A Sliding-Window Approach to Supporting On-Line Interactive Display for Continuous Media [1]

Chien-I Lee[†], Ye-In Chang[‡] and Wei-Pang Yang[†]

[†]Dept. of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan
Republic of China
{E-mail: leeci@dbsun1.cis.nctu.edu.tw}
{Tel: 886-3-5712121 (ext. 56601)}
{Fax: 886-3-5721490}

[‡]Dept. of Applied Mathematics
National Sun Yat-Sen University
Kaohsiung, Taiwan
Republic of China
{E-mail: changyi@math.nsysu.edu.tw}
{Tel: 886-7-5252000 (ext. 3819)}
{Fax: 886-7-5253809}

Abstract

To efficiently support continuous display for continuous media, many approaches based on the *striping* strategy that is implemented on a multi-disk drive have been proposed. However, the *striping* strategy can only support simultaneous display of continuous media which are predetermined before they are stored in the multi-disk drive. For an interactive display application, a system must support users to make any choice of objects for display even when display has started. Although Shahabi et. al. have proposed the *replication* and *prefetching* strategies for interactive display of continuous media, the combination of objects for display and the branch points for choices both have to be predetermined. Based on their strategies, they have to consider all the possible cases according to the given the combination of objects and the branch points of choices; it will require a lot of additional overhead of space and time. To reduce the overhead, in this paper, we will propose a *sliding window* approach to supporting interactive display for continuous media, in which we only record a little necessary information of retrieval of the following subobjects for display in a *sliding window*. For the way of interactive display described above, in which the combination of objects for display and the branch points for choices are predetermined, we call it *off-line* interactive display. As opposed to *off-line*, an *on-line* interactive display is the one, in which the combination of objects for display and the branch points for choices are dynamically determined. To support *on-line* interactive display, we will extend the *sliding window* approach to the *dynamic sliding window* approach. In this *dynamic sliding window* approach, the size of the *sliding window* can be changed according to the future requirements of data for display.

(*Key Words: data placement strategies, digital continuous media, interactive display, multi-disk drive, real-time database systems, striping*)

[1]This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-86-2213-E-110-004.

1 Introduction

Some media (such as audio and video) are classified as *continuous* because they consist of separate media *quanta* (such as audio samples or video frames), which convey meaning only when presented in time. Several multimedia types, video, in particular require high bandwidths and large storage space. For example, one hour and a half of video based on HDTV (High Definition Television) quality images has approximately 36 Gbits of data and requires approximately a 100 Mbps bandwidth while a current typical magnetic disk drive has only an 80 Gbit capacity and approximately a 20 Mbps bandwidth. In general, conventional file systems are unable to guarantee that clients can access continuous media in a way that permits delivery deadlines to be met under normal buffering conditions. Therefore, finding a way to support continuous retrieval of multimedia data at the required bandwidth and a way to store the multimedia data are challenging tasks [2, 5, 7, 13, 14, 21]. In this paper, for convenience, we use an object to denote an object of digital continuous media.

Previous approaches to supporting real-time applications of digital continuous media can be classified into three directions: *continuous retrieval*, *random access* and *interactive browsing*. To support *continuous retrieval*, some strategies clustered the data on a single disk to reduce the cost of disk head movement [6, 16, 17, 18, 22, 23], and some strategies tended to increase the bandwidth of disk device by using *parallelism*, which combines the bandwidths of multiple disks to provide a high data bandwidth requirement [1, 9, 12]. To support *random access*, like *editing operations*, since there is a trade-off between the flexible placement and the overhead of disk head movement in a disk drive, a straightforward idea is to find a compromise between them, which restricts a group of consecutive data to be stored consecutively in each cylinder on the disk while such a group of data can be randomly stored on any cylinder [11]. To support *interactive browsing* such as *fast forward* and *fast backward*, some strategies used the *scalable compression algorithms* to generate the multiresolution data [10], and some strategies supported browsing at any desired display speed by a predetermined *sampling* procedure [4].

Since future demands for high storage capacity and high bandwidth are expected, to efficiently support these three different directions for real-time applications, the *striping* strategy [1, 12] implemented on a multi-disk drive has been proposed. Basically, in the *striping* strategy [1, 12], the object is split up into subobjects and placed in various locations on the disks. Moreover, in the *simple striping* strategy [1], the striped subobjects are stored among the multi-disk drive in a predetermined sequence and must be read in this predetermined sequence to guarantee continuous retrieval. Furthermore, Berson et al. [1] further generalized the simple *striping* (called *staggered striping*) to support a database that consists of a combination of objects, each with a different bandwidth requirement.

Note that the *striping* strategy can only support simultaneous display of objects which are predetermined before they are stored in the multi-disk drive. For an interactive display application, a system must support users to make any choice even when display has started. Although in [3, 20], they have proposed the *replication* and *prefetching* strategies for interactive display, the combination of objects for display and the branch points for choices both have to be predetermined. Based on their strategies, they have to resolve all the possible *conflicts* before display starts, where a *conflict* means that a pair of two subobjects which are stored in the same disk must be retrieved simultaneously. Consequently, their strategies will require a lot of additional overhead of space and time. Moreover, display may be interrupted by the users at any time. When this current display is interrupted and is no longer needed, efforts for *prefetching* or *replication* are wasted because the following subobjects for display do not need to be retrieved. Therefore, to reduce the overhead, in this paper, we will propose a *sliding window* approach to supporting interactive display for continuous media, in which we only record a little necessary information of retrieval of the following subobjects for display in a *sliding window* and resolve the possible conflicts within the *sliding window*. From the simulation results, we find that the larger the size of a *sliding window* is, the larger the waste of time and space once display is interrupted. Therefore, we prefer a *sliding window* with a *smaller* size. However, a *hiccup* can occur when the size of the *sliding window* is not large enough, where a *hiccup* means that the subobjects for being displayed has not been retrieved and will be ready in the next time interval. Therefore, we have to select a proper *sliding window* size for a predetermined interactive display. A mathematical analysis will be studied to speed up the selection of the value of a *sliding window* size.

Furthermore, for the way of interactive display described above, in which the combination of objects for display and the branch points for choices are predetermined, we call it *off-line* interactive display. As opposed to *off-line*, an *on-line* interactive display is the one, in which the combination of objects for display and the branch points for choices are dynamically determined. To support *on-line* interactive display, we will extend the *sliding window* approach to the *dynamic sliding window* approach. In this *dynamic sliding window* approach, the size of the *sliding window* can be changed according to the future requirements of data for display. Since the subobjects for future display are not predictable, we use some information of previous retrieval to guess the possible subobjects for future display. From the simulation results, we find that the probability of *hiccup* is decreased as the amount of information of previous retrieval is increased.

Basically, our proposed approach can be applied not only to the multi-disk drives, but also to the parallel database management systems, such as, parallel multimedia systems based on the *share-nothing* architecture [9]. A *share-nothing* architecture consists of a number of processors interconnected by a high speed communication network. Processors do not share disk drives or random access memory and can only communicate with one another by sending messages using an interconnection network. In this case, we assume that the bandwidth of both the network and the network device driver exceeds the bandwidth requirement of an object. The rest of the paper is organized as follows. Section 2 briefly describes the *striping* strategy that is applied in our approach. Section 3 presents the proposed *sliding-window* approach. Section 4 presents the simulation results of the proposed approach. In Section 5, we will present a mathematical analysis of the *sliding window* approach. In Section 6, we will extend the *sliding window* approach to support *on-line* interactive display. Section 7 contains a conclusion.

