Multimedia Tools and Applications 9, 121–145 (1999)

Buffer-Sharing Techniques in Service-Guaranteed Video Servers

WEN-JIIN TSAI wjtsai@info1.csie.nctu.edu.tw
SUH-YIN LEE
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. Although a variety of studies have been focused on designing video servers suitable for VOD applications, the number of concurrent on-demand services supported by such servers is often limited by the I/O bandwidth of the storage systems. Recently, several researchers have focused on providing *guaranteed* services instead of on-demand services in a VOD system. A service is said to be guaranteed if every viewer can be served within a specified waiting time after the video has been subscribed. This paper describes a model called a *buffer-sharing autonomy*, which combines *batching* and *bridging* techniques to provide guaranteed services to VOD systems. We also describe how *buffer-sharing* techniques can be used in this model to provide the same services with fewer buffer resource through the use of playback-rate alteration. Additionally, in order to make buffer sharing efficient, it is important to group appropriate video streams together to share buffer resource. Four grouping policies are proposed for this purpose and the benefits yielded are also analyzed and compared. Finally, the results of our experiments show that the proposed buffer-sharing techniques promise the provision of guaranteed video services at low cost.

Keywords: buffer sharing, video server, video-on-demand

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 [7, 8], over networks. Recently, emerging video-on-demand (VOD) applications have received enormous attention from the telecommunications, entertainment, and computer industries. A variety of studies have focused on the design of high-performance VOD systems capable of handling a large number of services simultaneously [9, 14]. A typical VOD system consists of a *video archive* and a set of *video servers* [6, 15, 20]. The video archive maintains a collection of all available video files, and the video servers maintain small sets of frequently requested videos. This paper focuses on the design of a video server.

The characteristics of digital video files and video traffic differ substantially from those of conventional applications with regard to *continuity* and *high bandwidth* requirement. Continuity means that client stations must acquire the needed video data in time, namely, video servers must support enough I/O bandwidth to accommodate all the video streams so that they can be displayed uninterrupted. A variety of studies have focused on placement and scheduling techniques to boost server I/O bandwidth [1, 17, 18]. Because I/O

bandwidth is finite, the number of concurrent on-demand services a VOD system can support is often limited. Recently, studies aimed at resolving I/O problems have tended to reduce I/O demands on video servers through *batching*, *bridging*, and *adaptive piggybacking*. *Batching* delays video playback start time in order to serve multiple viewers using a single I/O stream [5]. *Bridging* reduces I/O demands by keeping recently used data in memory so it can be reused by subsequent viewers [11, 19, 21]. *Adaptive piggybacking* adjusts in-progress video display rates, in order to merge their respective I/O streams into a single stream [10]. Although these approaches create problems when providing VCR functionality, several solutions have been attempted. Dan et al. solved the problem in the context of batching by using *reserved channels* [3]. Yu et al. solved a similar problem by using *look-ahead scheduling* [23]. However, the I/O problem has not been solved yet.

In this paper, we focus on providing guaranteed services instead of on-demand services for VOD systems. A service is said to be *guaranteed* if every viewer is assured of being served within a specified waiting time after the video is subscribed. We first propose a model, called *buffer-sharing autonomy*, which combines *batching* and *bridging* techniques to support guaranteed services on video servers. We also describe how buffer sharing can be employed in this model to provide guaranteed services with less buffer requirement through the use of playback-rate alteration. In order to make buffer sharing efficient, it is important to share buffer resource by grouping appropriate videos together. Four grouping policies are thus proposed, including *non-sharing*, *non-grouping*, *symmetric grouping*, and *asymmetric grouping* policies. The benefits yielded by each of these grouping policies are also formally described and compared. Through experiment analysis we observed that buffer sharing, when used in conjunction with various grouping policies, saves a large number of buffers on the video servers. Finally, we believe that the methods proposed in this paper show a promise for providing guaranteed services to video servers at low cost.

1.1. Video server environment

Clients make requests for playbacks of movies at arbitrary times. A logical channel is reserved prior to starting playback. This channel corresponds to the set of resources required for the playback of a stream, i.e., network, disk, CPU processing power, etc. required to guarantee continuous delivery of the video stream. In this paper, we assume that channels in the video server are divided into three classes: *dedicated*, *on-demand*, and *contingency* channels.

The dedicated channels are used for movie playbacks at regular intervals to support guaranteed service, i.e., each client for these movies is guaranteed to be served within a pre-specified period of time. Since the *reneging* behavior (a client withdraws his/her request) of clients is influenced by the maximum waiting-time guarantee [5], we assume that the dedicated channels are reserved for popular movies, referred to as hot movies, to minimize overall reneging probabilities.

The remaining movies, called cold movies, share a common pool of on-demand channels. An on-demand channel is allocated to each request for a cold movie so that a new stream can be started, and is returned to the pool when the playback is completed. Because this common

pool has limited resources, it is possible that no on-demand channel is available when a playback request arrives, thus, the maximum delay cold-movie viewers may experience cannot be guaranteed. Several approaches to optimizing utilization of on-demand channels have been presented. In [2] it was proposed that more streams can be served concurrently through a method called *Interval Caching Policy*. Using this policy, data fetched by a stream is retained in memory so that it can be reused by other closely following streams. The data blocks between a pair of consecutive streams showing the same video is called an interval. When an interval is cached, the next stream can avoid a disk access because it can take advantage of a cache hit. In [4] an extended policy called *Generalized Interval Caching* was proposed to handle mixed interactive and long video workloads while retaining the benefits of interval caching such as superior performance and adaptation to changing access patterns. In [22], the buffer size required for interval caching was reduced using a method called *Buffer Shrinking*.

Contingency channels are designated to handle VCR control operations. In [5], it was proposed that pause and resume operations can be handled by setting aside a small pool of contingency channels. Using this method, a paused stream returns its resources to the server and re-acquires them from the contingency channel pool upon resumption. In [3], it was shown that when sufficient contingency channels are set aside, this method can guarantee a high probability of a resumption request being started within a short, pre-specified period of time.

Dividing channels into these three classes (i.e., dedicated, on-demand, and contingency) requires considering whether it is more beneficial to play back a movie at regular intervals, or simply start a new stream on demand in order to reduce overall reneging probability. The decision is also influenced by the probability that clients will pause while viewing movies because resumption requests cannot be delayed for long periods of time. In this paper, we assume that the channel allocation policy proposed in [3] is used for optimal allocation of channels to these three classes. This policy guarantees that overall reneging probabilities and resumption delays fall within certain pre-specified limits. In order to adapt to changes in workload patterns, we also assume that the server periodically uses this policy for system reconfiguration, i.e., the sets of dedicated, on-demand, and contingency channels may need to be resized. Such a system reconfiguration can easily be done by reassigning channels that are idle (i.e., not currently used by streams) to other classes. If not enough idle channels can be found, the reconfiguration is delayed until a sufficient number of in-progress channels have completed their playbacks.

Moreover, we assume that the interval-caching policy in [2, 4] is employed to optimize utilization of on-demand channels, and that the contingency-channel method in [5] is used to handle VCR control operations. Integration of the contingency-channel method with the interval-caching policy can be found in [2, 4]. The objective of this paper is to focus on optimizing the use of dedicated channels that play back hot movies at regular intervals. The method we propose is called *buffer-sharing autonomy*, and it optimizes the utilization of server resources by means of sharing. The contingency-channel method can also be integrated with our method for providing VCR control operations, as shown later.

2. Buffer-sharing autonomy

In this section, we present a technique called *buffer-sharing autonomy*, which uses batching, bridging, and buffer-sharing techniques to provide guaranteed services at low cost. Batching is employed to support guaranteed services, combining bridging with batching reduces required disk bandwidth, and buffer sharing further reduces required buffer resource costs.

2.1. Combining batching and bridging

As we have described, prior to starting the playback of a movie a logical channel is reserved to guarantee continuous delivery of the video stream. If more than one viewer requests the same movie within a short period of time, then they can all be batched together such that a single channel serves them all. However, a viewer may withdraw his/her request (renege) if the delay in starting the playback is significant. The probability of a viewer reneging increases with the length of the starting delay. To minimize the overall starting delay, a simple way for batching is to pre-allocate channels for movie playbacks at regular intervals, referred to as *batch intervals* (denoted by *Bt*), thus satisfying viewers' requests for movies. To ensure that viewers who request specific movies are served within T min, we can use T as the batch interval for these movies. In this way, video servers can easily achieve guaranteed services. In this paper, we refer to the channels offering guaranteed services as dedicated channels. As we have described, video servers often reserve dedicated channels for frequently requested movies (*hot movies*) to minimize the overall reneging probability. The disk bandwidth required for a dedicated channel is called a *dedicated session*.

