An Efficient Conflict-Resolution Approach to Support Read/Write Operations in a Video Server [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

In this paper, we propose an efficient *conflict-resolution* approach based on the multi-disk architecture for the insertion/deletion operations on continuous media that are split up into blocks and placed in various locations on the disk, without reorganizing the whole data. When a new subobject is inserted after subobject i, it will be assigned with an identification number $(i + 1)$ and be inserted to a disk in which the retrieval of the new subobject does not conflict with the retrieval of any other subobject, where a *conflict* means a pair of two consecutive subobjects that are stored in the same disk have to be retrieved simultaneously. However, a new conflict on the same disk may occur since all the identification numbers of subobjects after subobject i are increased by one. Only when such a new conflict occurs, one movement operation is required, so does the case of a deletion operation. Moreover, to reduce those additional movement cost, a *deferring* approach is proposed at the cost of an additional buffer. In this approach, n data insertions are deferred and stored in a buffer. Then, the system starts to insert those data after an optimal insertion sequence is determined. Based on this approach, two strategies are proposed: the *conflict-resolved-first-deferring* strategy (the $CRFD$ strategy) and the *conflict-resolved-last-deferring* strategy (the $CRLD$ strategy). From our performance analysis, we will prove that the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers is an optimal strategy based on the proposed *deferring* approach.

(*Key Words: data placement strategies, digital continuous media, multi-disk drive , random access, 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

People frequently apply the term "multimedia" to almost any combination of text, graphics, animation, sound, and video. Some media (such as audio and video) are classified as *continuous* because they consist of a separate of media *quanta* (such as audio samples or video frames), which convey meaning only when presented in time. Several multimedia types, in particular video, require high bandwidths and large storage space. For example, a one half hours of video based on the HDTV (High Definition Television) quality images has approximately 36 Gbits data and requires approximately a 100 Mbps bandwidth, while a current typical magnetic disk drive has only 80 Gbits capacity and serves approximately 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, how to support a continuous retrieval of multimedia data at its required bandwidth and how to store the multimedia data are challenging tasks [2, 4, 6, 12, 18].

Previous approaches to support 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 [5, 14, 15, 16, 19, 20], 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, 8, 11]. 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 constrains a group of consecutive data to be stored consecutively in each cylinder on the disk. While a group of data can be randomly stored on any cylinder [10]. To support *interactive browsing* such as *fast forward* and *fast backward*, some strategies used the *scalable compression algorithms* to generate the multiresolution data [9], and some strategies supported browsing at any desired display speed by a predetermined *sampling* procedure [3].

In all of the previous proposed storage placement strategies for continuous retrieval [1, 5, 8, 11, 14, 15, 16, 19, 20], the storage ordering is assumed to be the same as the retrieval ordering; therefore, there is a requirement of a large amount of file copying or merging operations when users insert/delete some data into/from the media data. The observation is also important for video applications, since features, like video editing, require the support of random access on individual video frames. Although in [10], Liu et al. have proposed a

strategy based on a single disk to support random access, they only consider the case when the the required bandwidth is lower than the bandwidth of a disk drive.

In this paper, we propose an efficient *conflict-resolution* approach to the insertion/deletion operations on continuous media. Since future demands for even high bandwidth are expected, we design the new strategy based on a multi-disk architecture which can increase the data transfer rate and the storage capacity in proportion to the number of disks. Basically, in our proposed strategy, the continuous media are split up into blocks and placed in various locations on the disks (called *simple striping* [1]). Under this situation, the striped subobjects are stored among the multi-disk drive in a predetermined sequence and have to be read in this predetermined sequence to guarantee the continuous retrieval. An insertion or a deletion of subobjects may disturb the property of continuous retrieval because there may be a pair of two consecutive subobjects that are stored in the same disk and have to be retrieved simultaneously. Therefore, an efficient *conflict-resolution* approach is proposed for the insertion/deletion operations on continuous media that are striped into a multi-disk drive without reorganization of the whole data. In the proposed approach, when a new subobject is inserted after subobject i, it will be assigned with an identification number $(i + 1)$ and be inserted to a disk that has minimum number of used blocks (for load balancing concerns) and in which the retrieval of the new subobject does not conflict with the retrieval of any other subobject, where a *conflict* means a pair of two consecutive subobjects that are stored in the same disk have to be retrieved simultaneously. However, a new conflict on the same disk may occur since all the identification numbers of subobjects after subobject i are increased by one. Only when such a new conflict occurs, one movement operation is required, so does the case of a deletion operation. From our performance analysis and simulation results, we show that the average number of additional movements for any subobject insertion is no more than 5, when there is an object that is divided into 120 subobjects and is striped on a multi-disk drive with 12 disks.

