrates the backward shrinking process from cycle 54 to cycle 105, where d_i denote the ith segment of the external data. Table 2 shows that two streams, st_{13} and st_{14}, will enter the insertion mode at the end of segment 26, while three streams ($st_{13}, st_{14}, st_{15}$) and six streams ($st_{13}, \ldots, st_{18}$) will respectively enter insertion mode at the end of segment 44 and 65. Note that, in Table 2, each time stream st_{13} enters the insertion mode, a stream, st_i, with $|\delta(st_{13}) - \delta(st_i)| < \text{SIZE}(\alpha)$ will also enter insertion mode at the same α offset. Thus, stream st_{13} always precedes all the other streams in G'_2 and starvation never occurs even when producing process stops retrieving video data (i.e., when st_{13} stays in the insertion mode). With backward shrinking, the maximal viewing distance ℓ_G can be reduced by $n_b \times \text{SIZE}(\alpha)$. As an example, in Table 2 $\ell_{G'_2}$ equals 13 in cycle 54, and 4 in cycle 105. Consequently, $\ell_{G'_2}$ is reduced by nine in total and nine buffers can be released for the stream group G'_2 shown in figure 4.

3.2.2. Forward shrinking. Forward shrinking strategy speeds up the content progressing rate of the streams created lately in a stream group G to reduce its maximal playback distance ℓ_G. Speeding up the content progressing rate of a stream means skipping some optional data during the playback. With forward shrinking, although optional data will be skipped in some streams, the producing process refreshes buffers with a full video file such that this single version is capable of satisfying various content progressing rates for different streams. Each stream in G then, based upon the desired the content progressing rate, dynamically consume data in the adequate consuming buffers for playback. Let b_{last} denote the buffer that the last stream st_{last} is consuming. Each time st_{last} skips optional data, the producing process will wrap data refresh to the buffer next to b_{last} in order to release buffers. To prevent starvation for any stream, we also need to ensure that the inequality

$\delta(st_{\text{last}}) \leq \delta(st_i) \leq \delta(st_{\text{first}})$ is always satisfied for any stream st_i in G during playback. Let $P(st_i)$ denote the set of imprecise offsets allowed for stream st_i. With forward shrinking, we derive each $P(st_i)$ based on the following rules:

Rule B-1: Let $e_1, e_2, \ldots, e_{n_f}$ denote the imprecise offsets in $P(st_{\text{last}})$, in an ascending order, and n_f denote the number of elements. For each e_i in $P(st_{\text{last}})$, the following conditions must hold:

C4: $e_i \in \beta$-set
C5: $|e_i - e_j| - \text{SIZE}(\beta) \geq \mathit{MinInterval}_\beta$ for $i \neq j$
C6: $n_f \times \text{SIZE}(\beta) \leq \min(\ell_G - \text{SIZE}(\beta), \mathit{MaxTotal}_\beta)$

Rule B-2: For a stream st_i in G, the corresponding $P(st_i) = P(st_{\text{last}}) - \{e_1, e_2, \ldots, e_{q_i}\}$, where $q_i = \min(\lfloor(\delta(st_i) - \delta(st_{\text{last}}))/\text{SIZE}(\beta)\rfloor, n_f)$.

Consider the imprecise viewing model in Example 3.1 and the stream group G_2' in figure 4 for forward shrinking. Note that, in this case, $\ell_{G_2'} = 13$ and $st_{\text{last}} = st_{21}$. According to rule B-1, we can choose $n_f = 5$ and $P(st_{\text{last}}) = \{16,\ 30,\ 46,\ 65,\ 80\}$ and, according to tule B-2, we can easily derive $P(st_{13}) = P(st_{\text{last}}) - \{16,\ 30,\ 46,\ 65,\ 80\} = \phi$ since $q_{13} = \min(\lfloor(26 - 14)/2\rfloor, 5) = 5$. All the other $P(st_i)$ can also be derived as follows: $P(st_{14}) = \phi$, $P(st_{15}) = \{80\}$, $P(st_{16}) = \{65, 80\}$, $P(st_{17}) = P(st_{18}) = \{46, 65, 80\}$, $P(st_{19}) = \{30,\ 46,\ 65,\ 80\}$ and $P(st_{20}) = P(st_{21}) = \{16,\ 30,\ 46,\ 65,\ 80\}$. Table 3 illustrates the forward shrinking process from cycle 54 to cycle 114. In Table 3, two streams (i.e., st_{20} and st_{21}) will skip optional data at the end of segment 16, while three, five, six and seven streams will respectively skip optional data at the end of segment 30, 46, 65 and 80. Note that, each time stream st_{21} skips optional data, a stream st_i with $|\delta(st_i) - \delta(st_{12})| < \text{SIZE}(\beta)$ will also skip the same optional data at the same β offset. Thus, st_{21} always runs behind all the other streams in G_2' and starvation never occurs even though those buffers behind st_{21} will be released. With forward shrinking, the maximal viewing distance ℓ_G can be reduced by $n_f \times \text{SIZE}(\beta)$. As an example, in Table 3 we have $\ell_{G_2'} = 13$ for cycle 54, while $\ell_{G_2'} = 3$

Table 3. The forward shrinking process.