Bridging has been proposed as a way of reducing I/O demands (disk session requirements) through the use of buffers [11, 19, 21]. It keeps recently used video data in buffers so that subsequent viewers who request the same video can acquire the data directly out of buffers instead of initiating another disk session. The length (in time) of the video data held in the buffers is called the *bridge length* (denoted by *Br*). Joining bridging with batching makes it possible to use fewer disk sessions to provide the same levels of guaranteed services to viewers. Consider a movie 100-min long. We can play this movie back every 20 min to ensure that all the viewers who want to see the video can be served within 20 min. In this case, $100/20 = 5$ dedicated sessions are required for this movie. Now consider another case in which each session uses buffers to hold 5-min video frames ($Br = 5$ min). Thus, viewers requesting within 5 min after sessions are initiated can also be served immediately through the use of the data in the buffers, and do not need to wait for the next session to be initiated. In this case, only 4 dedicated sessions are required to ensure that the maximum delay viewers experience is also 20 min ($100/4 - 5 = 20$).

2.2. Buffer sharing

As we have described above, if we want viewers arriving b min apart to be able to share the same disk session, then a buffer holding b min video frames is required for bridging. In this section, we present a technique called *buffer sharing* which makes buffers as sharable resources among in-progress sessions in order to reduce the total buffer requirement in

video servers. The idea behind is to recognize that smaller buffer sizes are needed for bridging when we reduce the temporal gaps between viewer requests. Such reductions can be achieved by altering the playback rates of the video streams. For example, if viewer B requests b min after viewer A, a b min buffer is required to bridge between viewers A and B. However, if the video playback for viewer B progresses at an accelerated rate (e.g., 5% faster than normal), and that for viewer A progresses at a slower rate (e.g., 5% slower than normal), then viewers A and B get temporally closer by one minute for every 10 min of viewing, which means that one-minute buffer space can be released for every 10 min. When this playback-rate alteration is applied to n sessions in the video server, n minutes' worth of buffer space can be released for every 10 min of playback. Since these sessions do not need b-sized buffers at all times through playback-rate alteration, a simple way to minimize total buffer-space consumption and thus maximize buffer utilization is to allow these sessions to share buffers. We refer to the group of sessions in which buffer sharing is performed as a *buffer-sharing autonomy*. The benefits of combining buffer sharing with the previously mentioned batching and bridging are described below.

Figure 1 shows a simple buffer-sharing autonomy consisting of four dedicated sessions, i.e., at most four sessions can be in progress during each movie cycle. The black rectangles in this figure represent the buffers in use and the gray ones, the buffers released. Every d minutes, a session is initiated with a b-byte-long buffer for bridging. L denotes the movie length (in minutes) and S^* the buffer release rate (bytes/min) per session duration. Let us consider the streams served by session ss_4. As this figure shows, these streams get closer and closer through playback-rate alteration and, therefore, the amount of buffer being used for bridging becomes lesser and lesser near the end of the video. A buffer of dS^* bytes is released for every d min. Consequently, although each of the four sessions is initiated with a b-byte-long buffer for bridging, this autonomy need not use four b-sized buffers at all time through buffer sharing. Figure 2 depicts the buffer size required for each session

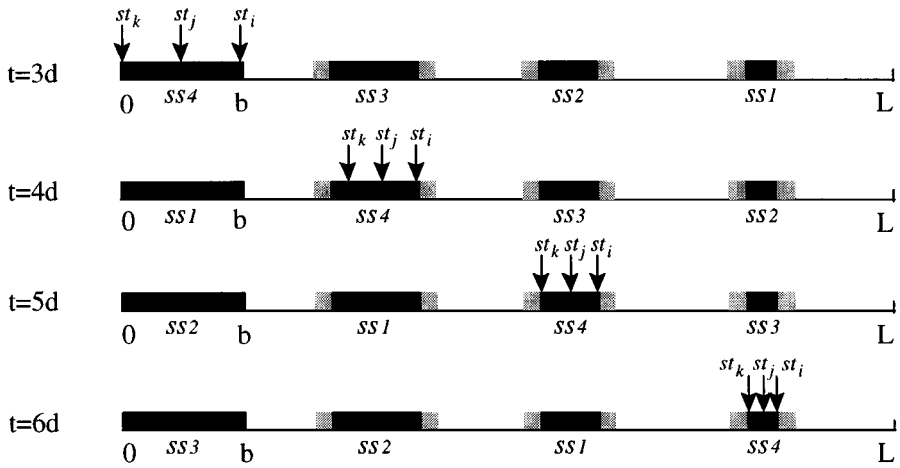


Figure 1. A simple buffer sharing autonomy.

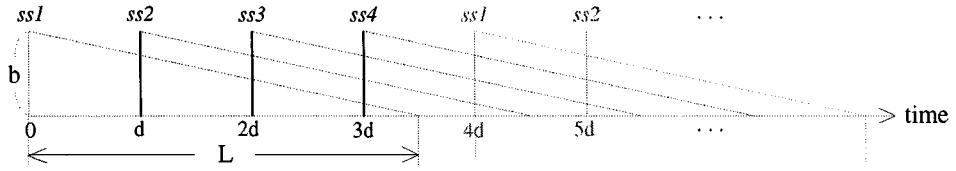


Figure 2. The benefits of buffer sharing.

Table 1. The buffer size needed at time $3d$ for each session in figure 2.

Session	ss_1	ss_2	ss_3	ss_4
Buffer size needed	$b - 3S^*d$	$b - 2S^*d$	$b - S^*d$	b

of this autonomy as a function of time. Consider the buffer requirement at time $3d$, i.e., four sessions are currently in progress and the 4th session has not yet released any buffer space. The buffer requirement for each session at this time is depicted in Table 1. Summed over the four sessions, the total buffer requirement is equal to $4b - 6S^*d$. By comparison with not using buffer sharing, we find that buffer space of size $6S^*d$ has been saved on the video server. This can easily be extended to n sessions, as is discussed in Section 4. Note that buffer measurement in this case is simplified by assuming that each session is capable of releasing buffer space at a rate of S^* throughout the entire movie cycle (and thus $b > 3S^*d$). In Section 4 we will analyze buffer requirement for more general situations when sessions may not have enough buffer space to release throughout the entire movie cycle, and when sessions may not be initiated at equal intervals, and may need unequal buffer sizes for bridging.

2.2.1. Buffer release rate. According to Table 1, the amount of buffer space that can be saved by using buffer sharing is dominated by the buffer release rate S^* which further depends on how fast the video streams (which use the same sessions through bridging) get closer via playback-rate alteration. There are several techniques for streams to get closer through rate alteration. Golubchik et al. proposed a technique called *piggybacking* [10], which alters playback rates to merge video streams. Their method of rate alteration adds or removes one frame for every n original frames, where n depends on the desired playback rates. There is ample evidence showing that video playback rates within $\pm 5\%$ of the nominal rate can be achieved in such a way that it is not perceivable by viewers [10]. This implies that buffer release rates can be as high as 10% of the normal display rate. Tsai et al. also proposed a technique called *buffer shrinking* [22], which alters playback rates to save buffer resource. They altered playback rates by inserting advertisement frames or skipping optional frames at predefined offsets in video streams. Their method was thus able to achieve larger-scale alterations although these changes may not be constant over short periods of time (because the rate changes occur only at predefined offsets). This implies that the buffer release rate S^* may not be constant over short periods of time either. In this paper we require that the time of session initiation be multiples of t_{Unit} relative to the initiation of the first session in the respective autonomy. t_{Unit} is the minimum period of time over which

buffer release per t_{Unit} minutes is constant. The selection of t_{Unit} depends on the choice of what playback-alteration methods to use as well as the video compression techniques used. To ensure that sufficient buffer spaces are available for continuous playback, we also require that each session must have at least $b_{\min}$-sized buffer space after buffers are released, where $b_{\min}$ is the minimum buffer requirement for a stream to maintain continuous display. The selection of $b_{\min}$ depends on disk block size and the retrieval cycle time used by the server.