Moreover, to reduce those additional movement cost, a *deferring* approach is proposed at the cost of an additional buffer. This approach defers every n insertion operations and puts them in a buffer, where $n \geq 1$. Next, the system finds an optimal insertion sequence that needs minimum additional movement cost, and inserts those data according to such an insertion sequence. Based on this approach, two strategies are proposed: the *conflict-resolved-first-deferring* strategy (the $CRFD$ strategy) and the *conflict-resolved-last-deferring* strategy (the $CRLD$ strategy). In the the $CRFD$ strategy, those n new subobjects are inserted one after one according to the optimal insertion sequence and a conflict is resolved immediately if it occurs. In the $CRLD$ strategy, it resolves all the

conflicts after those n new subobjects are inserted. From the simulation study, we show that the $CRLD$ strategy has a lower insertion cost than the $CRFD$ strategy. Moreover, from our performance analysis, we will prove that the optimal insertion sequence in the $CRLD$ strategy will always be the one with an ascending order of identification numbers. That is, the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers is an optimal strategy based on the proposed *deferring* approach.

The rest of the paper is organized as follows. Section 2 surveys the related works on the data retrieval of digital continuous media. Section 3 briefly describes the *simple striping* strategy that is applied in our approach. Section 4 presents the algorithms for insertion and deletion operations on digital continuous media. Section 5 presents the performance analysis and simulation results of the algorithms. Section 6 presents the *deferring* approach to reduce the additional movement cost. Section 7 contains a conclusion.

2 Related Works

In this section, we will survey several research directions in a video server, which are classified into three areas, including *continuous retrieval*, *random access* and *interactive browsing*.

2.1 Continuous Retrieval

Since magnetic disks have large seek and rotational latencies that cause the display of a continuous media to starve, how to arrange and place the media data on disks in order to support real-time retrieval and flexible placement is an important task. Gemmell and Christodoulakis [5] were the first to develop a general theoretical framework for studying the performance of any display system with a dedicated storage device and discuss several strategies for placing audio data on a storage device in order to display multiple-channel audio synchronously. Yu et al. [20] proposed a storage placement strategy by using *gaps* between consecutive blocks to provide the necessary delays to match the consumption rate without requiring excessive buffering. Moreover, since blocks of a new data may be stored in the gaps of already existing data on the disk, there were several merging algorithms [14, 15, 16, 19, 20] proposed to utilize the disk space efficiently. However, these approaches are effective only when the maximum bandwidth of disk is higher than the required bandwidth.

Moreover, current studies in a multi-disk drive were proposed to increase the data transfer rate and the storage capacity in proportion to the number of disks. *Declustering*

and *striping* are two popular techniques employed by both general purpose multi-disks I/O subsystems (e.g., disk arrays) and parallel database management systems (e.g., share-nothing architecture). In [8], they extended the concept of *declustering* to a parallel multimedia system based on the the shared-nothing architecture and proposed two alternative approaches to enable the system to support simultaneous display of multiple multimedia data. The first approach, termed *disk multitasking*, configures the system to support a fixed number of users (U_{max}) or simultaneous display. The major disadvantage of *multitasking* is the limit on the number of requests and the waste of bandwidth when the total number of requests is less than U_{max}. The second approach replicates the fragments of a media data across several processors so that the system can use a fragment of a media data that resides on an idle processor to service a request. However, the replications have a high overhead of copying. In [1, 11], they used the concept of *striping* to support continuous retrieval of digital continuous media. Moreover, Berson et al. [1] further generalized the simple *striping* (called *staggered striping*) to support a database that consists of a mix of multimedia objects, each with a different bandwidth requirement.