Cycle	..	54 55 56 57	..	67 68 69	..	81 82 83	..	98 99 100	..	111 112 113 114	..
$\delta(st_{13})$	..	26 27 28 29	..	39 40 41	..	53 54 55	..	70 71 72	..	83 84 85 86	..
$\delta(st_{14})$	..	25 26 27 28	..	38 39 40	..	52 53 54	..	69 70 71	..	82 83 84 85	..
$\delta(st_{15})$	..	23 24 25 26	..	36 37 38	..	50 51 52	..	67 68 69	..	80 83 84 85	..
$\delta(st_{16})$	..	20 21 22 23	..	33 34 35	..	47 48 49	..	64 65 68	..	79 80 83 84	..
$\delta(st_{17})$	..	19 20 21 22	..	32 33 34	..	46 49 50	..	65 68 69	..	80 83 84 85	..
$\delta(st_{18})$	..	18 19 20 21	..	31 32 33	..	45 46 49	..	64 65 68	..	79 80 83 84	..
$\delta(st_{19})$	..	16 17 18 19	..	29 30 33	..	45 46 49	..	64 65 68	..	79 80 83 84	..
$\delta(st_{20})$	..	15 16 19 20	..	30 33 34	..	46 49 50	..	65 68 69	..	80 83 84 85	..
$\delta(st_{21})$	..	14 15 16 19	..	29 30 33	..	45 46 49	..	64 65 68	..	79 80 83 84	..
$\ell_{G'2}$		**13** **11**		**9**		**7**		**5**		**3**	..

for cycle 113. Thus, $\ell_{G'_2}$ is reduced by 10 in total and ten buffer blocks can be released for the stream group G'_2 shown in figure 4.

3.2.3. Two-way shrinking. Note that, there is an unbalanced quality loss among streams for backward and forward shrinking. For backward shrinking, those streams created earlier in a stream group suffer from more quality loss due to more external data insertion. In Table 2, for example, stream st_{13} incurs three insertions of external data during its playback but st_{21} proceeds its playback without any insertion. For backward shrinking, the lately created streams also suffer from more quality loss due to more optional data skipping as Table 3 shows. Two-way shrinking compromises these two shrinking strategies. For a stream group G, we not only slow down the content progressing rate of the streams created earlier in G and but also speed up the content progressing rate of the streams created lately. To prevent starvation during playback, in addition to rules A-1, A-2 and B-1, B-2, the following rule must be satisfied for two-way shrinking strategy:

Rule C-1: $n_b \times \text{SIZE}(\alpha) + n_f \times \text{SIZE}(\beta) \leq \ell_G - \max(\text{SIZE}(\alpha), \text{SIZE}(\beta))$,

where n_b and n_f respectively denote the element number in $P(st_{\text{first}})$ and $P(st_{\text{last}})$. Consider the imprecise viewing model in Example 3.1 and the stream group G'_2 in figure 4 for two-way shrinking. In this case $\ell_{G'_2} = 13$, $st_{\text{first}} = st_{13}$ and $st_{\text{last}} = st_{21}$. To reduce $\ell_{G'_2}$ as much as possible, we can choose $n_b = b_f = 2$ based on rule C-1 and, to release buffers as soon as possible, we can choose $P(st_{13}) = \{16_\alpha, 44_\alpha\}$ and $P(st_{21}) = \{16_\beta, 30_\beta\}$ based on rules A-1 and B-1 (The element with subscript α denotes an α offset and β denotes a β offset). According to rules A-2 and B-2, all the other $P(st_i)$ are derived as follows: $P(st_{14}) = \{26_\alpha, 44_\alpha\}$, $P(st_{15}) = \{44_\alpha\}$, $P(st_{16}) = P(st_{17}) = P(st_{18}) = \phi$, $P(st_{19}) = \{30_\beta\}$ and $P(st_{20}) = \{P(st_{21}) = \{16_\beta, 30_\beta\}$. Table 4 shows the two-way shrinking process of G'_2 from cycle 54 to 80. Compared with Table 2 (backward shrinking) and Table 3 (forward shrinking), Table 4 indicates a more balanced quality loss among streams. We also observe

Table 4. The two-way shrinking process.

Cycle	..	54 55 56 57 58 59	..	67 68 69 70	..	75 76 77 78 79 80	..
$\delta(st_{13})$	..	26 d_1 d_2 d_3 27 28	..	36 37 38 39	..	44 d_1 d_2 d_3 45 46	..
$\delta(st_{14})$	..	25 26 d_1 d_2 d_3 27	..	35 36 37 38	..	43 44 d_1 d_2 d_3 45	..
$\delta(st_{15})$	..	23 24 25 26 27 28	..	36 37 38 39	..	44 d_1 d_2 d_3 45 46	..
$\delta(st_{16})$	..	20 21 22 23 24 25	..	33 34 35 36	..	41 42 43 44 45 46	..
$\delta(st_{17})$	..	19 20 21 22 23 24	..	32 33 34 35	..	40 41 42 42 44 45	..
$\delta(st_{18})$	..	18 19 20 21 22 23	..	31 32 33 34	..	39 40 41 42 43 44	..
$\delta(st_{19})$	..	16 17 18 19 20 21	..	29 30 33 34	..	39 40 41 42 43 44	..
$\delta(st_{20})$	..	15 16 19 20 21 22	..	30 33 34 35	..	40 41 42 43 44 45	..
$\delta(st_{21})$	..	14 15 16 19 20 21	..	29 30 33 34	..	39 40 41 42 43 44	..
$\ell_{G'2}$	..	**13** **8**		**6**		**3**	..

that two-way shrinking reduces $\ell_{G'_2}$ more quickly than backward shrinking and forward shrinking do because, with two-way shrinking, external data insertion and optional data skipping can be performed in parallel without interference as shown in Table 4. Thus, buffers can be released quickly when two-way shrinking is used. As an example, in Table 4 ten buffers are released after cycle 80, but in Table 3 they can be released only after cycle 114.