2 The Striping Strategy

In our approach, we apply the *striping* strategy [1] to arrange the objects on a multi-disk drive. Suppose the bandwidth of both the network and the network device driver exceeds the bandwidth requirement of an object. Assume that there are N disks which operate independently, called a *multi-disk* drive and each disk has a fixed bandwidth d, a worst seek time WS, and a worst latency time WL. The *simple striping* strategy uses the aggregate bandwidth of several disk drives by striping an object across multiple disks [1]. For example, an object X with bandwidth requirement C_X at least requires the aggregate bandwidth of $M_X = \lceil \frac{C_X}{d} \rceil$ disk drives to support continuous retrieval of X. (Note that the maximum aggregate bandwidth of a multi-disk drive with N disks is $(N \times d)$ which must not be smaller than C_X.) Moreover, object X is organized as a sequence of equi-sized subobjects (X_0, X_1, X_2, ...), where the size of a subobject is s_X Mbits. Each subobject X_i represents a contiguous portion of X and is stored randomly in the disks. For the load balance for each disk, the subobjects of X are assigned to the N disks in a round-robin manner and the N disks are divided into R $(= \lfloor \frac{N}{M_X} \rfloor)$ disk clusters, where each cluster is assigned to an object for retrieval of the M_X consecutive subobjects to guarantee the real-time transfer. The duration of retrieval of a subobject is fixed for all subobjects and is in terms of a *time interval* I. According to [16], concurrent pipelining of retrieval and

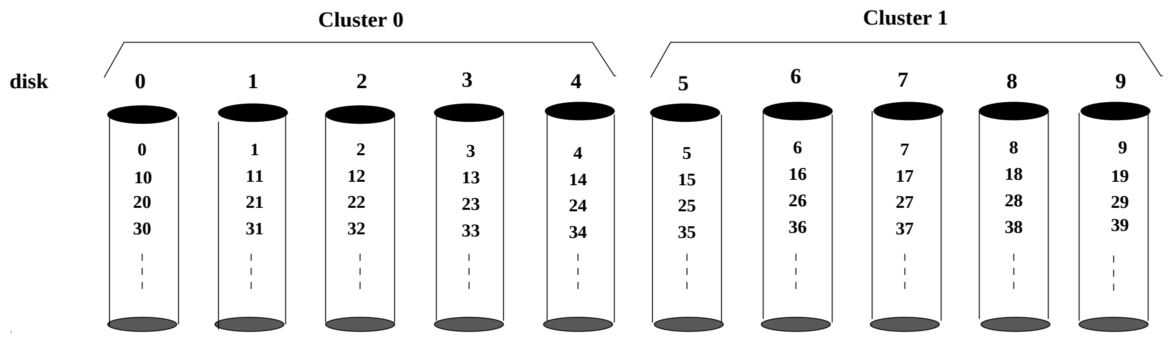


Figure 1: An example of object X striped on a multi-disk drive.

display of an object requires prefetching and at least two buffers with size M_X subobjects. One buffer is for retrieval of the next M_X consecutive subobjects while the other one is to store the previous retrieved M_X consecutive subobjects which are currently displayed. The real-time retrieval (i.e., continuous retrieval) can be achieved by satisfying following equations:

$$\begin{aligned}
&M_X \geq \lceil \tfrac{C_X}{d} \rceil, \\
&M_X \leq N, \\
&R = \lfloor \tfrac{N}{M_X} \rfloor, \\
&I = \frac{s_X}{\frac{C_X}{M_X}}, \\
&WS + WL + \tfrac{s_X}{d} \leq \frac{s_X}{\frac{C_X}{M_X}}.
\end{aligned}$$

Hence, display of X employs only a single cluster at a time in a round-robin manner. In each cluster, consecutive subobjects of object X are stored on these M_X disks in a liner order. Figure 1 shows an example of simple striping for an object X with bandwidth requirement $C_X = 80$ Mbps, where $N = 10$, $d = 20$ Mbps, $WS = 30$ ms (ms $= 10^{-3}$ seconds), $WL = 10$ ms and the value i denotes subobject X_i inside a disk. Suppose $M_X = 5$ $(\geq \frac{80}{20})$, then s_X $(= \frac{(WS+WL)\times d \times \frac{C_X}{M_X}}{d - \frac{C_X}{M_X}}) = 3.2$ Mbits (Mbits $= 10^6$ bits), $I = 0.2$ seconds and $R = 2$. Note that display of X first employs cluster 0 to read the subobjects X_0, X_1, X_2, X_3 and X_4 from disks 0, 1, 2, 3 and 4, respectively, into a buffer in the first time interval. In the second time interval, subobjects X_5, X_6, X_7, X_8 and X_9 are read into the other buffer from cluster 1. At the same time, subobjects X_0, X_1, X_2, X_3 and X_4 are displayed. Then, alternatively, the subsequent subobjects of X are read from cluster 0 or cluster 1 into two buffers and then are displayed.

Moreover, when the database consists of a combination of objects each with a different

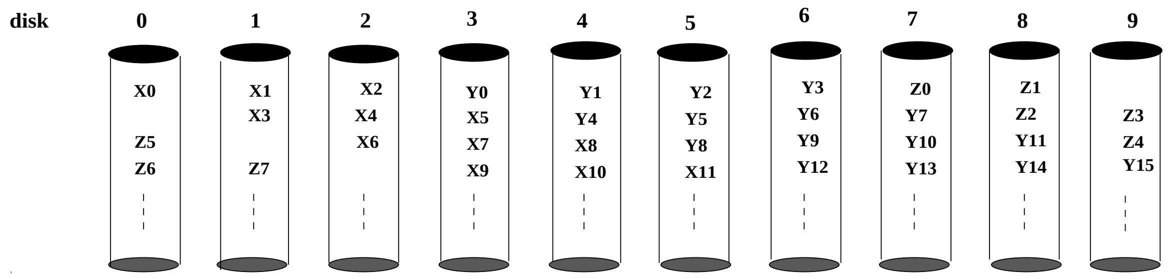


Figure 2: An example for *staggered striping* with a combination of 3 different objects.

bandwidth requirement, the design of simple *striping* is extended to minimize the percentage of wasted disk bandwidth by constructing the disk clusters based on the media type that has the highest bandwidth requirement. The percentage of waste disk bandwidth can be large. Therefore, Berson et al. [1] proposed a generalization of simple *striping*, called *staggered striping*, which constructs the disk clusters *logically* (instead of physically). It assigns the subobjects such that the disk containing subobject X_{i+M_X} is g disks (modulo the total number of disks) apart from the disk drive that contains subobject X_i. The objects with different bandwidth requirements are assigned to disk drives independently but all with the same value of k. Figure 2 illustrates the assignment of objects X, Y and Z, where $g = 1$, $M_X = 3$, $M_Y = 4$, $M_Z = 2$ and $N = 10$.

3 The Sliding-Window Approach

In this section, we will describe the basic idea of the proposed *sliding-window* and the *retrieval* algorithm for the proposed approach.

3.1 Basic Idea

Suppose the desired subobjects of these 3 *striped* objects for display are changed to those shown in Figure 3. Obviously, two conflicts will occur. One conflict occurs between subobjects Z_3 and Y_{15} and the other one occurs between subobjects Z_5 and X_{30}. To resolve these conflicts, a straightforward method is to reorganize these 3 objects according to the *striping* strategy. However, it requires a lot of overhead. A better solution as the one in [3, 20] is to prefetch the conflicting subobjects. For example, subobjects Z_3 and Z_5 are prefetched to resolve the conflicts in Figure 3.

Recall that based on the *striping* strategy, the duration (in terms of a *time interval* I) of retrieval of a subobject is fixed in each disk of the multi-disk drive. Logically, we can

time-interval \ disk	0	1	2	3	4	5	6	7	8	9
1	X0	X1	X2	Y0	Y1	Y2	Y3	Z0	Z1	
2		X3	X4	X5	Y4	Y5		Y7	Z2	Z3/Y15
3	Z5/X30			X7	X8	Y8	Y9	Y10	Y11	Z4

Figure 3: An example of display.

time-interval \ disk	0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1	0
2	0	1	1	1	1	1	0	1	1	2
3	2	0	0	1	1	1	1	1	1	1
⋮						⋮				

Figure 4: The *time table of retrieval* of Figure 3.

use a *time table of retrieval* to record the retrieved subobjects for each time interval. For example, the logical *time table of retrieval* (TT) of the display in Figure 3 is shown in Figure 4, where the value of each entry TT_{ij} denotes the number of subobjects that have to be retrieved from disk j in time interval i. For an entry TT_{ij} which value is greater than 1, a conflict will occur due to more than one subobjects that have to be retrieved in the same time interval i from disk j. To resolve the conflict, we have to prefetch (TT_{ij} - 1) subobjects of these TT_{ij} conflicting subobjects from disk j before time interval i. Logically, such *prefetching* operations can be viewed as a series of *replacement* operations, each of which is to find an entry TT_{kj} with value = 0 for such an entry TT_{ij} and then, TT_{kj} is set to 1 and TT_{ij} is decreased by one, where k is the maximum value such that $1 \leq k < i$. This *replacement* operation will be repeated until TT_{ij} is reduced to 1. Therefore, to guarantee continuous retrieval, we have to find (TT_{ij} - 1) entries with value = 0 for each such an entry TT_{ij} which value is greater than 1.