2.2.2. Sizing the batch interval and the bridge length. Let V(*Bt*, *Br*) denote a video V characterized by a *batch interval* (denoted by *Bt*) and a *bridge length* (denoted by *Br*), meaning that the server displays video V every *Bt* minutes and, for each display, a buffer that can hold *Br* minutes' worth of video frames is used for bridging. Note that in order to save buffer resource, *Bt* must be greater than *Br* so there will be no duplicate video frames in the buffers. Assuming that viewer request rate follows a Poisson distribution, the average waiting W_i for a video $\text{V}_i(Bt_i, Br_i)$ would be approximately $W_i = (Bt_i - Br_i)^2/2Bt_i$ (as obtained from $0 \times \frac{Br_i}{Bt_i} + \frac{Bt_i - Br_i}{2} \times \frac{Bt_i - Br_i}{Bt_i}$). Let W_{Avg} denote the expected average waiting time for N service-guaranteed videos in the system. This is measured as $W_{\text{Avg}} = \sum_{i=1}^{N} W_i p_i$, where p_i denotes the probability of video V_i being requested among the N videos. In order to favor popular videos, we make $W_i p_i$ the same and equal to W_{Avg}/N for all videos and therefore, the average waiting time for a video will be inversely proportional to its access probability, i.e., $W_i = W_{\text{Avg}}/Np_i$. According to the definition of W_i, we have $(Bt_i - Br_i)^2/2Bt_i = W_{\text{Avg}}/Np_i$ from which Br_i is obtained as follows:

$$Br_i = \begin{cases} Bt_i - \sqrt{2Bt_i W_{\text{Avg}}/Np_i} & \text{if positive;} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

From this equation we can see that the choice of Br_i for video V_i (which determines how many buffers must be dedicated to V_i), is dominated by the size of Bt_i (which determines how many sessions must be dedicated to V_i). Let SS_{Total} denote the total number of sessions dedicated to the N videos and SS_i the number of sessions dedicated to video V_i. Note that, for video V_i of length L, SS_i means that we can make Bt_i equal to L/SS_i. The problem at hand is how to distribute SS_{Total} sessions among N videos to minimize total buffer requirement, subject to the expected average waiting time for each video. The straightforward way for doing so is to generate all possible values for $SS_1, SS_2, \ldots, SS_N$ to find a combination such that $\sum_{i=1}^{N} SS_i = SS_{\text{Total}}$ holds and $\sum_{i=1}^{N} Br_i SS_i$ is minimum, where

$$Br_i = \begin{cases} L/SS_i - \sqrt{2(L/SS_i)W_{\text{Avg}}/Np_i} & \text{if positive;} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

This equation is derived from Eq. (1) by substituting L/SS_i for Bt_i. Since the number of sessions dedicated to a video must be positive (i.e., $SS_i \geq 1$ must hold), $P(SS_{\text{Total}} - N, N)$ permutations must be generated in order to find the best one. Clearly this is a very complex problem for a large SS_{Total} and a large N because we need to search through all possible permutations before determining an optimal solution. Thus, we settle for ways for finding sub-optimal solutions. Since a popular movie needs more dedicated sessions to minimize overall average waiting time, and hence, more buffer space is required for bridging, we set

SS_i to be proportional to $\widehat{p}_i$ where $\widehat{p}_i$ is a function of p_i, the access probability of video V_i. Instead of enumerating all permutations of $SS_1, SS_2, \ldots,$ and SS_N, we calculate SS_i according to $SS_i = 1 + \lfloor \widehat{p}_i (SS_{\text{Total}} - N) \rfloor$, where subtracting N from SS_{Total} means reserving one dedicated session for each of the N videos. When there are r residual sessions, they are assigned to r different videos with highest access probabilities. As we will show later, an appropriate selection of $\widehat{p}_i$ makes the sub-optimal solution approximating the optimal one. After determining SS_i for a video V_i, the batch interval Bt_i can be obtained by using $Bt_i = L/SS_i$, and the bridge length Br_i can also be determined by using Eq. (2). Given the expected average waiting time W_{Avg}, the number of dedicated sessions SS_{Total}, and the access probabilities $p_i s$ for N service-guaranteed videos (usually, the N most popular movies), the algorithm for choosing an optimal Bt_i and Br_i for each video is summarized as follows:

1. Normalize the access probability for each of the N videos such that $\sum_{i=1}^{N} p_i = 1$.
2. Generate all possible permutations for $SS_1, \ldots, SS_N$, and search one such that $\sum_{i=1}^{N} SS_i = SS_{\text{Total}}$ with $SS_i \geq 1 \, \forall_i$ holds and $\sum_{i=1}^{N} Br_i SS_i$ for Br_i defined in Eq. (2) is minimum.
3. For each video, derive the batch interval using $Bt_i = L/SS_i$ and the bridge length using Eq. (2).

To find a sub-optimal solution for Bt_i and Br_i, the above Step 2 is replaced as follows:

2*. For $1 \leq i \leq N$, determine SS_i using the equation $SS_i = 1 + \lfloor \widehat{p}_i (SS_{\text{Total}} - N) \rfloor$.

In the following, we show how the optimal solution can be approached using the sub-optimal algorithm with an appropriate $\widehat{p}_i$. Consider a situation in which a video server dedicates 26 sessions to the 12 most popular videos with the expected average waiting time equal to 5 min. Figure 3 depicts the buffer size (i.e., $\sum_{i=1}^{N} Br_i SS_i$) required for three sub-optimal solutions (SubOpt1, SubOpt2, and SubOpt3) and the optimal solution (Optimal), as a function of video access patterns. The access patterns are modeled using a Zipf distribution in which the access probability for the ith most popular video is given by

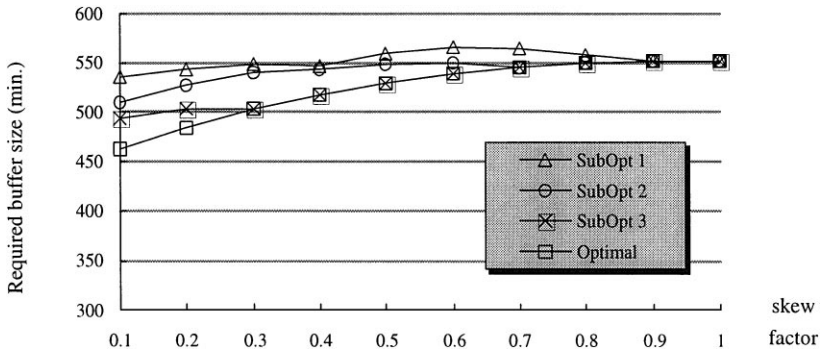


Figure 3. Comparison between optimal and sub-optimal algorithms.

$p_i = c/i^{(1-\theta)}$, and the parameter $\widehat{p}_i$ used for the three sub-optimal solutions are p_i, $\sqrt{p_i}$, and $\sqrt[4]{p_i}$, respectively. Note that c in the above formula is the normalization constant and θ is the skew factor. The details regarding Zipf distribution is described in Section 5.2. The result in figure 3 shows that SubOpt3 using the sub-optimal algorithm with $\widehat{p}_i = \sqrt[4]{p_i}$ most approximates the optimal solution.

2.3. VCR control

A video-on-demand system may allow clients to pause and restart viewing videos at arbitrary times. This section illustrates how PAUSE and RESUME operations are handled in the proposed buffer-sharing autonomy approach. A pause request in a buffer-sharing autonomy causes no action except ceasing to deliver video data to the client through the respective stream. While the stream is paused, no resources are returned to the system and all dedicated sessions keep on playing back toward the end of the video. Upon resumption of a paused playback, a session is re-acquired to serve this paused stream and one of three events might occur at this moment. First, the resumed stream may happen to lie in a range already being served by a session so it will be served immediately. Second, the temporal distance between the resumed stream and the closest session running behind it is less than a pre-specified time interval (referred to as the *maximum resumption delay*) and the stream will wait for the session to progress and then begin service. Third, there may be no sessions within the time interval required to re-start the resumed stream while honoring the pre-specified maximum resumption delay. In this case, a channel from the contingency pool is allocated to serve the resumption request.

Figure 4 gives an example in which a buffer-sharing autonomy consists of four dedicated sessions, ss_1, ss_2, ss_3, and ss_4. The movie length is L and the maximum resumption delay is T_{resume}. The shaded rectangles in this figure represent the ranges served by the corresponding

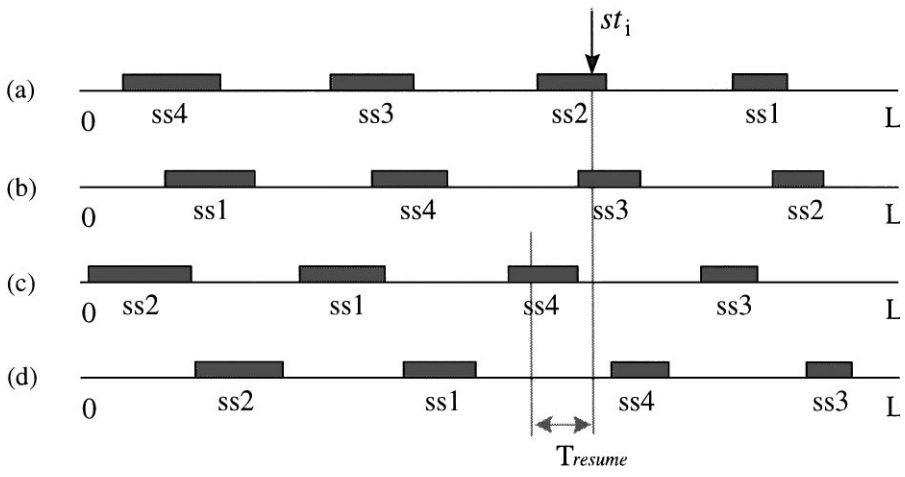


Figure 4. Handling PAUSE and RESUME operations.

sessions. Note that these "ranges" are formed through bridging and may become shorter near the end of the video due to rate-alterations (i.e., the ranges are time-compressed to release buffer space for sharing as described in the previous section). Figure 4(a) shows that stream st_i served by session ss_2 was paused at some time and figures 4(b) through (d) show three events that might occur upon resumption of st_i. In the case shown in figure 4(b), st_i will be served by ss_3 immediately and, in the case shown in figure 4(c), it will be served by ss_4 after a delay of less than T_{resume}. A contingency channel is required to resume st_i in the case shown in figure 4(d).