3.3. *Remarks*

In this section, three shrinking strategies were explored to reduce buffer requirements for VOD systems where buffer sharing is employed. Note that, in addition to buffers, network channel requirements can also be reduced when shrinking strategies are used because shrinking may create multiple streams of the same playback offset. These distinct streams with the same offset can form a single multi-casting group without disturbing the continuity in playback and hence share a network channel for the delivery of video data. In figure 4, for example, there are initially eight multi-casting groups (one for each stream) but only three in the end ($\{st_3, st_6\}$, $\{st_2, st_5, st_7, st_8\}$ and $\{st_1, st_4\}$). Thus, five network channels can be saved in total. For a stream group G, the number of multi-casting groups is bounded by ℓ_G, the maximum playback distance in G. By using shrinking operations and reducing ℓ_G, fewer multi-casting groups are required and network channels can be saved.

However, shrinking strategies suffer from the overheads of maintaining imprecise offsets for each stream, especially when there are a large number of streams in the system. One solution to reduce the overhead is to maintain the imprecise offsets in terms of stream groups rather than individual streams. Let $P(st_{\text{first}})$ and $P(st_{\text{last}})$ respectively denote the set of imprecise offsets (sorted in an ascending order) allowed for the first stream st_{first} and the last stream st_{last} in stream group G. According to rule A-2, the α offsets allowed for any stream in G must consist of consecutive elements in $P(st_{\text{first}})$ and end with the last one in $P(st_{\text{first}})$. Similarly, according to rule B-2, the β offsets allowed for any stream in G must consist of consecutive elements in $P(st_{\text{last}})$ and end with the last one in $P(st_{\text{last}})$. Therefore, for a stream group G, we only need to maintain $P(st_{\text{first}})$ and $P(st_{\text{last}})$ together with the starting element for each stream in G. By using this method, the overheads for maintaining imprecise offsets can be largely reduced. Te show the correctness of all the shrinking strategies, four theorems are presented in the Appendix. Theorems 1, 2 and 3 show that streams never incur starvation during playback when shrinking strategies are employed, and Theorem 4 shows that all the shrinking strategies meet the imprecise viewing model defined in Section 3.1.

4. Simulation

In our simulations, six buffer management schemes described in Table 5 were examined, where schemes S4, S5, and S6 assume that the imprecise viewing model is available in the system. The average waiting time is used as the criterion to compare the performance of various buffer management schemes. Upon viewer arrival, the viewer joins an active session if buffer sharing is possible, or creates a new session if it is not. When there are not

Table 5. The six buffer management approaches.

Scheme	Description
S1	Double buffering
S2	Discrete buffer sharing model 1 (without splitting and merging)
S3	Discrete buffer sharing model 2 (with splitting and merging)
S4	Scheme S3 with two-way shrinking
S5	Scheme S3 with backward shrinking
S6	Scheme S3 with forward shrinking

enough resources for creating a new session, the viewer is placed in a waiting queue. The waiting time is measured as the time the viewer stays in the waiting queue.

4.1. Simulation model

Two sets of experiments were conducted. For the first set (Set1), we assume there is a single video file in the system so that all the buffers and disk bandwidths are dedicated to this video only. Schemes S1, S2, S3 and S4 were examined for different buffer sizes, disk bandwidths, and mean interarrival times. Schemes S4, S5 and S6 were examined for different amounts of quality loss, defined as the summation of the inserted external data and the skipped optional data during the playback of a video. For the second set (Set2), we assume there are several video topics in the system. Schemes S1, S2, S3 and S4 were examined for different numbers of video topics. The default values of parameters used for the simulations are given in Table 6. Moreover, all the experiments ran for 18000 sec (i.e., five hours) with a cycle time of 1 sec. The viewer arrivals were modeled using a Poisson process. In our simulations, MPEG-1 format is used. Due to variable bit rates in MPEG-1 (in general, from 1.5 Mb/sec to 2 Mb/sec), the buffer block size is chosen as 2 Mb (i.e., 250 KB), which means 450 MB buffer and 900 MB buffer can respectively accommodate about 30 and 60 minutes of MPEG-1 streams. Both external data size and optional data size are 1.5 minutes of MPEG-1 stream.

Table 6. The default value of parameters used in simulations.

Symbol	Set 1	Set 2	Description
B	450 MB	900 MC	Buffer size
D	10 sessions	30 sessions	Disk bandwidth
t_{interval}	60 sec	60 sec	Mean interarrival time of viewers
M	100 min	100 min	Mean movie length
α_{maxTotal}	9 min	9 min	Max total amount of inserted external data
β_{maxTotal}	9 min	9 min	Max total amount of removed optional data

4.2. Simulation results and discussion

For the first set of experiments, figure 5 shows the average waiting time of schemes S1, S2, S3 and S4 as a function of buffer size, disk bandwidth and mean interarrival time, respectively. Figure 6 shows the performance of schemes S4, S5 and S6, parameterized by the amounts of quality loss during the video playback. After comparing the results, we can make the following observations.