Let us consider another example, where the *time table of retrieval* is shown in Figure 5. The total length of this display (Len) is 10 time intervals. For each TT_{ij} which value is greater than 1, we perform the *replacement* operation. For example, one of these 2 conflicting subobjects in entry TT_{41} has to be prefetched in entry TT_{31}. As shown in Figure 5, continuous display can be guaranteed after all the conflicts are resolved by a

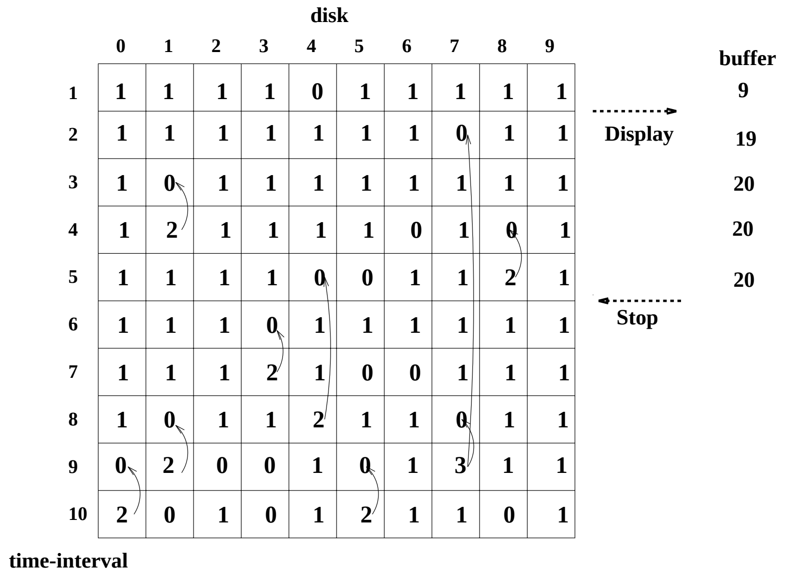


Figure 5: An example in the *prefetching* approach.

series of *replacement* operations. However, for an interactive display, users may stop this display at any time. In this example, suppose display is interrupted in time interval 5. That is, these retrieved subobjects in entry TT_{ij} for display after time interval 5 are no longer needed, where $6 \leq i \leq 10$ and $0 \leq j \leq 9$. In this case, there are one conflicting subobject of TT_{84} and one of TT_{97} which have been prefetched in time interval 5 and time interval 2, respectively. The disk bandwidths and buffers for retrieving these two subobjects are wasted. The corresponding sizes of the buffer (in terms of the number of subobjects) in each time interval for storing these retrieved subobjects are also shown in Figure 5.

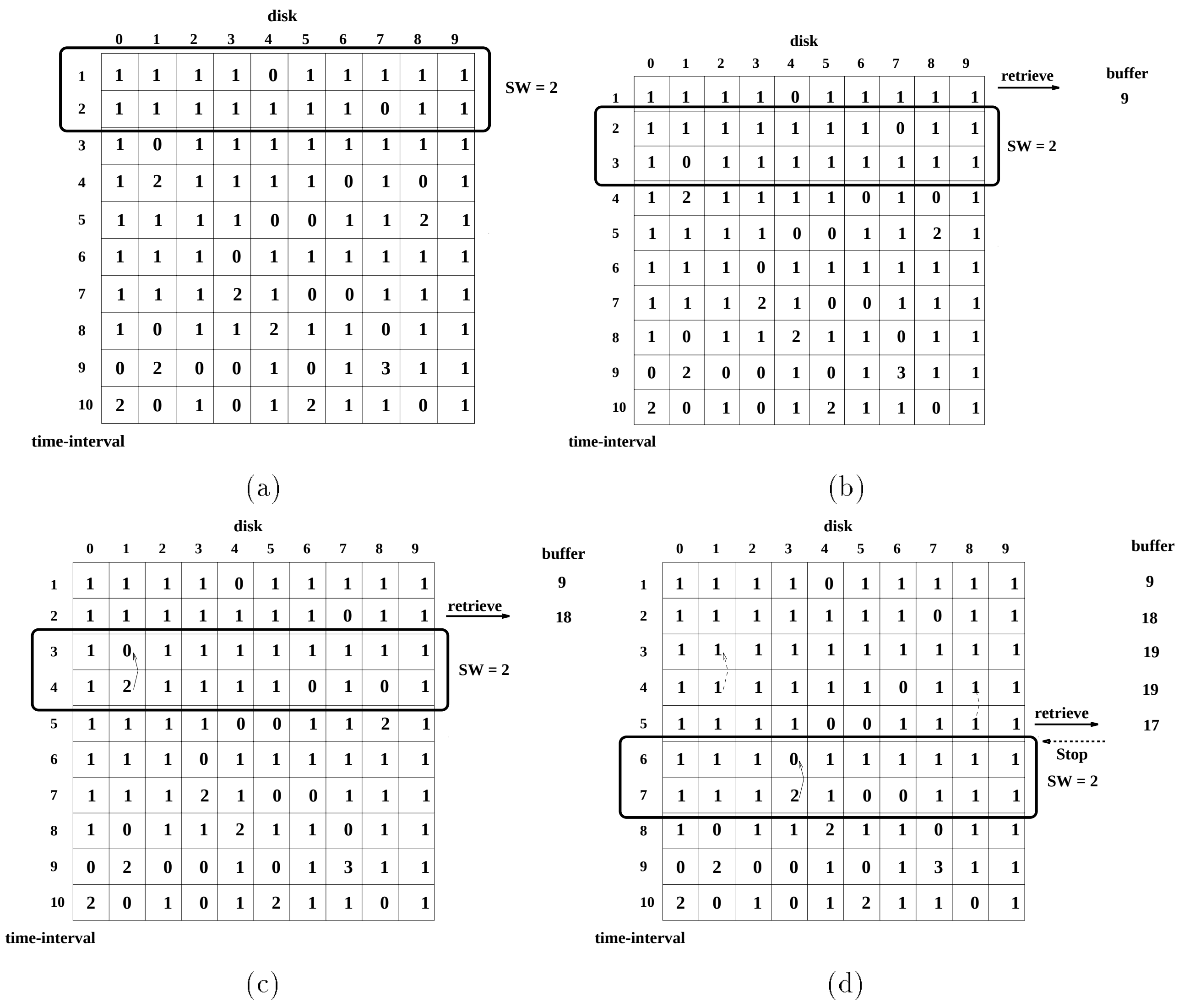
To avoid the waste of time to prefetch and the waste of buffer space to store these unnecessary prefetched subobjects as the example shown in Figure 5, in this paper, we propose a *sliding window* approach. The basic concept of a *sliding window* in our proposed approach is to record a little necessary information of retrieval of the following subobjects for display in a *sliding window*. In this proposed approach, first, we use a window with size = SW (≥ 2) time intervals to record the first SW consecutive entries for each disk in the *time table of retrieval*, i.e, time intervals 1, 2, ..., SW. Second, we only perform the *replacement* operations within the *sliding window*. (Note that when $SW = 1$, no *replacement* operation can be done for any entry with value > 1. Therefore, the minimun size of SW is 2.) After the possible conflicts within the *sliding window* have been resolved by a series of *replacement* operations, these subobjects in time interval 1 are ready to be retrieved for display and the *window* is slid forward by including time interval (SW

+ 1) and excluding time interval 1. Third, we resolve the conflicts within the *sliding window* again and then, slide the window forward. At the same time, these subobjects in time interval 2 are ready for retrieval. These *replacement* and *sliding* operations will be repeated until display is finished or interrupted.

For illustrative purpose, let us consider a simple example shown in Figures 6, where the objects for display are the same as the ones in Figure 5 and SW is assumed to be 2. (Note that we use $SW(a,b)$ to denote that the current *sliding window* includes time intervals a and b.) As shown in Figure 6-(a), first, no *replacement* operation is needed since all the entries in the *sliding window* ≤ 1. Second, we slide the *window* forward as shown in Figure 6-(b). At the same time, these subobjects in time interval 1 are ready to be retrieved and no *replacement* operation is needed within the current *sliding window*. Third, we slide the *window* again and perform a *replacement* operation for entry TT_{41} by setting TT_{31} and TT_{41} to be 1 as shown in Figure 6-(c). At the same time, these subobjects in time interval 2 are being retrieved and these subobjects that have been retrieved in time interval 1 are being displayed. Forth, we set TT_{58} and TT_{48} to be 1 for $TT_{58} = 2$ after the *sliding window* is slid again. When the *sliding window* has been slid forward to include time intervals 6 and 7 as shown in Figure 6-(d), suppose display is interrupted. In this case, we set TT_{63} and TT_{73} to be 1 for $TT_{73} = 2$. Since the current time interval for retrieval is time interval 5, this *prefetching* subobject for TT_{73} in TT_{63} has not be retrieved yet. Moreover, the waste of time and space to prefetch the conflicting subobjects for TT_{84} and TT_{97} in Figure 5 can be avoided.

From the above example, we observe that continuous retrieval can also be guaranteed by using the *sliding window* approach with $SW = 2$. Obviously, the larger the window size SW is, the longer the waste of time and the higher the buffer space to prefetch the unnecessary subobjects once display is interrupted. However, a *hiccup* can occur in the proposed *sliding window* approach when the window size SW is not large enough, where a *hiccup* means that the subobjects for being displayed has not been retrieved and will be ready in the next time interval. Therefore, from the view of users, display will not be continuous when a *hiccup* occurs.