As we have described above, the channel allocation algorithm proposed in [3] can be used to determine the number of channels that should be set aside for contingencies such that a resumed stream has a very high probability of being re-started after a short time delay.

3. Autonomy grouping

Video servers cannot take full advantage of buffer sharing to save buffer resource when their dedicated sessions are not initiated at proper times. Consider a situation in which all the dedicated sessions are initiated at the same time. In this case, all sessions need buffers for bridging at the same time before any buffer space has been released, and thus the released buffer spaces cannot be reused to save resource costs. In order to make buffer sharing efficient, this section describes four grouping policies for choosing appropriate videos in autonomies that will share buffers. For all policies, the batch interval (Bt) and the bridge length (Br) for each video are assumed to be constant and known. (Note that both can be determined using the method described in Section 2.2.2) The grouping policies are designed to fulfill the following requirements: (1) Each video must be displayed in accordance with its original batch interval and bridge length; (2) Fewer buffers are required after grouping. Before describing these grouping policies, we first summarize the notation used.

L	movie length (in min)
n	the number of sessions in an autonomy (denoted from 0 to $n-1$)
ss_i	the video served by session i
c_i	the initiation time of session i, relative to the initiation of the first session in the autonomy
$d_{i,j}$	the interval (in min) between sessions i and j, i.e., $d_{i,j} = c_j - c_i$
d	If all the $d_{i,i+1}$ with $(0 \le i \le n-2)$ are equal then we use d instead of $d_{i,i+1}$ to simply the notation
B_i	the bridge length (in min) used for session i
S^*	buffer release rate (bytes/min) which depends on the rate alteration technique used
R	video display rate (bytes/min) at normal speed
B_{need}	buffer size needed for an autonomy
B_{min}	minimum buffer requirement for a session to maintain continuous display of a video
t_{Unit}	minimum temporal gap (in min) between successive sessions in an autonomy

3.1. *Baseline policy*

This is the normal situation. When a session is initiated, there is no attempt to release its buffer resource, i.e., all sessions use their assigned buffers throughout the entire movie cycle and there is no buffer sharing.

3.2. *Non-grouping*

This policy groups the sessions displaying the same movie into an autonomy to share buffers. With this policy, every autonomy consists of only one video. Within autonomies all sessions use equal amounts of buffer space for bridging, and the intervals between successive sessions are constant. For a video $V_x(Bt, Br)$, the corresponding autonomy has $n = \lceil L/Bt \rceil$ and other parameters are derived as follows: $s_i = V_x$, $c_i = i \times Bt$, $b_i = Br \times R$ for $0 \leq i \leq n-1$, and $d_{i,i+1} = Bt$ for $0 \leq i \leq n-2$.

3.3. *Symmetric grouping*

The symmetric grouping policy groups multiple videos into an *equal-interval autonomy*, defined as an autonomy within which the intervals between successive sessions are all equal. There are several ways to achieve this goal. The simplest one, called *simple grouping*, is to group those videos with the same batch intervals into a single autonomy. For example, m videos of batch interval Bt can easily form an equal-interval autonomy if display of each video is initiated in an interleaved manner at intervals of Bt/m. A simple example of this is illustrated in figure 5(c) which shows an autonomy resulting from grouping two

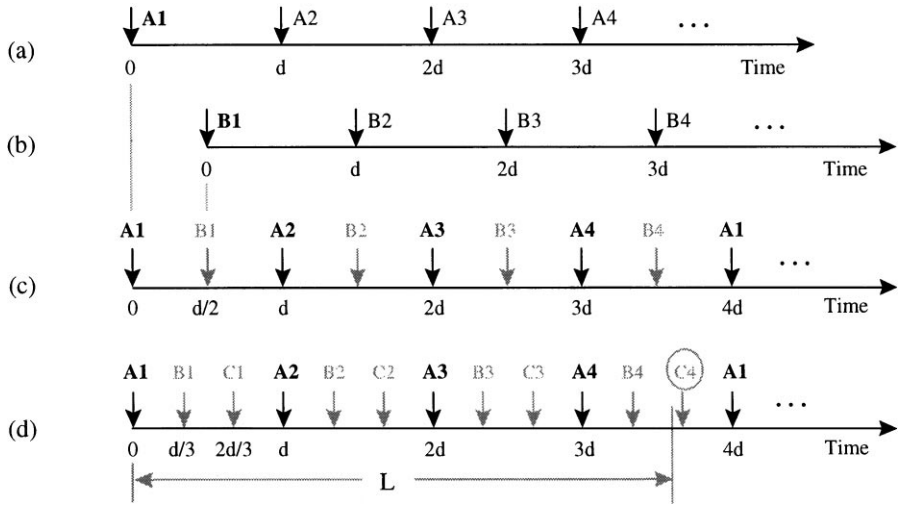


Figure 5. Symmetric grouping policy—simple grouping.

videos A and B depicted in figures 5(a) and 5(b), respectively. The batch intervals of both are assumed to be d and $\lceil L/d \rceil = 4$. As figure 5(c) shows, initiating the display of videos A and B in such an interleaved manner ensures that the result of grouping is an equal-interval autonomy and that each video is displayed in accordance with its own batch interval d. This technique can easily be extended to group more videos of the same batch intervals. Assuming m videos $V_1(Bt, Br_1)$, $V_2(Bt, Br_2), \ldots, V_m(Bt, Br_m)$ are grouped, the resulting autonomy has $n = m\lceil L/Bt \rceil$ and other parameters are derived as follows: $s_i = V_{(i \bmod m)}$, $c_i = i \times Bt/m$, $b_i = Br_{(i \bmod m)} \times R$ for $0 \le i \le n-1$ and $d_{i,i+1} = Bt/m$ for $0 \le i \le n-2$. Let a *movie cycle* be defined as the duration of a movie showing. It is a time interval of length L and starts when any session is initiated. In order to ensure that the playback sequences are identical in any two non-overlapping successive movie cycles, the above m videos can only be grouped if the following condition holds:

$$L - (n-1)Bt > \frac{m-1}{m} Bt. \tag{3}$$

Figure 5(d) shows a situation in which three videos A, B, and C are grouped in an autonomy that fails to satisfy inequality (3). As this figure shows, the playback sequence in the first movie cycle (starting at time 0) is $\{A_1, B_1, A_2, \ldots\}$ and that in the next non-overlapping movie cycle (starting at time $11d/3$) is $\{C_4, A_1, B_1, \ldots\}$. The two sequences are not identical. Such a situation never occurs when inequality (3) is satisfied in grouping. Subject to inequality (3), there is an upper bound on the number of videos grouped because of $m < \frac{Bt}{\lceil L/Bt \rceil Bt - L}$. This constraint dramatically decreases the complexity of measuring buffer requirement as described in Section 4.

An alternative to grouping multiple videos into equal-interval autonomies is repeatedly grouping the results of simple groupings, called *multi-level grouping*, which enables the accommodation of more videos in a single equal-interval autonomy. As mentioned above, m videos with the same batch interval (say Bt) can form an equal-interval autonomy (say EDA_1) with $d = Bt/m$ after a simple grouping is performed. Using multi-level grouping, other equal-interval autonomies with d being equal to Bt/m can also be merged with EDA_1, if they exist. As a result, more videos can be grouped into a single autonomy. Consider four videos, A, B, C, and D with batch intervals (Bt) equal to 45, 45, 45, and 15 min, respectively. The video length is 90 min. According to $n = \lceil L/Bt \rceil$ defined in non-grouping policy, the numbers of sessions dedicated to videos A, B, C, and D, are 2, 2, 2, and 6, respectively, as shown in the first column of Table 2. By using simple grouping, only videos A, B, and C

Table 2. Symmetric grouping policy.