(1) Schemes S2, S3 and S4 perform well in comparison with scheme S1 over most of the parameters, showing that the buffer sharing technique improves the performance of a VOD system. The result also indicates that splitting and merging operations do work in most cases, as shown by comparing schemes S3 with S2, and that the imprecise viewing model leads to better results, as shown by comparing schemes S4 with S3. To show the viewing quality in scheme S4, Table 7 lists the average size of quality loss during playback for figure 5(a). In the case when the buffer size equals 300 MB, for example, although the average quality loss is 48.21 sec for each viewer, the average waiting time

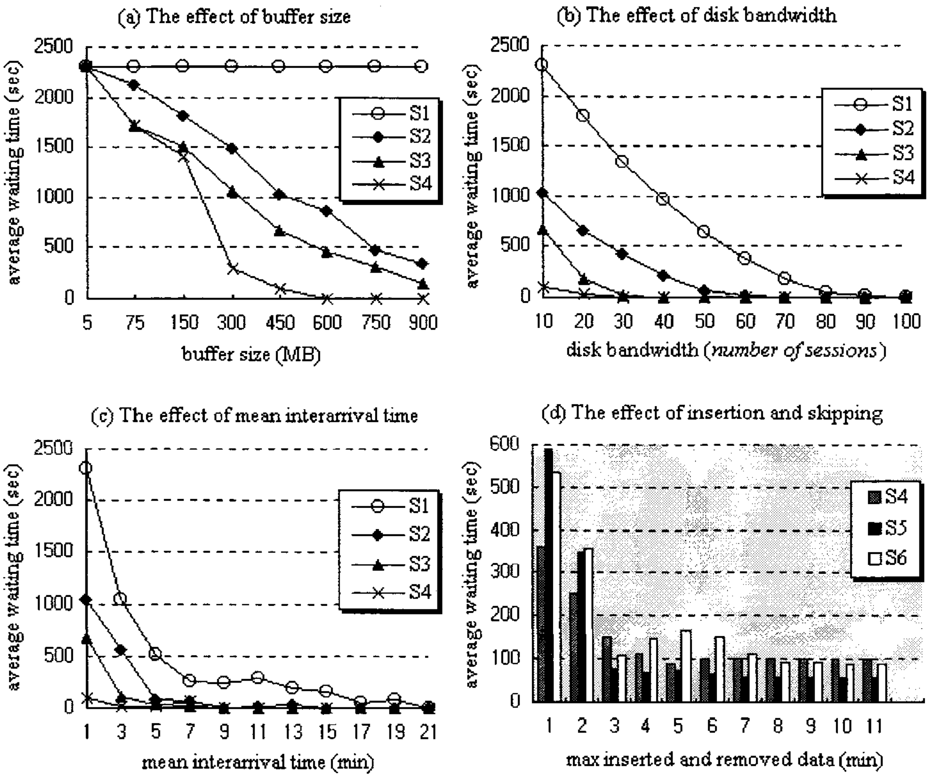

Figure 5. Simulation result.

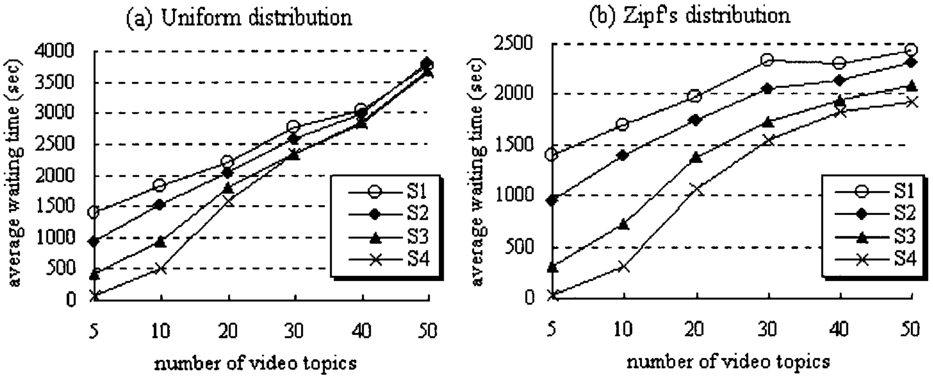

Figure 6. The effects caused by varying numbers of video topics.

is reduced by more than 700 sec as shown in figure 5(a). (Note she difference between S3 and S4.)

(2) Figure 5(a) shows that, as the buffer size increases, the average waiting time drops for schemes S2, S3 and S4 with S4 dropping the fastest followed by S3 and S2, respectively. The result reveals that large size buffers can help overcome the I/O bottleneck in VOD systems. Figure 5(a) also shows that scheme S1, the conventional double-buffering technique, is insensitive to buffer size. An increase in buffer size is useless for S1 because only two buffers are assigned for each video stream.

(3) Schemes S2, S3 and S4 perform much better than S1 when fewer sessions are supported by the storage. In figure 5(b), the performance of scheme S1 can catch up with others if the storage can support more than 80 sessions of MPEG-1 video. That is, for scheme S1, multiple disks must be employed owing to current storage technology which only supports about 4 to 10 sessions for a single disk. However, the number of sessions supported by the storage does not increase linearly with the number of disks due to the limitation of bus bandwidth and the problem of video file placement. The results in figure 5(b) reveal that, by using buffer sharing techniques, schemes S2, S3 and S4 overcome the I/O bottleneck to perform better.

(4) Figure 5(c) shows that average waiting time is improved for schemes S2, S3 and S4 more so than with scheme S1 over the range of mean interarrival times which are less than 21 minutes. The improvement is significant especially when the mean interarrival time is less than five minutes. The result shows that buffer sharing performs well when the mean interarrival time is small.