Such an example is shown in Figure 7, where $SW = 2$. From Figure 7-(a), we observe that we can not find an entry $TT_{i1} = 0$ for TT_{41} to resolve the conflict within the *sliding window*. Note that even though $TT_{21} = 0$, we can not set TT_{21} $(= 0)$ to be 1 to prefetch one conflicting subobjects for TT_{41} in time interval 2 because that these subobjects in time interval 2 are being retrieved. Therefore, TT_{41} is still 2 after the *replacement* operation. Then, a conflict will occur such that one of these 2 conflicting subobjects for retrieval in



(a)

time-interval	0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	0	1	1	1	1	1
2	1	1	1	1	1	1	1	0	1	1
3	1	0	1	1	1	1	1	1	1	1
4	1	2	1	1	1	1	0	1	0	1
5	1	1	1	1	0	0	1	1	2	1
6	1	1	1	0	1	1	1	1	1	1
7	1	1	1	2	1	0	0	1	1	1
8	1	0	1	1	2	1	1	0	1	1
9	0	2	0	0	1	0	1	3	1	1
10	2	0	1	0	1	2	1	1	0	1

(b)

time-interval	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	1	1	1	1	1	1	0	1	1	
3	1	0	1	1	1	1	1	1	1	1	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	1	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	1	0	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	0	1	0	1	3	1	1	
10	2	0	1	0	1	2	1	1	0	1	

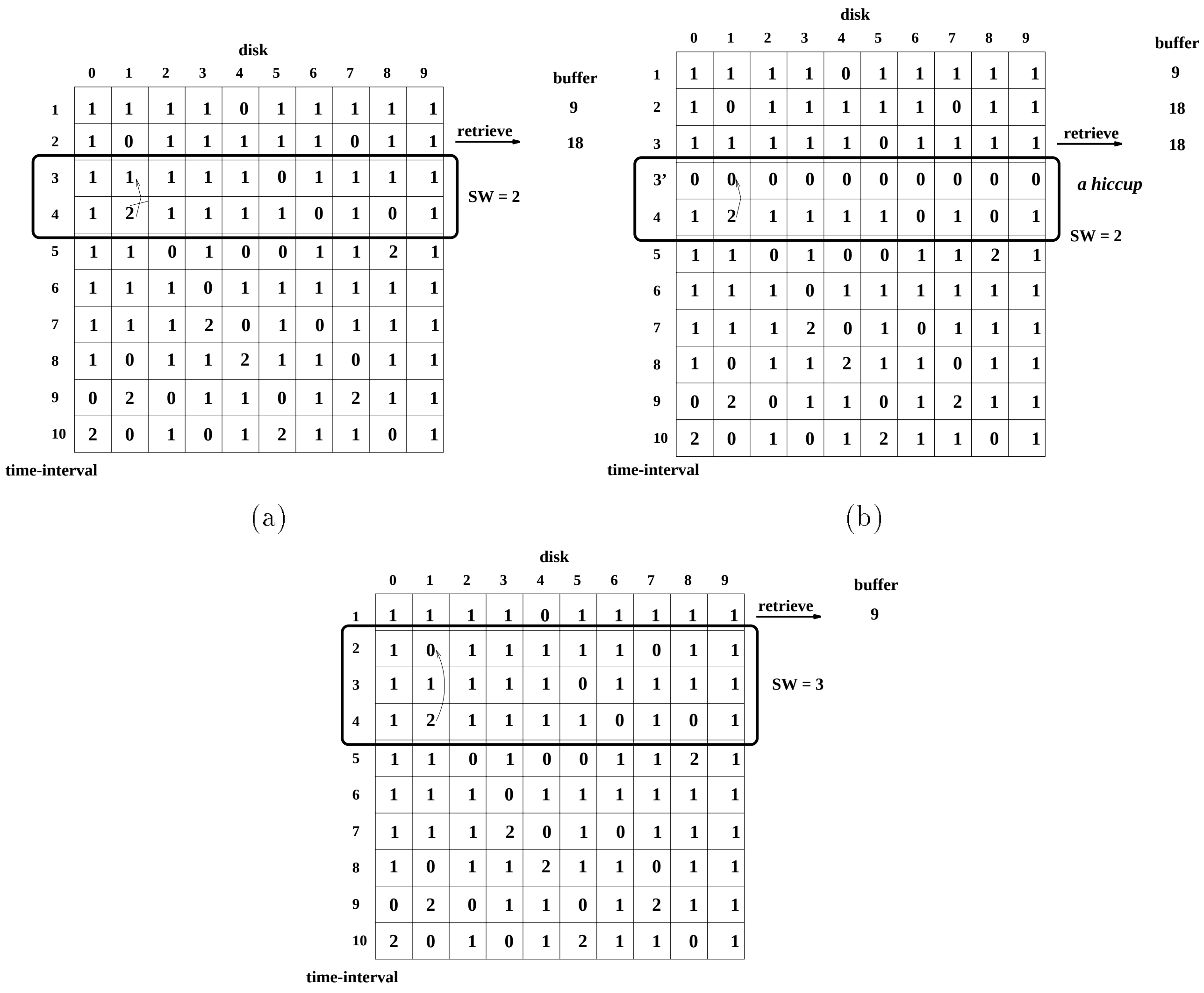
(c)

time-interval	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	1	1	1	1	1	1	0	1	1	18
3	1	0	1	1	1	1	1	1	1	1	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	1	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	1	0	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	0	1	0	1	3	1	1	
10	2	0	1	0	1	2	1	1	0	1	

(d)

time-interval	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	1	1	1	1	1	1	0	1	1	18
3	1	1	1	1	1	1	1	1	1	1	19
4	1	1	1	1	1	1	0	1	1	1	19
5	1	1	1	1	0	0	1	1	1	1	17
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	1	0	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	0	1	0	1	3	1	1	
10	2	0	1	0	1	2	1	1	0	1	

Figure 6: An example in the *sliding window* approach with $SW = 2$: (a) $SW(1,2)$; (b) $SW(2,3)$; (c) $SW(3,4)$; (d) $SW(6,7)$.



(a)

time-interval	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	0	1	1	1	1	1	0	1	1	18
3	1	1	1	1	1	0	1	1	1	1	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	0	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	0	1	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	1	1	0	1	2	1	1	
10	2	0	1	0	1	2	1	1	0	1	

(b)

time-interval	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	0	1	1	1	1	1	0	1	1	18
3	1	1	1	1	1	0	1	1	1	1	18
3’	0	0	0	0	0	0	0	0	0	0	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	0	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	0	1	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	1	1	0	1	2	1	1	
10	2	0	1	0	1	2	1	1	0	1	

(c)

time-interval	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	0	1	1	1	1	1	0	1	1	
3	1	1	1	1	1	0	1	1	1	1	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	0	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	0	1	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	1	1	0	1	2	1	1	
10	2	0	1	0	1	2	1	1	0	1	

Figure 7: An example in the *sliding window* approach: (a) with $SW = 2$; (b) *time interval stealing*; (c) with $SW = 3$.

entry TT_{41} will be lost when these subobjects in time interval 4 are being retrieved. To resolve this problem, one proposed solution is called *time interval stealing*, which inserts a new time interval $3'$ with all $TT_{3'j} = 0$ $(0 \leq j < 10)$ between time intervals 3 and 4 and slides the window forward as shown in Figure 7-(b). Now, we can set $TT_{3'1}$ to be 1 for TT_{41}. However, a *hiccup* will occur. Another better proposed solution is to select a large size of the *sliding window* $SW = 3$, instead of $SW = 2$, initially. As shown in Figure 7-(c), we can set TT_{21} to be 1 for TT_{41}. Moreover, by applying the *sliding window* approach with $SW = 3$, continuous display can be guaranteed. Therefore, to select a proper size of the *sliding window* for display is an important task and will be investigated in Sections 4 and 5.