Non-grouping	Simple grouping	Multi-level grouping
$G_A = \{A_1, A_2\}_{Bt=45}$ $G_B = \{B_1, B_2\}_{Bt=45}$ $G_C = \{C_1, C_2\}_{Bt=45}$ $G_D = \{D_1, D_2, D_3,$ $D_4, D_5, D_6\}_{Bt=15}$	$G_{A,B,C} = \{A_1, B_1, C_1, A_2, B_2, C_2\}_{Bt=15}$ $G_D = \{D_1, D_2, D_3, D_4, D_5, D_6\}_{Bt=15}$	$G_{A,B,C,D} = \{A_1, D_1, B_1, D_2, C_1, D_3, A_2,$ $D_4, B_2, D_5, C_2, D_6\}_{Bt=7.5}$

can be grouped into an autonomy because they have the same batch interval and inequality (3) holds for them. The results shown in the second column of Table 2 show six sessions in the autonomy with $Bt = 15$ and A, B, and C being initiated in an interleaved manner. Now, using multi-level grouping, this autonomy can be made to include video D, which also has $Bt = 15$. Their resulting autonomy has 12 sessions with $Bt = 7.5$, as depicted in the third column of Table 2. Note that all the sessions in this autonomy will be initiated at equal intervals, and videos A, B, C, and D will be displayed in accordance with their original batch intervals 45, 45, 45, and 15, respectively. It is interesting to observe in this table that, through repeated use of the simple grouping, the multi-level grouping policy enables grouping of videos with different batch intervals into a single equal-interval autonomy. The multi-level grouping algorithm is summarized as follows, where u^* is the maximum number of autonomies that can be grouped at a time using simple grouping policy.

Multi-level grouping algorithm
Begin
  while (more than one (say u) autonomies of the same batch interval (say v) are found) {
    Derive the maximum m for v according to the inequality (3).
    while ($u > 0$) {
      Let $u^* = \mathbf{min}(u, m)$;
      From the u autonomies, perform *simple grouping* on u^* autonomies;
      Set the batch interval of the resulting autonomy to be v/u^*;
      $u = u - u^*$;
    }
  }
End

3.4. Asymmetric grouping

As opposed to symmetric grouping, which is aimed at obtaining equal-interval autonomies, asymmetric grouping allows *general autonomies* to be the result of grouping in order to accommodate more videos in a single autonomy. A general autonomy is one in which the playback intervals between successive sessions need not be equal. A simple example of this is illustrated in figure 6(c) which shows a general autonomy resulting from grouping two videos A and B depicted in figures 6(a) and 6(b), respectively. The batch interval of video A is assumed to be $L/4$ and that of video B be $L/3$. In order to playback each video in accordance with its own batch interval, figure 6(c) shows that grouping videos of different batch intervals may result in an autonomy consisting of non-equal playback intervals. We refer to this as *asymmetric grouping*.

Let $\mathrm{V}_1(Bt_1, Br_1), \mathrm{V}_2(Bt_2, Br_2), \ldots,$ and $\mathrm{V}_m(Bt_m, Br_m)$ denote m videos, and Bt_{LCM} denote the lowest common multiple (LCM) of the m batch intervals; i.e., $Bt_{\mathrm{LCM}} = \mathrm{LCM}(Bt_1, Bt_2, \ldots, Bt_m)$. Using asymmetric grouping, these m videos can only be grouped if there is an integer k such that the following condition holds:

$$k\,Bt_{\mathrm{LCM}} > L \quad \text{and} \quad k\,Bt_{\mathrm{LCM}} - Bt_i < L \quad \text{for } 1 \le i \le m. \tag{4}$$

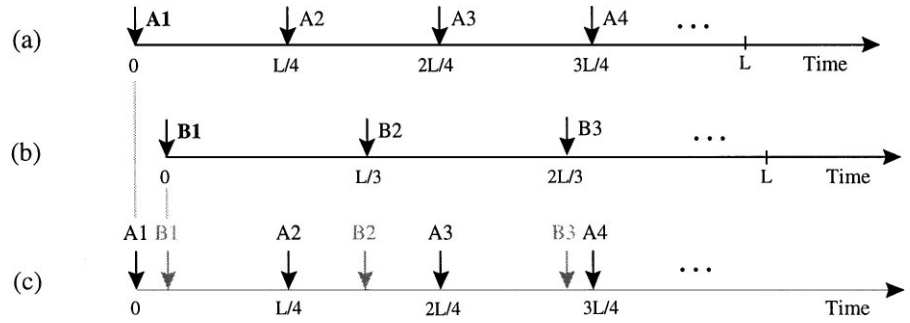


Figure 6. Asymmetric grouping policy.

The above condition ensures that, after a multiple of time L, the playback sequence will eventually return to the first playback sequence. This simplifies buffer measurement for the resulting autonomy. The asymmetric grouping policy works as follows. Consider two videos to be grouped, $V_1(Bt_1, Br_1)$ and $V_2(Bt_2, Br_2)$. Let V_1 be the only video in an autonomy as in the non-grouping policy, thereby allowing V_1 to be displayed in accordance with its batch interval Bt_1. To take full advantage of buffer sharing in the autonomy, the displays of video V_2 must start in the temporal gaps between the initiation times of V_1. The constraint on the process of grouping V_2 with V_1 is that V_2 must also be displayed in accordance with its own batch interval. The simplest way to lay out V_2's initiation times is to fill them into V_1's temporal gaps starting from the very first gap. After setting the first initiation time for V_2, the remaining times are all set subject to the batch interval of V_2 (i.e., Bt_2). If some initiation time does not fall into a temporal gap between showings of V_1, all initiation times for V_2 are delayed with t_{Unit} and verified again. This process continues until all the initiation times for V_2 can be accommodated within the temporal gaps between showings of V_1 (which we call a *feasible pattern* for V_2). If a feasible pattern is found, the grouping of V_1 and V_2 yields a valid autonomy; otherwise, asymmetric grouping fails to group V_2 with V_1. Assuming that two videos, $V_x(Bt_x, Br_x)$ and $V_y(Bt_y, Br_y)$, are to be grouped, the algorithm for grouping videos by means of this policy is given as follows.

Asymmetric grouping algorithm
Begin
if (videos V_x and V_y meet inequality (4), the asymmetric grouping condition.)
{
settle V_x and V_y in two autonomies (say A_x and A_y respectively) using non-grouping policy;
offset := 0;
conflict := TRUE;
while ((*conflict* = TRUE) and ($c_{\max} + offset < L$ holds for the max initiation time $c_{\max}$ in A_y)) {
for each initiation time c_i in A_y, derive $c_i' := c_i + offset$;

```
                check if there is any conflict between c_i' in A_y and each initiation time
                    in A_x;
                if (yes) {
                        increase offset by t_Unit;
                        conflict := TRUE;
                } else {
                        conflict := FALSE;
                }
        }
        if (conflict = FALSE) {
                for each initiation time c_i in A_y, increase c_i by offset;
                merge autonomies A_x and A_y into a single autonomy, say A_{x,y};
        } else {
                return the two autonomies cannot be grouped;
        }
    }
End
```

Note that the above binary grouping techniques can be easily extended to group three or more videos. Let us suppose we have to group m videos, $V_1, V_2, \ldots, V_m$. After the first two videos V_1 and V_2 have been grouped, the resulting autonomy can be viewed as that of a composite video $V_{1,2}$. The grouping of V_3 with $V_{1,2}$ can be carried out using binary grouping techniques to yield the autonomy $V_{1,2,3}$. Continuing the grouping operation in this fashion, the m videos can be grouped using a sequence of $m - 1$ binary grouping operations, resulting in a final autonomy of $V_{1,2,\ldots,m}$.

4. Performance analysis

This section presents analytic solutions for measuring the buffer requirement for the baseline, non-grouping, symmetric grouping, and asymmetric grouping policies.

4.1. Analysis of the baseline policy

We begin with the analysis of the baseline policy, which is very simple because there is no buffer sharing between sessions in progress. For a video $V(Bt, Br)$, since there are at most $\lceil L/Bt \rceil$ sessions that can be activated within a movie cycle and each session uses a buffer of size $Br \times R$ for bridging, the needed buffer size is determined by $B_{\text{need}} = \lceil L/Bt \rceil \times Br \times R$.

4.2. Analysis of the non-grouping policy

With a non-grouping policy, each video forms an autonomy that has equal intervals between successive sessions and has equal bridge lengths for all sessions. Let d and b denote respectively the batch interval and the bridge length of the resulting autonomy; i.e., $d_{j,j+1} = d$ for $1 \leq j \leq n-1$ and $b_i = b$ for $1 \leq i \leq n$. We first consider the case in which the

Table 3. Buffer needed for each of the n sessions.