Table 7. The average quality loss of scheme S4 in figure 5(a).

Buffer size (MB)	5	75	150	300	450	600	750	900
Inserted data (sec)	0	0	21.66	20.75	41.80	100.37	142.48	97.93
Skipped data (sec)	0	0	1.53	27.46	110.75	136.98	129.36	175.42
Qualitf Loss (sec)	**0**	**0**	**23.19**	**48.21**	**152.55**	**237.35**	**271.84**	**273.35**

(5) Schemes S4, S5 and S6 have similar performances because the differences between their average waiting times are less than 60 sec over most cases in figure 5(d). Since S4 performs much better than S1, S2 and S3 as shown in figures 5(a)–(c), we can conclude that the shrinking strategies (i.e., S4, S5 and S6) perform better than the schemes without shrinking (i.e., S1, S2 and S3). Figure 5(d) also indicates that shrinking strategies reduce the average waiting time to a greater degree when more quality loss is allowed.

For the second set of experiments, figure 6 shows the average waiting time of scheme S1, S2, S3 and S4, parameterized by the number of video topics in the system. The selection of video topics is modeled using *uniform distribution* in figure 6(a) and *Zipf's distribution* in figure 6(b). Let the topics $1, 2, \ldots, n$ be sorted in a descending order of popularity, that is, $p_1 \geq p_2 \geq \cdots \geq p_n$ where p_i denotes the probability that a viewer choose topic i. According to Zipf's law [19], we have $p_i = c/i$ where c is a normalizing constant such that $\sum p_i = 1$. In figure 6, we observe that, no matter what type of distribution is used in topic selection, the average waiting time for all the schemes increases as the number of video topics increases. However, in most cases schemes S2, S3 and S4 still perform much better than scheme S1 especially when Zipf's distribution is employed. Since Zipf's distribution has been shown to closely approximate real-world user viewing behavior, the result demonstrates that our buffer management significantly improves in the performance of a VOD system.

5. Conclusion

Recently, advances in RAM technology have made it feasible to design a video server equipped with large size buffers. In this paper, we examined the issues of buffer management in video-on-demand (VOD) systems. First, we proposed a discrete buffer sharing model which employs batching and buffer sharing techniques in video servers to support a large number of concurrent services. Two operations, splitting and merging, were used in the model to fully utilize system resources such as buffers and disk bandwidths. Second, we introduced the concept of imprecise video viewing which assumes that certain degree of quality loss is allowed during the video playback. Based upon this assumption, three strategies which include backward shrinking, forward shrinking and two-way shrinking, were explored to further reduce the buffer requirements. Four theorems were proven in the Appendix to show the correctness of all the shrinking strategies. Besides, we also conducted several experiments to compare the performance of various buffer management schemes. The results demonstrate that our discrete buffer sharing model performs better than traditional buffer management techniques over most parameters and that resource adjusting operations, splitting and merging, significantly improve the reduction of average waiting times in the system. We also observe that, with concept of imprecise viewing, the proposed shrinking strategies can save much waiting time at minimal quality loss in video viewing. In conclusion, we believe that the proposed buffer management schemes help solve the I/O problems in VOD systems.

Appendix

In this appendix, four theorems were proven to show the correctness of the proposed shrinking strategies. Without loss of generality, we assume that $P(st_{\text{first}}) = \{\alpha_1, \alpha_2, \ldots, \alpha_{n_\alpha}\}$ and $P(st_{\text{last}}) = \{\beta_1, \beta_2, \ldots, \beta_{n_\beta}\}$, where n_α and n_β respectively denote the number of elements in $P(st_{\text{first}})$ and $P(st_{\text{last}})$. Note that, $n_\beta = 0$ for backward shrinking and $n_\alpha = 0$ for forward shrinking based on rules A-1 and B-1. For illustration purposes, the following notations and definitions are used.

$\bar{\alpha}$	: denotes the size of external data, in terms of number of segments.
$\bar{\beta}$	: denotes the size of optional data, in terms of number of segments.
round(e_i)	: denotes the time interval for all the streams associated with the imprecise offset e_i to complete data insertion or skipping at e_i.
st_i^k	: denotes a stream st_i that has completed its kth external data insertion or optional data skipping during video playback. Note that, we have $0 \le k \le n_\alpha$ for st_{first}^k and $0 \le k \le n_\beta$ for st_{last}^k.
$\bar{\delta}(st_i^k)$	: denotes the playback offset distance between st_i^k and st_{last}^0, i.e., $\bar{\delta}(st_i^k) - \delta(st_i^k) - \delta(st_{\text{last}}^0) + 1$. According to this definition, we have $\bar{\delta}(st_i^k) = \bar{\delta}(st_i^{k-1}) - \bar{\alpha}$ for external data insertion and $\bar{\delta}(st_i^k) = \bar{\delta}(st_i^{k-1}) + \bar{\beta}$ for optional data skipping.