2.2 Random Access

In previous described storage placement strategies, they require a large amount of file copying and merging operations in the editing on the continuous media. The observation is important for video applications since features, like video editing, require the support of random access on individual video frames. Liu et al. [10] were the first to investigate a solution based on a video-frame level, which offers more flexible storage placement and efficient buffering schemes. They proposed two buffering schemes which are the *two-buffer scheme* and the *k-buffer compensation scheme*. The *two-buffer scheme* requires only a small group of sequential video frames stored consecutively (i.e., clustered) in each cylinder on the disk placement. The *k-buffer compensation scheme* uses more than two buffers and requires some blocks to be placed randomly in the same cylinder on disk placement. The idea of *compensation* is motivated by the fact that the disk data transfer rate varies; it is faster when video frames are placed close together and slower when more seek and latency time are required.

2.3 Interactive Browsing

One of most important challenges in a Video-On-Demand system is to support *interactive browsing* functions such as *fast forward with scan* and *fast backward with scan*. There

are several possible approaches to implementing these functions: (1) The data is retrieved and transmitted at the desired rate which times the normal display rate. Obviously, this solution requires additional resources at the storage system, the memory buffers, and the network. (2) The storage system retrieves and transmits every desired number of block to the end station. This solution also requires significant additional system resources since the multimedia file must be indexed to retrieve individual blocks. (3) The system switches over to a separately coded "scan forward" data to scan. This solution eliminates any additional read bandwidth or network bandwidth. However, it is extremely expensive in terms of storage space and inflexible in which it supports a fixed scan rate.

Chen et al. [3] proposed two new schemes for supporting interactive browsing for video display. The first methods, called *segment-sampling* scheme, supports browsing at any desired speed while balancing the load on the disk array as well as minimizing the variation on the number of video segments skipped between samplings. Since *segment-sampling* scheme makes the segments retrieved not perfectly uniformly distributed, the second scheme, called *segment-placement* scheme, uniformly retrieves the segments in a predetermined fast-forward speed.

3 Simple Striping

In our approach, we apply the *simple striping* strategy to arrange the continuous media on a multi-disk drive. Suppose the bandwidth of both the network and the network device driver exceeds the bandwidth requirement of an object, where an object means an object of digital continuous media. 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 at least requires the aggregate bandwidth of $M = \lceil \frac{C}{d} \rceil$ disk drives to support the continuous display 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.) 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 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 ($= \lfloor \frac{N}{M} \rfloor$) disk clusters, where each cluster is assigned to an object for the retrieval of the M consecutive subobjects to guarantee the real-time transfer. It is known

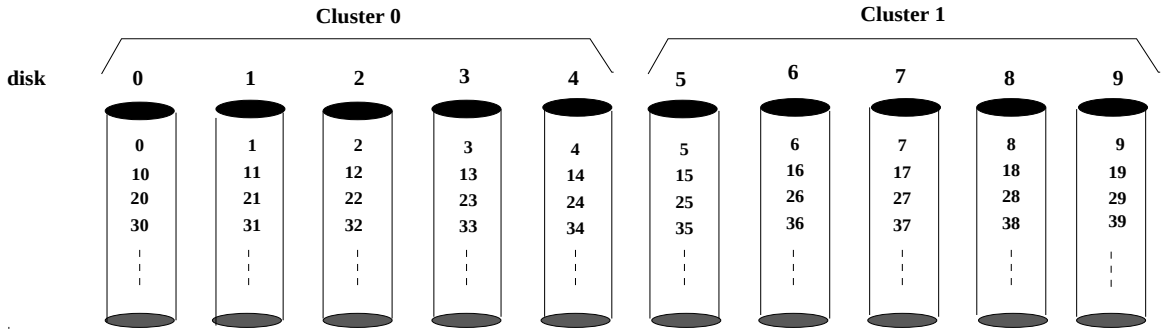

Fig. 1. An example of object X striped on a multi-disk drive.