Sessions	ss_1	ss_2	...	ss_{n-1}	ss_n
Buffer needed	$b-(n-1)S^*d$	$b-(n-2)S^*d$	...	$b-S^*d$	b

b-sized buffer assigned to each session can be released at a rate of S^* over the entire movie cycle; i.e., $b - B_{\min} \geq S^*L$. The maximum buffer size this autonomy requires occurs when the n sessions are all in progress and before the n^{th} session has released any buffer space. We compute the buffer size needed at this time for each session and show them in Table 3. Summed over the n sessions, the needed buffer size is equal to $B_{\text{need}} = \sum_{i=0}^{n-1}[b - iS^*d]$. However, since it is possible that the buffer space dedicated to each session cannot be released at rate S^* over the entire movie cycle due to $b - B_{\min} < S^*L$, the needed buffer size B_{need} is modified as follows

$$B_{\text{need}} = \sum_{i=0}^{n-1} B_i, \quad \text{where } b_i = \begin{cases} b - iS^*d & \text{if greater than } B_{\min}; \\ B_{\min} & \text{otherwise.} \end{cases} \tag{5}$$

For a video V(Bt, Br), using a non-grouping policy, the buffer size needed for the corresponding autonomy can be easily derived from Eq. (5) by using $n = \lceil L/Bt \rceil$, $d = Bt$, $b = BrR$.

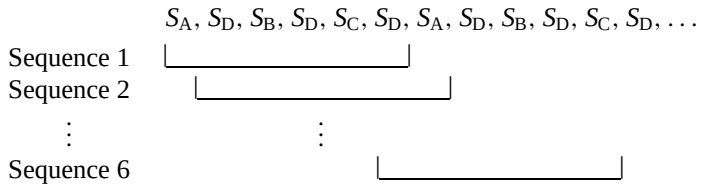
4.3. Analysis of the symmetric grouping policy

Measuring buffer requirement for the symmetric grouping policy is different from that for non-grouping policy because symmetric grouping may result in an autonomy consisting of multiple videos and, therefore, different bridge lengths may be used at the same time by different sessions belonging to this autonomy. When there are different bridge lengths in an autonomy, the needed buffer size depends on the sequence that videos are displayed. First we consider the case in which two videos with the same batch interval have been grouped into an autonomy. Let $V_A(Bt, Br_A)$ and $V_B(Bt, Br_B)$ denote these two videos. With a symmetric grouping policy, the resulting autonomy has $n = 2\lceil L/Bt \rceil$, $d = Bt/2$, and $b = Br_AR$ or Br_BR. The maximum buffer space needed for this autonomy occurs when the n sessions are all in progress. Since the display of the two videos will be initiated in interleaved fashion as follows

$$S_A, S_B, S_A, S_B, S_A, S_B, S_A, S_B, S_A, S_B, \ldots,$$

at any time instant the sequences of n in-progress sessions will be either $\{S_A, S_B, S_A, S_B, \ldots S_A, S_B\}$ or $\{S_B, S_A, S_B, S_A, \ldots S_B, S_A\}$. where S_A denotes the session for video A and S_B that for video B. Note that the buffer size needed for these two sequences would be unequal if videos A and B use different amount of buffer space for bridging. For each sequence (referred as *active sequence*), the maximum buffer space requirement occurs when the

last session has just been initiated and before any buffers are released. In the first active sequence, in which the last session shows video B, the required buffer size (say, $b^{\mathrm{B}}_{\mathrm{need}}$) can be derived from Eq. (5) by using $b = Br_{\mathrm{B}}R$ if (i **mod** 2) = 0, and $b = Br_{\mathrm{A}}R$ if (i **mod** 2) = 1. In the second active sequence, in which the last session shows video A, the required buffer size (say, $b^{\mathrm{A}}_{\mathrm{need}}$) can also be obtained from Eq. (5) by using $b = Br_{\mathrm{A}}R$ if (i **mod** 2) = 0 and $b = Br_{\mathrm{B}}R$ if (i **mod** 2) = 1. As a result, the buffer size required for this autonomy is determined to be the larger of buffer sizes required by either of these sequences; i.e., $B_{\mathrm{need}} = \mathbf{max}(b^{\mathrm{A}}_{\mathrm{need}}, b^{\mathrm{B}}_{\mathrm{need}})$. The result can be easily extended to m videos. Using symmetric policy, grouping m videos of the same batch interval will result in an autonomy which has exact m different active sequences, each ended with different video. Namely, when measuring the needed buffer size, at most m situations need to be considered. However, measuring buffer space requirement is more complicated if the autonomy results from multiple levels of simple grouping. We now consider a two-level symmetric grouping in which three videos, A, B, and C, have been grouped on level one, and later grouped with video D on level two. The playback sequence can be



Assuming the resulting autonomy has $n = 6$, there are six different active sequences $\{S_{\mathrm{A}}, S_{\mathrm{D}}, S_{\mathrm{B}}, S_{\mathrm{D}}, S_{\mathrm{C}}, S_{\mathrm{D}}\}$, $\{S_{\mathrm{D}}, S_{\mathrm{B}}, S_{\mathrm{D}}, S_{\mathrm{C}}, S_{\mathrm{D}}, S_{\mathrm{A}}\}, \ldots,$ and $\{S_{\mathrm{D}}, S_{\mathrm{A}}, S_{\mathrm{D}}, S_{\mathrm{B}}, S_{\mathrm{D}}, S_{\mathrm{C}}\}$ as the above shows. It is interesting to observe that the number of active sequences may not be equal to the number of videos grouped when multi-level grouping is used. More different active sequences may exist when more grouping levels are used and more videos are grouped. Fortunately, this situation is simplified by inequality (3), which ensures that each time a multiple of L minutes passes (and, therefore, a multiple of n sessions has been initiated) after the autonomy starts, the active sequence will eventually return to the first one. Namely, there are at most n different active sequences for the resulting autonomy and they can be easily derived as follows. Let $\{S_1, S_2, \ldots, S_n\}$ be the first active sequence of an autonomy. By regarding this sequence as a circular one, the other $n-1$ different active sequences are then the $n-1$ sequences ended with $S_1, S_2, \ldots, S_{n-1}$, respectively. Since each active sequence has n elements, the one ended with S_i starts from $S_{(i+1) \bmod n}$. Accordingly, the buffer size (B_{need}) required for an autonomy resulting from the symmetric grouping policy can be measured as follows:

Begin

$B_{\mathrm{need}} := 0$; ** the buffer size needed for the autonomy using symmetric grouping policy **/*

$b^i_{\mathrm{need}} := 0$; ** the buffer size needed for the active sequence ended with s_i **/*

for $i := 0$ **to** $n-1$ {

$$b^i_{\mathrm{need}} = \sum_{j=0}^{n-1} B_j, \text{ where } B_j = \begin{cases} b_{(i-j+n) \bmod n} - jS^*d & \text{if greater than } B_{\min}; \\ B_{\min} & \text{otherwise;} \end{cases}$$

if ($b^i_{\text{need}} > B_{\text{need}}$) $B_{\text{need}} := b^i_{\text{need}}$;
}
End

4.4. Analysis of the asymmetric grouping policy

The major difference between the symmetric and asymmetric grouping policies is that the result yielded from asymmetric grouping is a general autonomy instead of an equal-interval one. Analyzing buffer requirement for a general autonomy is more complicated because two identical active sequences may require different buffer sizes if the intervals between successive sessions are different. Fortunately, this problem is simplified by inequality (4), which ensures that, even when interval is considered, at most n different active sequences exist for the resulting autonomy. The buffer size required for an autonomy resulting from the asymmetric grouping policy is derived as follows:

Begin
$B_{\text{need}} := 0$; */* the buffer size needed for the autonomy using asymmetric grouping policy */*
$b^i_{\text{need}} := 0$; */* the buffer size needed for the active sequence ended with s_i */*
for $i := 0$ **to** $n - 1$ {

$$b^i_{\text{need}} = \sum_{j=0}^{n-1} B_j, \text{ where}$$

$$B_j = \begin{cases} b_{(i-j+n) \bmod n} - S^*((d_{(i-j+n) \bmod n,i} + L) \bmod L) & \text{if } > B_{\min}; \\ B_{\min} & \text{otherwise;} \end{cases}$$

if ($b^i_{\text{need}} > B_{\text{need}}$) $B_{\text{need}} := b^i_{\text{need}}$;
}
End

5. Results and discussion

This section presents the results of studies on buffer-sharing autonomies. Four grouping policies, *baseline* (BL), *non-grouping* (NG), *symmetric grouping* (SG), and *asymmetric grouping* (ASG), were examined. We conducted two sets of experiments. For the first set, the four policies were examined for different buffer release rates, bridge lengths, and batch intervals. The bridge lengths and the batch intervals used in this set were randomly generated. For the second set, the four policies were examined for different numbers of dedicated sessions, average waiting times, and access patterns. In this set the bridge lengths and the batch intervals used for each video were determined according to the method presented in Section 2.2.2. The default values of parameters used are summarized in Table 4. In the following discussion, we considered the total buffer requirement the video server needed to reserve for popular (service-guaranteed) videos as the measurement of interest. To be independent of video formats, the sizes of buffers were all measured in time (minute).