Lemma 1. *With backward shrinking, a stream st_i in G with $\delta(st_{\text{first}}) - \delta(st_i) < \bar{\alpha}$ at the end of round(α_{k-1}) will enter insertion mode in round(α_k), where $0 \le k \le n_\alpha$.*

Proof: A stream st_i with $\delta(st_{\text{first}}) - \delta(st_i) < \bar{\alpha}$ at the end of round(α_{k-1}) means that the following inequality is satisfied, where y denotes the number of external data insertions incurred in st_i during the first $k-1$ rounds, i.e., $0 \le y \le k-1$.

$$\bar{\delta}\big(st_{\text{first}}^{k-1}\big) - \bar{\delta}\big(st_j^y\big) < \bar{\alpha}$$

By adding $(k-y-1)\bar{\alpha}$ to each side of the inequality above, we will have

$$\big[\bar{\delta}\big(st_{\text{first}}^{k-1}\big) + (k-1)\bar{\alpha}\big] - \big[\bar{\delta}\big(st_i^y\big) + y\bar{\alpha}\big] < (k-y)\bar{\alpha}.$$

With backward shrinking, we can obtain the following inequality by substitution.

$$\bar{\delta}\big(st_{\text{first}}^0\big) - \bar{\delta}\big(st_i^0\big) < (k-y)\bar{\alpha}$$

Now, according to the definition of q_i in rule A-2, we will have

$$q_i = \big\lfloor \big(\bar{\delta}\big(st_{\text{first}}^0\big) - \bar{\delta}\big(st_i^0\big)\big)/\bar{\alpha} \big\rfloor \le k-y-1.$$

Since $0 \le y \le k-1$, we can easily derive $0 \le q_i \le k-1$ which implies $\alpha_k \in P(st_i)$ (rule A-2). Thus, st_i will enter insertion in round(α_k). $\square$

Lemma 2. *With backward shrinking, a stream st_i^y with $y \neq 0$ at the end of round(α_{n_α}) must have $0 \leq \bar{\delta}(st_{\text{first}}^{n_\alpha}) - \bar{\delta}(st_i^y) < \bar{\alpha}$.*

Proof: According to sule A-2, a stream st_i^y with $y \neq 0$ at the end of round(α_{n_α}) must have its $P(st_i)$ starting with $\alpha_{n_\alpha - y+1}$, i.e., $P(st_i) = \{\alpha_{n_\alpha-y+1}, \dots, \alpha_{n_\alpha}\}$. Based upon the definition of q_i in rule A-2, we have

$$q_i = \lfloor (\bar{\delta}(st_{\text{first}}^0) - \bar{\delta}(st_i^0))/\bar{\alpha} \rfloor = n_\alpha - y.$$

By expanding this equation, we can obtain the following inequality:

$$(n_\alpha - y)\bar{\alpha} \leq \bar{\delta}(st_{\text{first}}^0) - \bar{\delta}(st_i^0) < (n_\alpha - y + 1)\bar{\alpha}$$

With backward shrinking, we can derive the following inequality by substitution:

$$(n_\alpha - y)\bar{\alpha} \leq \left[\bar{\delta}(st_{\text{first}}^{n_\alpha}) + n_\alpha\bar{\alpha}\right] - \left[\bar{\delta}(st_i^y) + y\bar{\alpha}\right] < (n_\alpha - y + 1)\bar{\alpha}$$

By subtracting $(n_\alpha - y)\bar{\alpha}$ from all the components above, we have $0 \leq \bar{\delta}(st_{\text{first}}^{k-1}) - \bar{\delta}(st_i^y) < \bar{\alpha}$ □

Theorem 1. *With backward shrinking, the inequality $\delta(st_{\text{last}}) \leq \delta(st_i) \leq \delta(st_{\text{first}})$ always holds for any stream, say st_i, in stream group G during the video playback.*

Proof: According to Lemma 1, st_{first} always proceeds st_i for backward shrinking since each time when st_{first} enters the insertion mode, st_i will also enter the insertion mode at the same offset if $\delta(st_{\text{first}}) - \delta(st_i) < \bar{\alpha}$. Thus, $\delta(st_i) \leq \delta(st_{\text{first}})$ is always satisfied for backward shrinking. To show $\delta(st_i) \geq \delta(st_{\text{last}})$, we only need to ensure that $\bar{\delta}(st_i^y) \geq 0$ holds at the end of round(α_{n_α}), where y is the element number in $P(st_i)$. Note that, this must be true for with $y = 0$ since there is no external data is inserted into st_i. Consider $y \neq 0$. According to Lemma 2, we can easily derive

$$\begin{aligned}
\bar{\delta}(st_i^y) &> \bar{\delta}(st_{\text{first}}^{n_\alpha}) - \bar{\alpha} \\
&= \left[\bar{\delta}(st_{\text{first}}^0) - n_\alpha\bar{\alpha}\right] - \bar{\alpha} \\
&= \bar{\delta}(st_{\text{first}}^0) - (n_\alpha + 1)\bar{\alpha} \\
&= \ell_G - (n_\alpha + 1)\bar{\alpha}
\end{aligned}$$

for backward shrinking. Since we have $\ell_G > (n_\alpha + 1)\bar{\alpha}$ based upon condition C3 in rule A-1, the inequality $\bar{\delta}(st_i^y) \geq 0$ is true for st_i^y with $y \neq 0$. Accordingly, $\delta(st_i) \geq \delta(st_{\text{last}})$ is also satisfied for the backward shrinking strategy. □

Theorem 2. *With forward shrinking, the inequality $\delta(st_{\text{last}}) \leq \delta(st_i) \leq \delta(st_{\text{first}})$ always holds for any stream st_i in G during the video playback.*

Proof: Due to space limitations, we omit this proof here. The proof of Theorem 2 is similar to the case of backward shrinking in Theorem 1. □