that concurrent pipelining of retrieval and displaying of an object requires prefetching and at least two buffers [14]. One buffer is for retrieval of the next M consecutive subobjects while the other one is to store the previous retrieved M consecutive subobjects which is displayed. The real-time retrieval can be achieved by satisfying following equations:

$$M \geq \lceil \frac{C}{d} \rceil,$$
$$M \leq N,$$
$$R = \lfloor \frac{N}{M} \rfloor,$$
$$WS + WL + \frac{s}{d} \leq \frac{s}{\frac{C}{M}}, \text{ for each disk drive,}$$

where s denotes the block size.

Hence, the 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 those M disks in a liner order. Figure 1 shows an example of simple striping for an object X with bandwidth requirement 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 $M = 5$ ($\geq \frac{80}{20}$), then s ($= \frac{(WS+WL)\times d\times \frac{C_i}{M}}{d-\frac{C_i}{M}}$) $= 3.2$ Mbits (Mbits $= 10^6$ bits) and $R = 2$. Note that the 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. Second, 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 the two buffers and are then displayed.

4 The Conflict-Resolution Approach

Editing operations such as insertion, deletion and modification are necessary for a general purpose file storage system. If an object is read only, then the object striped on a multi-disk drive is satisfied for the continuous display as described in Section 3. However, an insertion or a deletion of subobjects may disturb the property of continuous retrieval because there may be a pair of two consecutive subobjects that are stored in the same disk and have to be retrieved simultaneously. Therefore, in the following subsections, we will present a *conflict-resolution* approach to support insertion and deletion operations to guarantee continuous display, respectively. (Note that a modification operation can be replaced with a deletion operation followed by an insertion operation.) In this approach, when a conflict is detected after an insertion or a deletion operation, a conflict-resolution operation is applied by moving one of the conflicting subobjects to some other disks such that the conflict can be resolved. Since each subobject is organized as a block on the disks in terms of a sequence of continuously digital media, which sequence is immutable, we regards blocks as immutable objects. Therefore, we assume that a block (i.e., a subobject) is regarded as a unit for all editing operations.

4.1 Insertion

Assume that a new subobject is inserted into an object striped on a multi-disk drive. A straightforward approach to support an insertion operation will require substantial movements of subobjects after the inserted subobject to their corresponding disks depending on the simple striping placement strategy. Obviously, these movements will consume significant amount of time and space. Therefore, we propose an algorithm based on the conflict-resolution approach as shown in Figure 2 for inserting a new subobject x after the subobject with identification number $= i$ (i.e., subobject i) without reorganizing the whole data, which usually performs an insertion of subobject without requiring any additional movement cost of other subobjects.

First, when a new subobject is inserted after subobject i, all the identification numbers of subobjects after subobject i are increased by one. A *conflict* occurs when a pair of two consecutive subobjects that are stored in the same disk have to read into the buffer simultaneously to guarantee the continuous display. Therefore, the new subobject with identification number $= (i + 1)$ should be inserted into a disk j in which the retrieval of the new subobject does not conflict with the retrieval of any other subobject. Moreover, for load balancing concerns, disk j should be the one that has minimum number of used

blocks. However, since all the identification numbers of subobjects after subobject i are increased by one, this may cause a new conflict to occur between a pair of subobjects with identification numbers = k and v in the same disk, where $v > k > (i + 1)$. Therefore, one more movement operation to resolve such a conflict is needed if such a new conflict occurs. (Note that since an aggregation bandwidth of M disks is achieved by retrieving M subobjects from the M disks into a buffer at each round, simultaneously, a conflict can be detected by comparing the value of $\lfloor \frac{k}{M} \rfloor$ with the one of $\lfloor \frac{v}{M} \rfloor$. When these values are the same, a conflict occurs because that subobject k and subobject v will be retrieved from the same disk at the same time.) For load balancing concerns about the disks, this proposed algorithm requires the following two arrays: 1) an array of *counters* to record the number of used blocks for every disk; 2) an array of *flags* to record whether a conflict occurs when a subobject is moved into a disk.