3.2 The Algorithm

In this subsection, we present the *retrieval* algorithm based on the *sliding window* approach as shown in Figure 8. Suppose the execution time for the *replacement* process in each for-loop routine of the *retrieval* algorithm is negligible compared with the time interval for retrieval (e.g., 0.2 seconds in Figure 1). Moreover, the *retrieval* algorithm is *preemptive*. That is, the execution of the *retrieval* algorithm can be interrupted whenever display is interrupted. In the *retrieval* algorithm, after given the value of the *sliding window* size and the *time table* for display, we logically put the first SW time intervals of the *time table* into the *sliding window* and resolve the conflicts within the *sliding window*. Then, we slide the window forward by excluding time interval 1 and including time interval $(SW + 1)$. Let *ptr* be 1, where *ptr* denotes the identification number of time interval, in which these subobjects are ready for retrieval. In the for-loop routine, such a *resolving-sliding* operation will be repeated until display is finished or interrupted. For each for-loop operation, first, these subobjects in time interval *ptr* are ready for retrieval, where time interval *ptr* is just removed from the *sliding window*. In other words, these entries TT_{ptrj} $(0 \leq j < N)$ in time interval *ptr* is no longer to be changed because that time interval *ptr* is not included in the *sliding window*. Second, for each $TT_{ij} > 1$ in time interval i, we find (TT_{ij} - 1) entries with values = 0 within the *sliding window* and set them to be 1 to resolve the conflicts. However, there may not exist enough entries with values = 0 for each $TT_{ij} > 1$. In this case, we have to *steal* some time intervals. Third, after the conflicts in time interval i are resolved, we slide the window again.

```
procedure retrieval ($SW$,$TT[Len][N]$);
var
        $N$ : integer; /* the number of disks in a multi-disk drive*/
        $Len$ : integer;
                /* the length of display in terms of the number of time intervals */
        $SW$ : integer; /* the size of a sliding window*/
        $TT[Len][N]$ : integer; /* the time table of retrieval */
        $Buf$ : a buffer; /* the buffer space for storing the retrieved subobjects */
        $i$,$j$,$k$,$m$,$flag$,$ptr$ : integer;
begin
        resolve any conflict in the first $SW$ time intervals by using the replacement approach;
        slide the window forward;
        $ptr = 1$;
        for ($i$=$SW$+1;$i$<=$Len$;$i$++) do
                retrieve $TT[ptr][*]$ for display;
                $ptr = ptr + 1$;
                for ($j$=0;$j$<$N$;$j$++) do
                    $flag = 0$;
                    while (($TT[i][j] > 1$) and ($flag == 0$)) do
                      begin
                        for ($k$=$i$-1;$k$>$i$-$SW$;$k$- -) do
                            if ($TT[k][j] == 0$)
                              begin
                                $TT[k][j] = 1$;
                                $TT[i][j] = TT[i][j] - 1$;
                                $k = -1$;
                            end if;
                        end for;
                        if ($k$ <> -1) $flag = 1$;
                    end while;
                end for;

          /***** steal time intervals to resolve the unresolved conflicts *****/
                if ($flag == 1$)
                  begin
                    $m = max(TT[i][*] - 1)$;    /* function max returns the maximun value */
                    insert $m$ new time intervals between time intervals ($i$ - 1) and $i$
                    in the time table of retrieval;
                    $Len = Len + m$;
                    $i = i + m - 1$;
                end if;
          /*****—————————————————————————————————*****/

                slide the window forward;
        end for;
        for ($i$=$ptr$;$i$<=$Len$;$i$++) do retrieve $TT[i][*]$ for display;
end;
```

Figure 8: Algorithm *retrieval*.

4 Simulation Results

In this section, we will present simulation results for the proposed algorithm based on the *sliding window* approach. In this simulation model, we assume that each disk in a multi-disk drive operates independently. When an I/O request arrives, it may be decomposed into subrequests, each of which will be serviced independently on a different disk. Objects are stored on the multi-disk drive by applying the *striping* strategy and all the subobjects of each object have the same size. Moreover, the duration of retrieval of a subobject is fixed for all objects and is in terms of a *time interval* I. At any time interval i, the required bandwidth RB_i for display should not be larger than the aggregate bandwidth AB of the multi-disk drive. The required bandwidth RB_i can be varied according to the combination of objects for display. To describe the desired display, we propose a data model. In this data model, first, we use a $load\_factor$ to denote the average load of a multi-disk drive for a display with length Len (in terms of the number of time intervals) and let $load\_factor$ be $\frac{\sum_{i=1}^{Len} RB_i}{AB \times Len}$. Second, to describe the status of conflicts in a display, we use a series of probabilities P_k^n $(0 \leq k \leq n)$, each of which is to denote the probability of an entry with value $= k$ when the desired display is combined with n objects. Consequently, $\sum_{k=0}^{n} P_k^n = 1$ and $\sum_{k=0}^{n} (k \times P_k^n) = load\_factor$. The performance measure is the average hiccup ratio ave_HR, which is the number of the number of *hiccups* num_HR divided by the length of display Len, that is, $ave\_HR = \frac{num\_HR}{Len}$. Another performance measure is the average size of buffer ave_Buf (in terms of the number of subobjects) that is used to store the retrieved subobjects during the duration of display.

Figures 9 show the relationship between the size of a *sliding window* (SW) for three different displays (D_1, D_2 and D_3) and the average hiccup ratio (ave_HR), where $N = 10$, $n = 3$ with $Len = 1000$ and $load\_factor$ is 0.7. The *time tables of retrieval* of three different displays are randomly generated, where (P_0^3, P_1^3, P_2^3, P_3^3) is (0.35, 0.6, 0.05, 0), (0.42, 0.47, 0.1, 0.01) and (0.44, 0.44, 0.1, 0.02), respectively. From this figure, we observe that ave_HR is decreased as the size of *sliding window* SW is increased. The reason is that the probability of a *hiccup* is decreased as SW is increased. Moreover, ave_HR is increased as the number of conflicts is increased (i.e., P_3^3 and P_2^3 is increased; while P_1^3 is decreased). To support continuous display with *hiccup*-free (i.e., $ave\_HR = 0$), the minimum sizes of the *sliding window* for these 3 display D_1, D_2 and D_3 are 42, 44 and 50, respectively. However, the users may choose a small SW with a tolerable average hiccup ratio to reduce to overhead once display is interrupted.

Figure 10 shows the relationship between the size of a *sliding window* (SW) for three

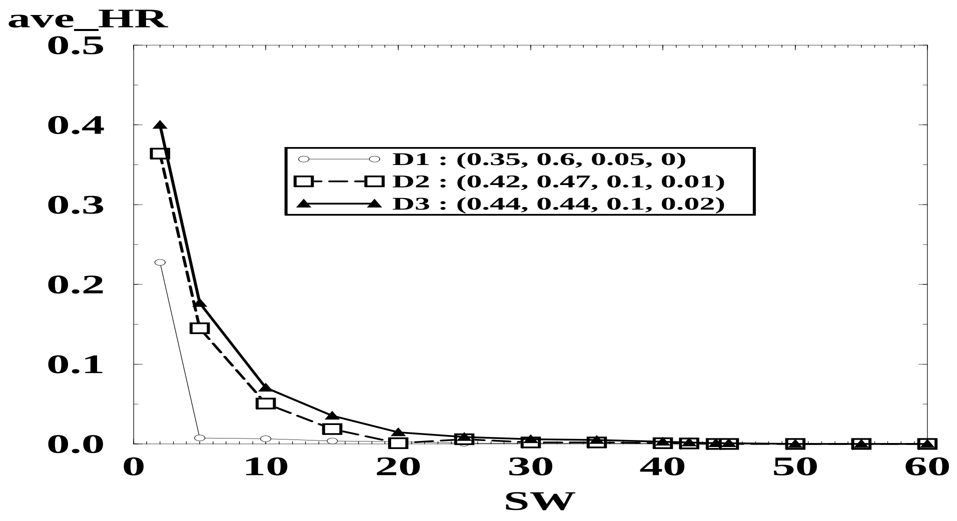


Figure 9: The relationship between the size of a *sliding window* (SW) and the average hiccup ratio (ave_HR).

different displays (D_1, D_2 and D_3) and the average buffer size (ave_Buf) used in Figure 9. Obviously, the larger the value of SW is, the larger the value of ave_Buf. The reason is that it requires a larger buffer space to store these prefetched subobjects within a larger *sliding window* than a smaller one within a smaller *sliding window*. Even though ave_Buf is also increased as $load\_factor$ is increased, ave_Buf is still constant when SW is large enough such that ave_HR is reduced to 0.

5 Analysis of an Sliding Window Size

In the *sliding window* approach, we have to select a proper value of SW to support continuous display. However, such a selection will be made by repeating a series of imitations of the *retrieval* algorithm from an initial value of $SW = 2$ as described in Figures 9 and 10; it will waste much time when P_k^n ($1 < k \leq n$) is large. For example, it requires to perform 42 times imitations for D_1; while it requires to perform 50 times imitations for D_3 in Figure 9. Therefore, to speed up such a process, in this section, we will present the mathematical analysis of average hiccup ratio ave_HR with a given value of SW in the *sliding window* approach. Moreover, given a tolerable value of ave_HR, we can analyze the minimum value of SW for display of a combination of n objects. Consequently, instead of $SW = 2$, a good initial value of SW can be obtained from the mathematical analysis in order to speed up the selection of a proper value of SW.