Table 4. The default values of parameters used.

Parameters	Default value
Total number of videos	1000
Number of service-guaranteed videos	15
Movie length	90 (min)
Mean release rates	10% of the normal rate
Mean bridge lengths	8 (min)
Mean batch intervals	30 (min)
B_{min}	1 (min)

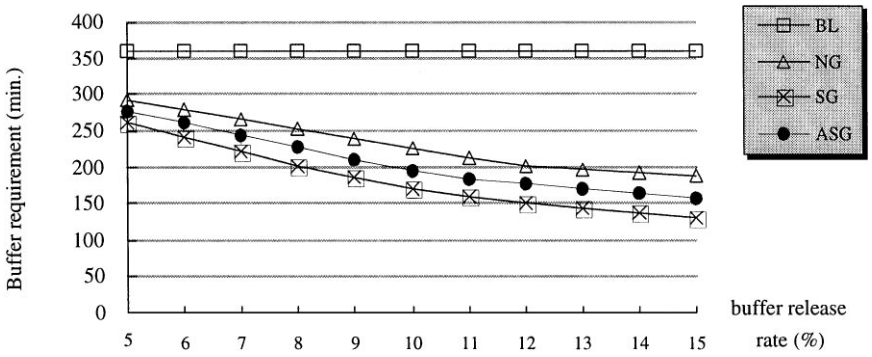


Figure 7. Effects of buffer release rates.

5.1. The effects of buffer release rate, bridge length, and batch interval

For the first set of experiments, figures 7–9 show the effects of buffer release rates, bridge lengths, and batch intervals on buffer-sharing autonomies, respectively. After comparing these figures, the following observations can be made.

1. We first consider the effects of buffer release rates on various grouping policies. The comparison is illustrated in figure 7, which shows the buffer requirement for various policies as a function of buffer release rates. This figure shows that, as the release rate increases, the total buffer requirement decreases quickly for all policies except the baseline policy which does not share buffers between different sessions. The result indicates that the proposed buffer-sharing technique does work to save buffer resource. This figure also implies that a greater reduction in buffer requirement can be achieved by using the symmetric grouping policy, than by using the asymmetric grouping policy.
2. Figure 8 depicts the buffer requirement for various grouping policies as a function of bridge lengths. In this figure it can be seen that, as the bridge length increases, more buffers are saved for policies NG, SG, and ASG as compared with the baseline policy. This is because using buffer sharing allows sessions with larger bridge lengths to release

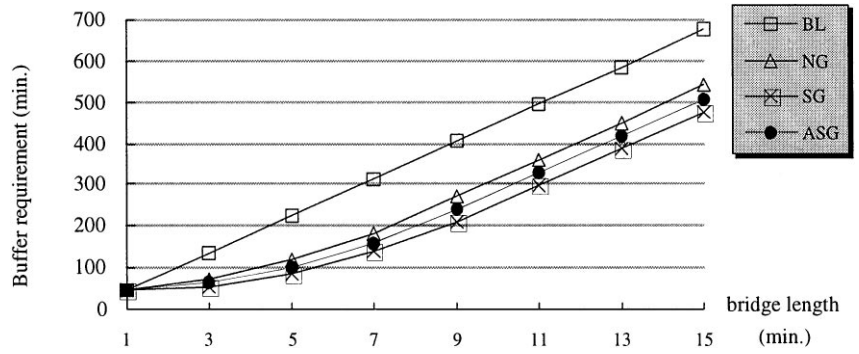


Figure 8. Effects of bridge lengths.

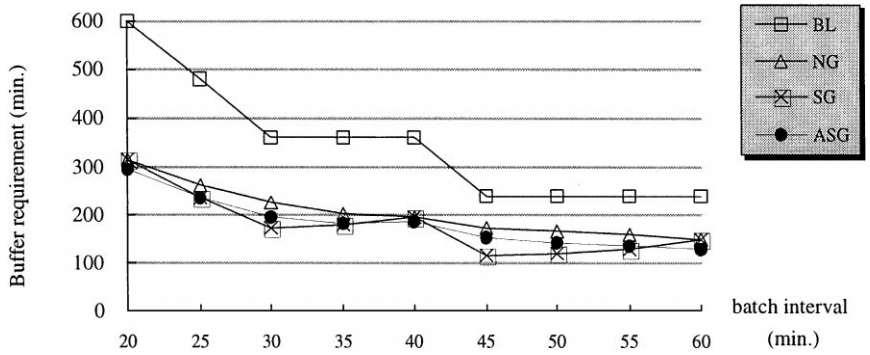


Figure 9. Effects of batch intervals.

more buffer space to other sessions. However, since the maximum buffer size that can be reused by other sessions is bounded by S^*L for each session, the maximum reduction in total buffer requirement is also bounded. That is why the curves of NG, SG, and ASG go in parallel with the curve of the baseline policy when the bridge length is greater than 10 min (In figure 8, $S^*L + B_{\min} = 10$). Figure 8 also shows that SG performs better than ASG because it saves more buffer spaces.

3. For a fixed movie length L, a smaller batch interval implies a larger number of sessions can be initiated within a movie cycle. Therefore, the buffer space released by one session can be reused by more sessions and fewer total buffer spaces are required. The result is illustrated in figure 9 which depicts the buffer requirement for various policies as a function of batch intervals. In this figure, buffer requirement is reduced more substantially for NG, SG, and ASG than for the baseline policy when the batch intervals were small. Moreover, since the number of videos that can be grouped using symmetric grouping is subject to inequality (3), an inappropriate batch interval selection may result in only a small number of videos being grouped together. For example, in the case that $L = 90$ and $Bt = 40$, the maximum m that satisfies inequality (3) is only 1. This shows that for some particular batch intervals in figure 9, there is only a small (or even none) reduction

in buffer requirement for SG as compared with NG. The results in this figure also show that the ASG policy is less affected by batch interval selection.

5.2. *The effects of dedicated sessions, average waiting time, and access patterns*

In the second set of experiments, the grouping policies were examined for different numbers of dedicated sessions, average waiting times, and access patterns. We assumed that guaranteed services were required for the 15 most popular videos in a video server that contained 1000 videos. The bridge lengths and the batch intervals used for popular videos were determined according to the method presented in Section 2.3. The access probabilities of videos were modeled using a Zipf distribution. In a Zipf distribution, if N videos are sorted according to the access frequency, then the access probability for the ith video is given by $p_i = c/i^{(1-\theta)}$, where θ is the distribution parameter and c the normalization constant. In our experiments, $\theta = 0.271$ which has been shown to approximate the user behavior at conventional video stores [5].

We first consider the effect of the number of sessions dedicated to the popular videos. This is illustrated in figure 10(a), in which the desired average waiting time was assumed to be 5 min. This figure shows that as the number of dedicated sessions increases, buffer spaces required to provide guaranteed services decreases as expected, no matter which

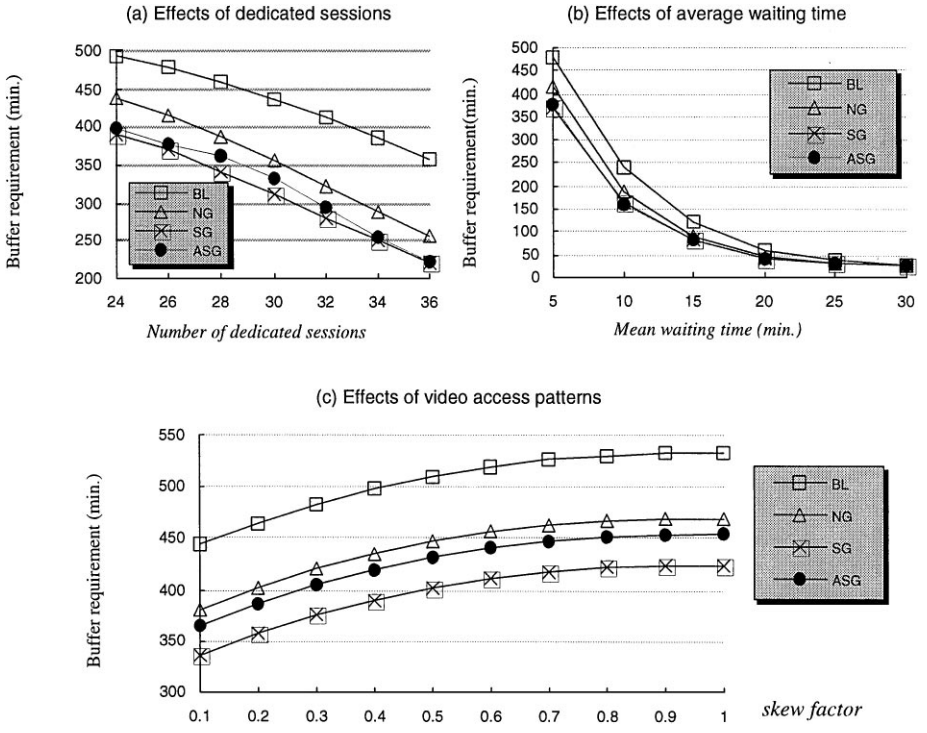


Figure 10. Effects of dedicated sessions, average waiting time, and access patterns.

policy is used. This figure also shows that, in comparison with the baseline policy, the NG, SG, and ASG policies can save a large amount of buffer spaces on the server. These reductions ranges from 20% to 30%, depending on the buffer-sharing technique and the grouping policies used.