Theorem 3. *With two-way shrinking, the inequality $\delta(st_{\text{last}}) \le \delta(st_i) \le \delta(st_{\text{first}})$ always holds for any stream st_i in G during the video playback.*

Proof: Let y denote the element number in $P(st_i)$. For $y = 0$, the inequality $\delta(st_{\text{last}}) \le \delta(st_i) \le \delta(st_{\text{first}})$ must always be true because there is no data insertion and data skipping for st_i. For $y \ne 0$, consider $P(st_i) \cap P(st_{\text{first}}) \ne \phi$. According to Theorem 1, $\delta(st_i) \le \delta(st_{\text{first}})$ is always satisfied during video playback based upon Lemma 1. To show $\delta(st_i) \ge \delta(st_{\text{last}})$, we only need ensure that the boundary condition $\bar{\delta}(st_i^y) \ge \bar{\delta}(st_{\text{last}}^{n_\beta})$ holds after all the insertion and skipping rounds have been completed. Since $y \ne 0$, we can derive the inequality $\bar{\delta}(st_i^y) > \bar{\delta}(st_{\text{first}}^{n_\alpha}) - \bar{\alpha}$ based upon Lemma 2, that is,

$$\begin{aligned}
\bar{\delta}(st_i^y) - \bar{\delta}(st_{\text{last}}^{n_\beta}) &\ge \left[\bar{\delta}(st_{\text{first}}^{n_\alpha}) - \bar{\alpha}\right] - \bar{\delta}(st_{\text{last}}^{n_\beta}) \\
&= \left[\bar{\delta}(st_{\text{first}}^{0}) - n_\alpha\bar{\alpha} - \bar{\alpha}\right] - \left[\bar{\delta}(st_{\text{last}}^{0}) + n_\beta\bar{\beta}\right] \\
&= \left[\bar{\delta}(st_{\text{first}}^{0}) - \bar{\delta}(st_{\text{last}}^{0})\right] - \bar{\alpha} - (n_\alpha\bar{\alpha} + n_\beta\bar{\beta}) \\
&= \ell_G - \bar{\alpha} - (n_\alpha\bar{\alpha} + n_\beta\bar{\beta}) \\
&\ge \ell_G - \max(\bar{\alpha}, \bar{\beta}) - (n_\alpha\bar{\alpha} + n_\beta\bar{\beta}).
\end{aligned}$$

Owing to $\ell_G > \max(\bar{\alpha}, \bar{\beta}) - (n_\alpha\bar{\alpha} + n_\beta\bar{\beta})$ based upon rule C-1, the inequality $\bar{\delta}(st_i^y) \ge \bar{\delta}(st_{\text{last}}^{n_\beta})$ would be true. Thus, $\delta(st_i) \ge \delta(st_{\text{last}})$ is also satisfied for the case $P(st_i) \cap P(st_{\text{first}}) \ne \phi$. Now, consider $P(st_i) \cap P(st_{\text{last}}) \ne \phi$. According to Theorem 2, $\delta(st_{\text{last}}) \le \delta(st_i)$ always holds during video playback. To show $\delta(st_i) \le \delta(st_{\text{first}})$, we only need to ensure that the boundary condition $\bar{\delta}(st_i^y) \le \bar{\delta}(st_{\text{first}}^{n_\alpha})$ is also satisfied after all the insertion rounds and skipping rounds have been completed. Since $y \ne 0$, we can easily derive the inequality $\bar{\delta}(st_i^y) < \bar{\delta}(st_{\text{last}}^{n_\beta}) + \bar{\beta}$, that is

$$\begin{aligned}
\bar{\delta}(st_i^y) - \bar{\delta}(st_{\text{first}}^{n_\alpha}) &< \left[\bar{\delta}(st_{\text{last}}^{n_\beta}) + \bar{\beta}\right] - \bar{\delta}(st_{\text{first}}^{n_\alpha}) \\
&= \left[\bar{\delta}(st_{\text{last}}^{0}) + n_\beta\bar{\beta} + \bar{\beta}\right] - \left[\bar{\delta}(st_{\text{first}}^{0}) - n_\alpha\bar{\alpha}\right] \\
&= (n_\alpha\bar{\alpha} + n_\beta\bar{\beta}) + \bar{\beta} - \left[\bar{\delta}(st_{\text{first}}^{0}) - \bar{\delta}(st_{\text{last}}^{0})\right] \\
&= (n_\alpha\bar{\alpha} + n_\beta\bar{\beta}) + \bar{\beta} - \ell_G \\
&\le (n_\alpha\bar{\alpha} + n_\beta\bar{\beta}) + \max(\bar{\alpha}, \bar{\beta}) - \ell_G
\end{aligned}$$

Owing to $\ell_G > \max(\bar{\alpha}, \bar{\beta}) - (n_\alpha\bar{\alpha} + n_\beta\bar{\beta})$ based on rule C-1, $\bar{\delta}(st_i^y) \le \bar{\delta}(st_{\text{first}}^{n\alpha})$ would be true which implies that $\delta(st_i) \le \delta(st_{\text{first}})$ is always satisfied during playback. Accordingly, with two-way, the inequality $\delta(st_{\text{last}}) \le \delta(st_i) \le \delta(st_{\text{first}})$ is always satisfied for any st_i during playback. □

Theorem 4. *All the proposed shrinking strategies meet imprecise viewing model.*

Proof: With backward shrinking, it can be seen that st_{first} meets imprecise viewing model because, according to rule A-1, all the elements in $P(st_{\text{first}})$ meet the constraints defined in