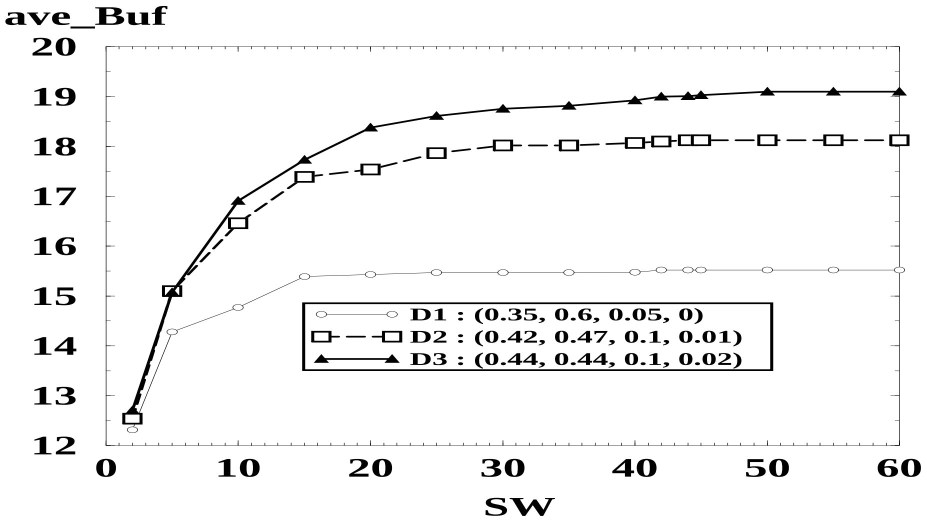


Figure 10: The relationship between the size of a *sliding window* (SW) and the average buffer size (ave_Buf).

Suppose there is a combination of n striped objects for display by using the *sliding window* approach with SW (≥ 2). The values of $load\_factor$ and the series of P_k^n ($0 \leq k \leq n$) are given. Assume that the probability of k subobjects that will be retrieved from any disk j in any time interval i has the same value P_k^n, where $0 \leq k \leq n$. In other words, the probabilities of all TT_{ij} with value = k are P_k^n. The subobjects that will be retrieved in each time interval are independent. Therefore, there are (n + 1) cases of the value of an entry TT_{ij}. Since the conflict-resolution process (i.e., the *replacement* operation) is performed from the initial time interval to the last one in the *sliding window* approach, the subobjects in time interval e which are ready for retrieval implies that those in time interval m are also ready for retrieval, where $1 \leq m < e$. Consequently, the value of each entry in these time intervals that are ready for retrieval will be either 0 or 1. The probability of such an entry with value = 0 is (1 - $load\_factor$); while the one of such an entry with value = 1 is $load\_factor$. Therefore, for a time interval i that is in the conflict-resolution process, the probabilities for an entry TT_{ij} with values = 0 and 1 are P_0^n and P_1^n, respectively. In these two case, no conflict and hiccup can occur in entry TT_{ij}. When TT_{ij} = 2, a hiccup can occur if there does not exist any one entry TT_{mj} = 0, where (i - SW + 1) $\leq m \leq$ (i - 1). The probability of such a case is $\binom{SW-1}{SW-1} f^{SW-1}$. Otherwise, no hiccup can occur. To simplify the notations, in the following formulas, we use f to denote $load\_factor$. The number of hiccup with TT_{ij} = 2 is obtained as

$$UH_2^n = P_2^n \times (1 \times \binom{SW-1}{SW-1} f^{SW-1}).$$

Similarly, the number of hiccup with $TT_{ij} = 3$ can be obtained as

$$UH_3^n = P_3^n \times (2 \times \binom{SW-1}{SW-1} f^{SW-1} + 1 \times \binom{SW-1}{SW-2} f^{SW-2}(1\text{-}f)).$$

In general, the number of hiccup with $TT_{ij} = k$ can be obtained as

$$UH_k^n = P_k^n \times ((k \text{ - } 1) \times \binom{SW-1}{SW-1} f^{SW-1} + (k \text{ - } 2) \times \binom{SW-1}{SW-2} f^{SW-2}(1\text{-}f) + \ldots + 1 \times \binom{SW-1}{SW-k+1} f^{SW-k+1}(1\text{-}f)^{k-2}).$$

Therefore, the average number of hiccup can be obtained as

$$ave\_HR = \sum_{k=2}^{n} UH_k^n.$$

6 The Dynamic Sliding Window Approach

In the *sliding window* approach, the combination of objects for display and the branch points for choices are predetermined. That is, the *time table of retrieval* has to be predetermined. To support *on-line* interactive display, in this section, we will extend the *sliding window* approach to the *dynamic sliding window* approach, which can support *on-line* interactive display for any combination of objects by applying a *dynamic window size*. The size of the *sliding window* is changed according to the future requirements of data for display. The basic concept is that display can be interrupted or may eventually be changed to another display by the users at any time. That is, the contents of the *time table of retrieval* can be dynamically changed. Therefore, we have to dynamically change the size of the *sliding window* according to the current status of subobjects for display in order to still support continuous retrieval with the a little overhead. Since the subobjects for future display are not predictable, in the *dynamic sliding window* approach, we use the PI ($\geq$ 1) pervious time intervals of retrieval to guess the subobjects for future display. Then, the size of the *sliding window* SW for the next time interval i is chosen to be the minimum value of SW for these PI time intervals with $num\_HR = 0$ by applying the *sliding window approach.*

The retrieval algorithm based on the *dynamic sliding window* approach is called the *retrieval** algorithm as shown in Figure 11. This *retrieval** algorithm is also preemptive. The differences between the *retrieval** algorithm and the *retrieval* algorithm are printed in bold font as shown in Figure 11. In the *retrieval** algorithm, after given the value of PI and an initial value of SW, we logically put the first SW time intervals of the *time*

```
procedure retrieval* (PI,TT[Len][N]);
var
    N,Len,PI,SW,CSW : integer;
    TT[Len][N] : integer; /* the time table of retrieval */
    WT[PI][N] : integer;
    Buf : a buffer; /* the buffer space for storing the retrieved subobjects */
    i,j,k,m,r,flag,ptr,I_flag,P_flag : integer;
begin
    SW = 2;        WT[*][*] = 0;
    resolve any conflict in the first SW time intervals;
    slide the window forward;
    for (r=1;r≤SW;r++) do WT[ptr-SW+r][*] = TT[r][*];
    ptr = 1;
    for (i=SW+1;i<=Len;i++) do
        for ( r=1;r<PI;r++) do WT[r][*] = WT[r+1][*];
        WT[PI][*] = TT[i][*];
        if (I_flag == 0)
          begin
            retrieve TT[ptr][*] for display;
            ptr = ptr + 1;
        end if;
        for (j=0;j<N;j++) do
            flag = 0;
            while ((TT[i][j] > 1) and (flag == 0)) do
              begin
                for (k=i-1;k>i-SW;k- -) do
                    if (TT[k][j] == 0)
                      begin
                        TT[k][j] = 1;
                        TT[i][j] = TT[i][j] - 1;
                        k = -1;
                    end if;
                end for;
                if (k <> -1) flag = 1;
            end while;
        end for;
        if (flag == 1)
          begin
            m = max(TT[i][*] - 1);
            insert m new time intervals between time intervals (i - 1)
            and i on the time table of retrieval;
            Len = Len + m;
            i = i + m - 1;
        end if;
        P_flag=0;
        for (r=2;r<=PI;r++) do
            imitate retrieval(r,WT[PI][N]);
    /* perform the retrieval algorithm without the retrieval operations;
       we only want to get the number of hiccups after
       the retrieval algorithm is applied on WT */
            if (the number of hiccups in the imitation is 0)
              begin
                CSW = r;
                P_flag = 1;
            end if;
        end for;
        if (P_flag == 0) CSW = PI;
        if (CSW<SW)
          begin
            for (j=CSW;j<SW;j++) do
                retrieve TT[ptr][*] for display;
                ptr = ptr + 1;
            end for;
            SW = CSW;
        end if;
        if (CSW>SW) { I_flag = 1; SW = SW + 1;}
        if (CSW=SW) I_flag = 0;
        slide the window forward;
    end for;
    for (i=ptr;i<=Len;i++) do retrieve TT[i][*] for display;
end;
```

Figure 11: Algorithm *retrieval**.