We now consider the effect of the desired average waiting time. Average waiting time illustrated in figure 10(b) was measured on the assumption that the server dedicated 26 sessions to the 15 most popular videos. We observe that, the smaller the average waiting time, the more buffers are saved by NG, SG, and ASG, but not the baseline policy. For large average waiting times, since only few buffer spaces are needed to provide guaranteed services, no buffer sharing could be performed to further reduce buffer requirement.

When considering the effect of video access patterns, we also assumed that the server dedicated 26 sessions to the 15 most popular videos and the expected average waiting time was 5 min. The result is illustrated in figure 10(c) which depicts the buffer requirements for various policies as a function skew factor θ of Zipf distribution. In this figure we observe that SG performs best over all the values of θ, and then is ASG. The BL policy has the worst performance in all cases. This shows that our buffer-sharing technique and grouping policies do work in reducing buffer requirements for various access patterns. As θ varies from 0 to 1, the access probabilities vary from a highly skewed distribution to a uniform distribution. Figure 10(c) also implies that the more skewed access patterns (the smaller θ), the less buffers are required to achieve the desired average waiting time.

5.3. Equal-interval autonomy vs. unequal-interval autonomy

According to inequalities (3) and (4), the asymmetric grouping policy, being able to group more videos into a single autonomy, should save more buffer spaces than the symmetric grouping policy. However, in figures 7–10, we see that the asymmetric grouping policy always consumes more buffer spaces than the symmetric does. This is because the asymmetric grouping policy may result in an unequal-interval autonomies which often need more buffer resource than equal-interval ones (the result of the symmetric grouping policy). We explored this by investigating the buffer size required by individual active sequences belonging to these two kinds of autonomies. Figure 11(a) presents the buffer requirement for each active sequence belonging to six different autonomies with $n = 10$. Note that an autonomy with $n = 10$ means that it has ten different active sequences. The six autonomies are differentiated by the variations in the intervals between their successive sessions. Six variations 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0 are shown in this figure. Note that variation 0.0 means an equal-interval autonomy and the others unequal-interval ones. For the case of variation 0.0, figure 11(a) shows that all the active sequences need equal buffer sizes and, as the variation increases, the difference between the buffer sizes required by different active sequences also increases. Since the buffer requirement for an autonomy is determined to be the maximum among the required buffer sizes of its active sequences, the result in figure 11(a) shows that the autonomies with larger interval variations need more buffer spaces than those with smaller interval variations. That is, an unequal-interval autonomy needs more buffer resource than an equal-interval one. Similar results are also shown in figure 11(b), which uses autonomies with $n = 6$ for illustration.

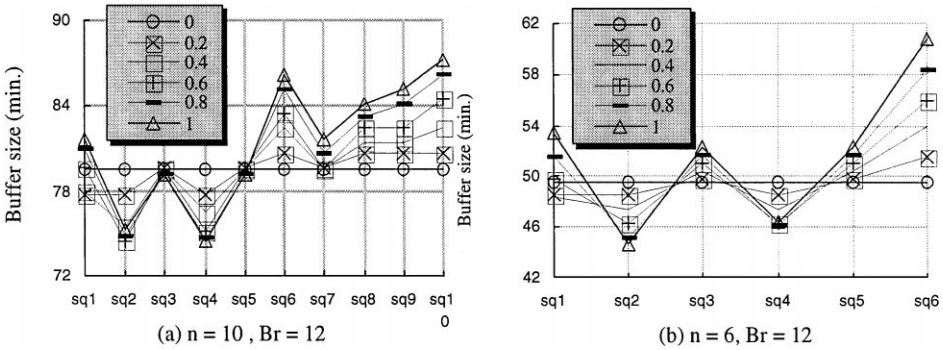


Figure 11. Equal-interval autonomy vs. unequal-interval autonomy.

6. Conclusions

The advances in RAM technology have made it feasible to design a video server equipped with large-sized buffers. This paper examined the issues of buffer management in a video server that provides guaranteed services for viewers. We first proposed a model that uses *batching* combined with *bridging* in video servers to support guaranteed services with fewer I/O requirement. In order to reduce buffer spaces required for bridging, we proposed a *buffer-sharing* technique that enables this model to provide the same services with fewer buffer resource through the use of playback-rate alteration. We also presented how to determine the bridge lengths and the batch intervals in this model according to the desired average waiting time and the number of dedicated sessions. Additionally, in order to make buffer-sharing efficient, several grouping policies were proposed, including baseline, non-grouping, symmetric grouping, and asymmetric grouping policies. The benefits that each policy yields were also analyzed and compared. The experimental results reveal that, by using buffer-sharing techniques, a large number of buffer spaces can be saved on the video server. This reduction in buffer requirement ranges from 20% to 30%. The results also indicate that the symmetric grouping policy performs better than the asymmetric because more buffers can be saved. In conclusion, we believe that our buffer management mechanism provides guaranteed services on a video server with low cost.

References

1. S. Berson, S. Ghandeharizdeh, R.R. Muntz, and X. Ju, "Staggered striping in multimedia information systems," ACM SIGMOD, 1994.
2. A. Dan, D.M. Dias, R. Mukherjee, D. Sitaram, and R. Tewari, "Buffering and caching in large-scale video servers," in Proc. of COMPCON'95, San Francisco, CA, March 1995, pp. 217–224.
3. A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley, "Channel allocation under batching and VCR control in Movie-On-Demand servers," IBM Research Report, May 1994.
4. A. Dan and D. Sitaram, "A generalized interval caching policy for mixed interactive and long video workloads," in Proc. of Multimedia Computing and Networking, San Jose, Jan. 1996.

5. A. Dan, D. Sitaram, and P. Shahabuddin, “Dynamic batching policies for an on-demand video server,” ACM Multimedia Systems, Vol. 4, pp. 112–121, 1996.
6. C. Federighi and L.A. Rowe, “A distributed hierarchial storage manager for a video-on-demand system,” in Proc. of IS&T/SPIE, San Jose, CA, 1994.
7. E.A. Fox, “Standards and emergence of digital multimedia systems,” Commun. ACM, Vol. 34, pp. 26–30, 1991.
8. E.A. Fox,“The coming revolution of interactive digital video,” Commun. ACM, Vol. 32, pp. 794–801, 1989.
9. 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.
10. L. Golubchik, J.C.S. Lui, and R. Muntz, “Reducing I/O demand in VOD storage servers,” TR 940037, CS Department, UCLA.
11. J. Huang and L.A. Stankovic, “Buffer management in real-time database,” Technical Report 90-65, Department of Computer Science, University of Massachusetts, July 1990.
12. D. Le Gall, “MPEG: a video compression standard for multimedia applications,” Commun. of ACM, Vol. 34, No. 4, pp. 46–58, 1991.
13. M.L. Liou, “Overview of the px64 kbps video coding standard,” Commun. of the ACM, Vol. 34, No. 4, 1991.
14. 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.
15. 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,” ACM Multimedia Systems, Vol. 3, pp. 53–65, 1995.
16. 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.
17. 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.
18. 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.
19. D. Rotem and J.L. Zhao, “Buffer management for video database systems,” IEEE Intl. Conf. on Data Engineering, pp. 439–448, 1995.
20. W.D. Sincoskie, “System architecture for a large scale video on demand service,” Comput. Networks ISDN Syst., Vol. 22, pp. 155–162, 1991.
21. A.J. Smith, “Sequentiality and prefetching in database systems,” ACM Trans. on Database Systems, Vol. 3, No. 3, pp. 223–247, Sept. 1978.
22. W.J. Tsai and S.Y. Lee, “Dynamic buffer management for near video-on-demand systems,” Multimedia tools and Applications, Vol. 6, pp. 61–83, 1998.
23. 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,” ACM Multimedia Systems, Vol. 3, pp. 137–149, 1995.
24. 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, and her Ph.D. degree in computer science and information engineering from the same school in 1997. She is now a technical staff member in Mentor Data System Inc. Her research interests include multimedia information systems, video-on-demand, server storage design, and set-top box 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 intersts 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.