Table 1. For any stream st_i in G, since we have $P(st_i) \subseteq P(st_{\text{first}})$ based on rule A-2, all the elements in $P(st_i)$ also meet the constraints in Table 1. Thus, with backward shrinking, all the streams meet imprecise viewing model. Similarly, with forward shrinking, all the streams also meet imprecise viewing model because we have $P(st_i) \subseteq P(st_{\text{last}})$ for any stream st_i in G (by rule B-2) and that all the elements in $P(st_{\text{last}})$ meet the constraints in Table 1 (by rule B-1). Two-way shrinking also meet imprecise viewing model because $P(st_i) \subseteq P(st_{\text{first}}) \cup P(st_{\text{last}})$ holds for any stream st_i in G. Consequently, all the shrinking strategies meet imprecise viewing model. □

References

1. S. Berson, S. Ghandeharizdeh, R.R. Muntz, and X. Ju, “Staggered stripping in multimedia information systems,” ACM SIGMOD, pp. 79–90, 1994.
2. A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley, “Channel allocation under batching and VCR control in Movie-On-Demand servers,” IBM Research Report, pp. 1–19, May 1994.
3. W. Effelsberg and T. Haerder, “Principles of database buffer management,” ACM Transactions on Database Systems, Vol. 9, No. 4, pp. 560–595, Dec. 1984.
4. E.A. Fox, “The coming revolution of interactive digital video,” Commun. ACM, Vol. 32, pp. 794–801, 1989.
5. E.A. Fox, “Standards and emergence of digital multimedia systems,” Commun. ACM, Vol. 34, pp. 26–30, 1991.
6. D. Le Gall, “MPEG: A video compression standard for multimedia applications,” Commun. of ACM, Vol. 34, No. 4, pp. 46–58, 1991.
7. A.D. Gelman, H. Kobrinski, L.S. Smoot, and S.B. Weinstein, “A store-and-forward architecture for video-on-demand service,” in Proc. of Intl. Conf. on Commun. (ICC), 1991, pp. 27.3.1–27.3.5.
8. L. Golubchik, J.C.S. Lui, and R. Muntz, “Reducing I/O demand in Video-On-Demand storage servers,” in Proc. of Intl. Conf. on Measurement and Modeling of Comp. Syst. (SIGMETRICS’95), 1995, pp. 25–36.
9. T.D.C. Little and D. Venkatesh, “Popularity-based assignment of movies to storage devices in a video-on-demand system,” Multimedia Systems, Vol. 2, No. 6, pp. 180–287, Jan. 1995.
10. K.K. Ramakrishnan, L. Vaitzblit, C. Gray, U. Vahalia, D. Ting, P. Tzelnic, S. Glaser, and W. Duso, “Operating system support for a video-on-demand file service,” Multimedia Syst., Vol. 3, pp. 53–65, 1995.
11. P.V. Rangan, H.M. Vin, and S. Ramanathan, “Designing an on-demand multimedia service,” IEEE Commun. Magazine, Vol. 30, pp. 56–64, July 1992.
12. P.V. Rangan and H.M. Vin, “Efficient storage techniques for digital continuous multimedia,” IEEE Trans. Knowl. Data Eng., Vol. 5, No. 4, pp. 564–573, Aug. 1993.
13. A.L.N. Reddy and J. Wyllie, Scheduling in a Multimedia I/O System, in Proc. of ACM Multimedia Conf., ACM Press: New York, 1992.
14. D. Rotem and J.L. Zhao, “Buffer management for video database systems,” IEEE Intl. Conf. on Data Engineering, pp. 439–448, 1995.
15. W.D. Sincoskie, “System architecture for a large scale video on demand service,” Comput. Networks ISDN Syst., Vol. 22, pp. 155–162, 1991.
16. A.J. Smith, “Sequentiality and prefetching in database systems,” ACM Trans. on Database Systems, Vol. 3, No. 3, pp. 223–247, Sept. 1978.
17. W.J. Tsai and S.Y. Lee, “Storage design and retrieval of continuous multimedia data using multi-disks,” in Proc. of Intl. Conf. on Parallel and Distributed Syst., Taipei, 1994, pp. 148–153.
18. P.S. Yu, J.L. Wolf, and H. Shachnai, “Design and analysis of a look-ahead scheduling scheme to support pause-resume for video-on-demand applications,” Multimedia Systems, Vol. 3, pp. 137–149, 1995.
19. G.K. Zipf, Human Behaviour and the Principles of Least Effort, Addison-Wesley: Reading, MA, 1949.

Wen-Jiin Tsai received her BS degree in Computer Science and Information Engineering from the National Chiao Tung University, Taiwan, in 1992. She is currently a Ph.D. candidate at the same university. Her research interests include multimedia information systems, video-on-demand, and server storage design.

Suh-Yin Lee received her BSEE degree from the National Chiao Tung University, Taiwan, in 1972, and her MS degree in Computer Science from the University of Washington, Seattle, in 1975. She joined the faculty of the Department of Computer Engineering at Chiao Tung University in 1976 and received the Ph.D. degree in Electronic Engineering there in 1982. Dr. Lee is now a professor in the Department of Computer Science and Information Engineering at Chiao Tung University. She chaired the department from 1991 to 1993. Her current research interests include multimedia information systems, object-oriented databases, image/spatial databases, and computer networks. Dr. Lee is a member of Phi Tau Phi, the ACM, and the IEEE Computer Society.