table into the *sliding window* and resolve the conflicts within the *sliding window*. (Note that as opposed to the predetermined *time table* in the *retrieval* algorithm, the one in the *retrieval** algorithm is dynamically determined.) Then, we slide the window forward. In the for-loop routine, such a *resolving-sliding* operation will be repeated until retrieval for display is finished or interrupted. For each for-loop operation, first, these subobjects in time interval *ptr* are ready for retrieval as the case in the *retrieval* algorithm. Second, we put the information of retrieval of previous PI time intervals into the working table WT. Then, for each $TT_{ij} > 1$ in time interval i, we find (TT_{ij} - 1) entries with values = 0 within the *sliding window* and set them to be 1 to resolve the conflicts. However, there may not exist enough entries with values = 0 for each $TT_{ij} > 1$. In this case, we have to *steal* some time intervals. Third, after the conflicts in time interval i are resolved, we have to predict a *sliding window* size CSW for the next time interval by imitating the *retrieval* algorithm with WT. In this imitation, we find the minimum value of *sliding window* size (CSW) for WT with *hiccup*-free. When the new *sliding window* size CSW is smaller than SW, we have to retrieve the subobjects in these time intervals *ptr*, (*ptr* + 1), ..., (*ptr* + SW - CSW - 1). The reason is that the size of *sliding window* will be reduced and these previous time intervals that will be removed from the *sliding window* are no longer to be changed. Therefore, the subobjects in these time intervals are ready for retrieval. On the other hand, when $CSW > SW$, we have to enlarge the *sliding window*. In this case, time intervals (*ptr* - 1), (*ptr* - 2), ..., (*ptr* - (CSW - SW - 1)) have to be included in the *sliding window*. However, since these subobjects in these time intervals before time interval *ptr* had been retrieved, we can not change the values of these entries in these time intervals to resolve any conflict. Therefore, in this case, the size of the *sliding window* is only increased by one, in which time interval *ptr* is not removed from the *sliding window* and time interval (i + 1) is included in the *sliding window*. That is, the *sliding window* size for the next time interval is (SW + 1). *I_flag* is set to 1 to denote that such a case occurs and to prohibit the retrieval operation of time interval *ptr*. Finally, we slide the window again.

Figures 12-(a), 12-(b) and 12-(c) show the relationship between the time interval t and the number of hiccups *num_HR*, where N = 10, n = 3 with *Len* = 1000 and PI is 10, 20 and 50, respectively. The *load_factor* and the series of P_k^3 $(0 \leq k \leq 3)$ are varied in each time interval. To simulate the unpredetermined subobjects for *on-line* interactive display, we use a random function of t to generate the values of the *load_factor* and the series of P_k^3 $(0 \leq k \leq 3)$ for each time interval. From these figures, we observe that *num_HR* is decreased as PI is increased due to that the more the information of retrieval are considered, the better the value of SW.

Figures 13-(a) and 13-(b) show the relationship between the time interval t and the size of buffer Buf, where the related parameters are the same as those in Figure 12. Since there are similar results in the other ranges of t, Figures 13-(a) and 13-(b) only show the range of t from 300 to 440 and from 440 to 500, respectively. From Figures 12 and 13, we observe that the size of buffer Buf is increased as num_HR is increased in all these 3 cases. The larger the value of PI is, the larger the size of buffer. Moreover, the corresponding value of SW for each time interval t in Figure 12 is also shown in Figures 14. Compared to Figure 12, we observe that the *sliding window* is changed according to the retrieval information of the previous PI time intervals. The larger the value of PI is, the better the value of SW. However, the overhead to decide a new value of SW is increased as the value of PI is increased. Therefore, based on the *dynamic sliding window* approach, users can choose a proper PI with a tolerable average hiccup ratio and overhead.

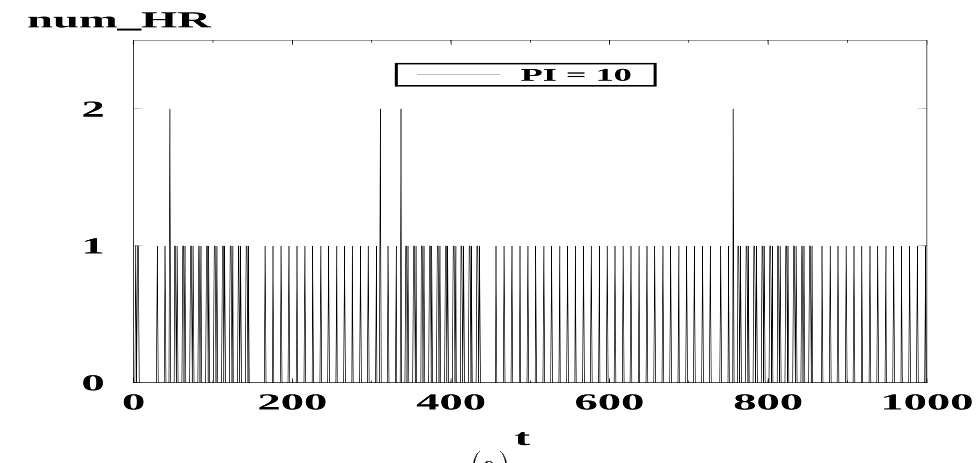
7 Conclusion

In this paper, we have proposed an efficient approach, called the *sliding window* approach, which can support interactive display for continuous media with a little overhead of $prefetching$. From the simulation results, we have observed that the smaller the size of a *sliding window* is, the smaller the waste of time and space once display is interrupted. However, a *hiccup* can occur when the size of the *sliding window* is not large enough. Moreover, we have presented a mathematical analysis of the *sliding window* approach to speed up the selection of a *sliding window* size. To support *on-line* interactive display, we have extended the *sliding window* approach to the *dynamic sliding window* approach. From the simulation results, we have observed that the probability of a *hiccup* is decreased as the amount of information of previous retrieval is increased. How to support *on-line* interactive display for continuous media at any desired display speed rate is a future research direction.

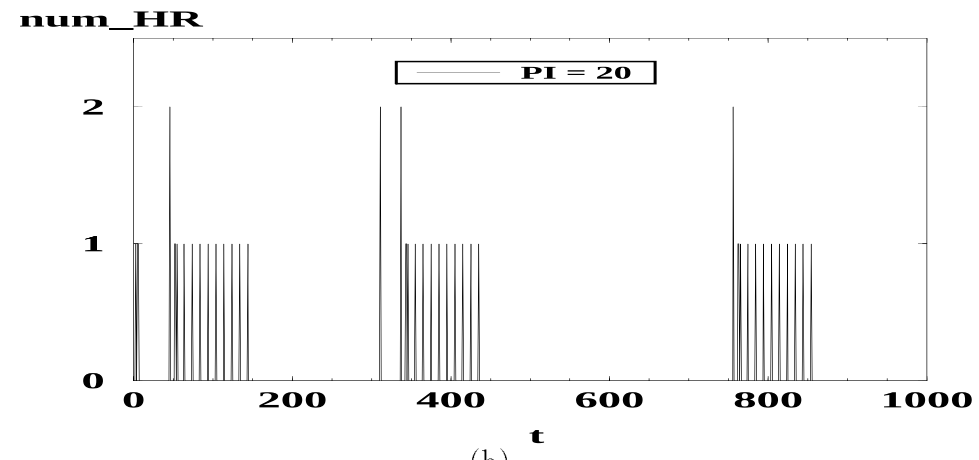
References

[1] Berson, Steven, Ghandeharizadeh, Shahram, Muntz, Richard and Ju, Xiangyu, "Staggered Striping in Multimedia Information Systems," *ACM SIGMOD*, pp. 79-90, 1994.

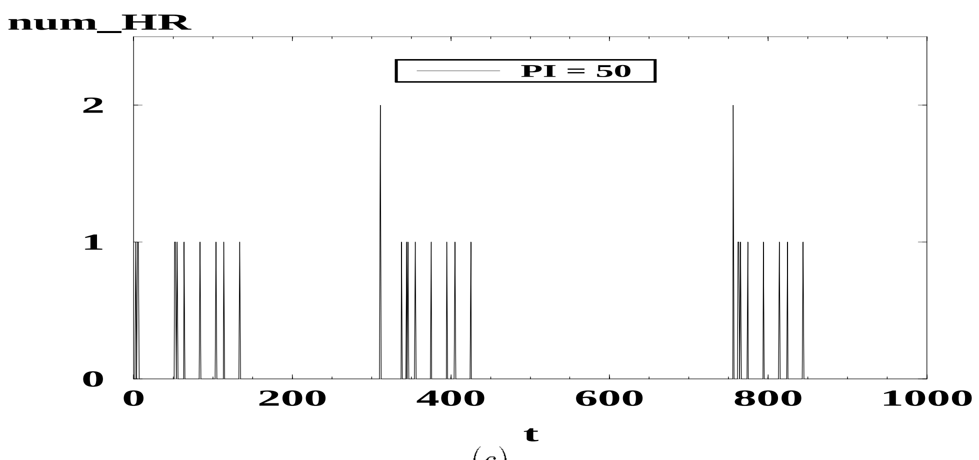
[2] Buford, John F. K., "Multimedia File Systems and Information Models," *Multimedia Systems*, Buford, John F. K., ed., Addison-Wesley, 1994.



(a)

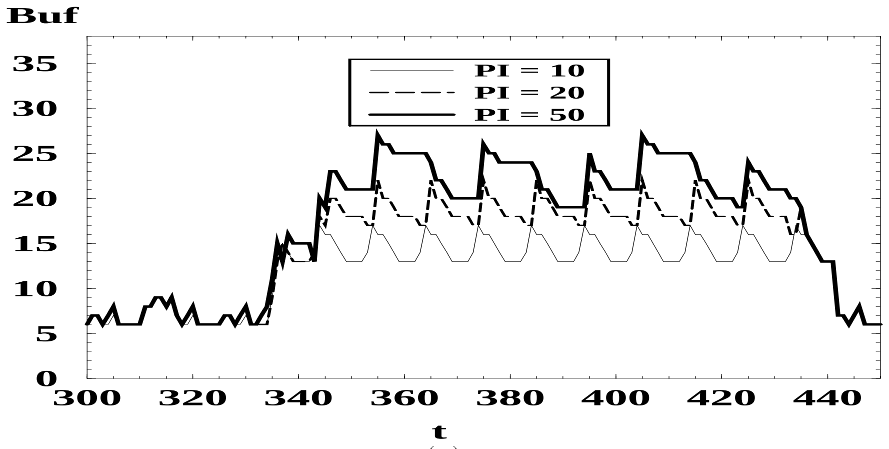


(b)

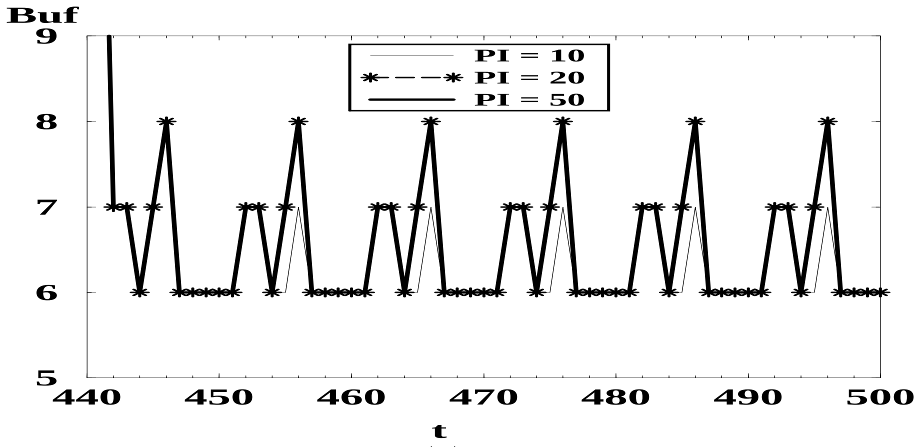


(c)

Figure 12: The relationship between the time interval t and the number of hiccups num_HR: (a) $PI = 10$; (b) $PI = 20$; (c) $PI = 50$.



(a)



(b)

Figure 13: The relationship between the time interval t and the size of buffer Buf: (a) t = (300, 440); (b) t = (440, 500).

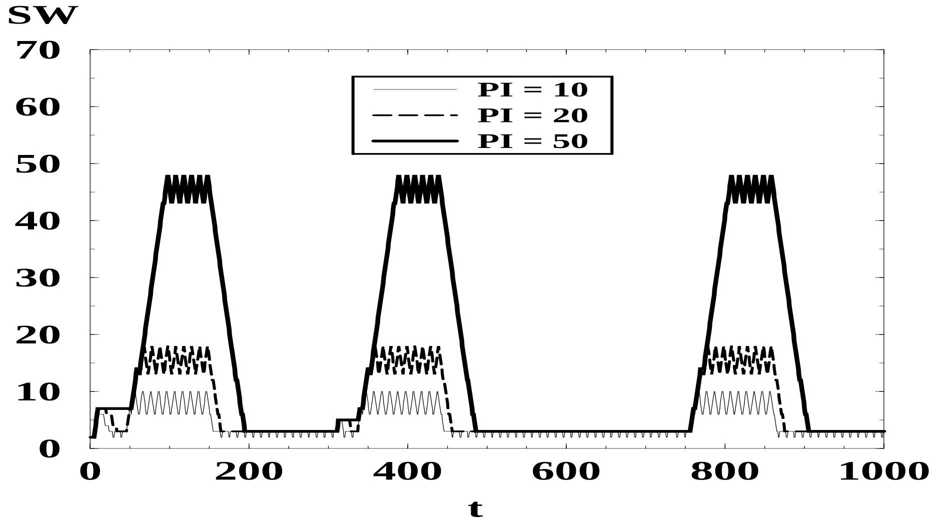


Figure 14: The relationship between the size of the time interval t and the *sliding window* SW.

[3] Chaudhuri, Surajit, Ghandeharizadeh, Shahram, and Shahabi, Cyrus, "Avoiding Retrieval Contention for Composite Multimedia Objects," *Proc. of the 21st VLDB Conference*, pp. 287-298, 1995.

[4] Chen, Ming-Syan, Kandlur, Dilip D. and Yu, Philip S., "Storage and Retrieval Methods to Support Fully Interactive Playout in a Disk-Array-Based Video Server," *ACM Multimedia Systems*, Vol. 3, pp. 126-135, 1995.

[5] Christodoulakis, S. and Koveos, L., "Multimedia Information Systems: Issues and Approaches," in *Modern Database Systems: the Object Model, Interoperability and Beyond*, Kim, W., Editor, Addison-Wesley, 1994.

[6] Gemmell, Jim and Christodoulakis, Stavros, "Principles of Delay-Sensitive Multimedia Data Storage and Retrieval," *ACM Transactions on Information Systems*, Vol. 10, No. 1, pp 51-90, Jan. 1992.

[7] Gemmell, D. James, Vin, Harrick M., Kandlur, D. D., Rangan, P. Venkat and Rowe, L. A., "Multimedia Storage Servers: A Tutorial," *IEEE Computer*, pp. 40-49, May 1995.

[8] Ghandeharizadeh, Shahram and Dewitt, D., "A Multiuser Performance Analysis of Alternative Declustering Strategies," *Proc. of IEEE International Conference on Data Engineering*, pp. 466-475, 1990.

[9] Ghandeharizadeh, Shahram and Ramos, Luis, "Continuous Retrieval of Multimedia Data Using Parallelism," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, pp. 658-669, August 1993.

[10] Keeton, Kimberly, and Katz, Randy H., "Evaluating Video Layout Strategies for a High-Performance Storage Server," *ACM Multimedia Systems*, Vol. 3, pp. 43-52, 1995.

[11] Liu, Jonathan C. L., Du, David H. C. and Schnepf, James A., "Supporting Random Access on Real-Time Retrieval of Digital Continuous Media," *Computer Communications*, Vol. 18, No. 3, pp. 145-159, March 1995.

[12] Lougher, P. and Shepherd, D., "The Design of a Storage Server for Continuous Media," *The Computer Journal*, Vol. 36, No. 1, pp. 33-42, 1993.

[13] Mourad, Antoine N., "Issues in the Design of a Storage Server for Video-On-Demand," *ACM Multimedia Systems*, Vol. 4, pp. 70-86, 1996.

[14] Ozden, Banu, Rastogi, Rajeev, and Silberschatz, Avi, "On the Design of a Low-Cost Video-On-Demand Storage System," *ACM Multimedia Systems*, Vol. 4, pp. 40-54, 1996.

[15] Patterson, D., Gibson, G. and Katz, R., "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *ACM SIGMOD*, pp. 109-116, 1988.

[16] Rangan, P. Venkat, and Vin, Harrick M., "Designing File Systems for Digital Video and Audio," *Proc. 13th ACM Symposium on Operating System Principles*, pp. 81-94, 1991.

[17] Rangan, P. Venkat, Vin, Harrick M. and Ramanathan, Srinivas, "Designing an On-Demand Multimedia Service," *IEEE Communications Magazine*, pp. 56-64, July 1992.

[18] Rangan, P. Venkat and Vin, Harrick M., "Efficient Storage Techniques for Digital Continuous Multimedia," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, pp. 564-573, August 1993.

[19] Salem, K. and Carcia-Molina, H., "Disk Striping," *Proc. of IEEE International Conference on Data Engineering*, pp. 336-342, 1986.

[20] Shahabi, Cyrus, and Ghandeharizadeh, Shahram, "Continuous Display of Presentations Sharing Clips," *ACM Multimedia Systems*, Vol. 3, pp. 76-90, 1995.

[21] Steinmetz, R., "Multimedia File Systems Survey: Approaches for Continuous Media Disk Scheduling," *Computer Communications*, Vol. 18, No. 3, pp. 133-144, March 1995.

[22] Vin, Harrick M. and Rangan, P. Venkat, "Designing a Multiuser HDTV Storage Server," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 1, pp. 153-164, Jan. 1993.

[23] Yu, Clement, Sun, Wei, Bitton, Dina, Yang Qi, Bruno, Richard and Tullis, John, "Efficient Placement of Audio Data on Optimal Disks for Real-Time Applications," *Communications of ACM*, Vol. 32, No. 7, pp. 862-871, July 1989.