For example, Figure 3-(a) shows the state of the multi-disk drive after the insertion of a new subobject X_{0a} after subobject X_0 of object X with bandwidth requirement = 80 Mbps, where $N = 5$, $d = 20$ Mbps, $WS = 30$ ms, $WL = 10$ ms, $M = 5$, $R = 1$, $s = 3.2$ Mbits and the number in "()" denotes the original identification number before any insertion operation occurs. First, all the identification numbers of subobjects after subobject 0 are increased by one. Then, the identification number of the new inserted object is assigned to 1. Figure 3-(b) shows the state of the multi-disk drive after X_{0b}, X_{0c} and X_{0d} are inserted. Moreover, Figure 3-(c) shows the state of the multi-disk drive after X_{0e} is inserted. (Note that at this moment, five subobjects are inserted in sequence after subobject 0; therefore, all the identification numbers after subobject 0 have been increased by 5.) At this moment, a new conflict occurs between subobject 5 (i.e., subobject 0a) and subobject 9. Therefore, subobject 5 is moved to disk 0 in which there is no subobject with an identification number = g such that $\lfloor \frac{g}{M} \rfloor = \lfloor \frac{5}{M} \rfloor$.

Consequently, whenever a conflict between two subobjects k and v occurs on disk j, one of these two conflicting subobjects, say subobject k, has to move to another disk f in which there will be no conflict after subobject k is moved. To prove that such a disk f exists at any time, that is, the repeat-until loop in algorithm *insertion* will always stop, the following theorem is used.

Theorem 1 *Given an object X striped on a multi-disk drive with N disks and a conflict between two subobjects with identification numbers k and v on disk l ($0 \leq l \leq N$ - 1), where $\lfloor \frac{k}{M} \rfloor = \lfloor \frac{v}{M} \rfloor$, there is a disk f ($0 \leq f \leq N$ - 1) in which there is no subobject with an identification number = g such that $\lfloor \frac{k}{M} \rfloor = \lfloor \frac{g}{M} \rfloor$.*

```
procedure insertion(i, x);
var
      Q[N] : an array of counters;
      CF[N] : an array of flags;
      M : integer; /* the number of disks in a cluster */
      c, i, j, k, f, v, g, h : integer;
      x : BLOB; /* Binary Large OBject */
begin
      increase all the identification numbers of subobjects after subobject i by one;
      CF[*] = 0;                  /* set all the values of CF to 0 */
      c = 1;
      repeat
            j = cmin(Q, c);
                  /* function cmin returns a disk identification number j that has the cth
                     minimum used space (as recored in Q) among all the values in Q */
            for each subobject with an identification number g on disk j do
                  if ⌊ (i+1)/M ⌋ = ⌊ g/M ⌋ then CF[j] = 1;
            end for;
            c = c +1;
      until CF[j] = 0;
      insert the new subobject x into disk j with an identification number = (i + 1);
      Q[j] = Q[j] + size of x;
                        /* detect all the conflicts and resolve them */
      for (j=0;j<N;j++) do
         while (there is a pair of subobjects in disk j with identification numbers k and v,
               where ⌊ k/M ⌋ = ⌊ v/M ⌋, and (i + 1) < k < v) do
                        /* a conflict is detected */
            CF[*] = 0;
            c = 1;
            repeat
                  f = cmin(Q, c);
                  for each subobject with an identification number g on disk f do
                        if ⌊ k/M ⌋ = ⌊ g/M ⌋ then CF[f] = 1;
                  end for;
                  c = c +1;
            until CF[f] = 0;
            move the subobject with an identification number k to disk f;
            Q[j] = Q[j] - size of subobject k;
            Q[f] = Q[f] + size of subobject k;
         end while;
      end for;
end;
```

Fig. 2. Algorithm *insertion*.

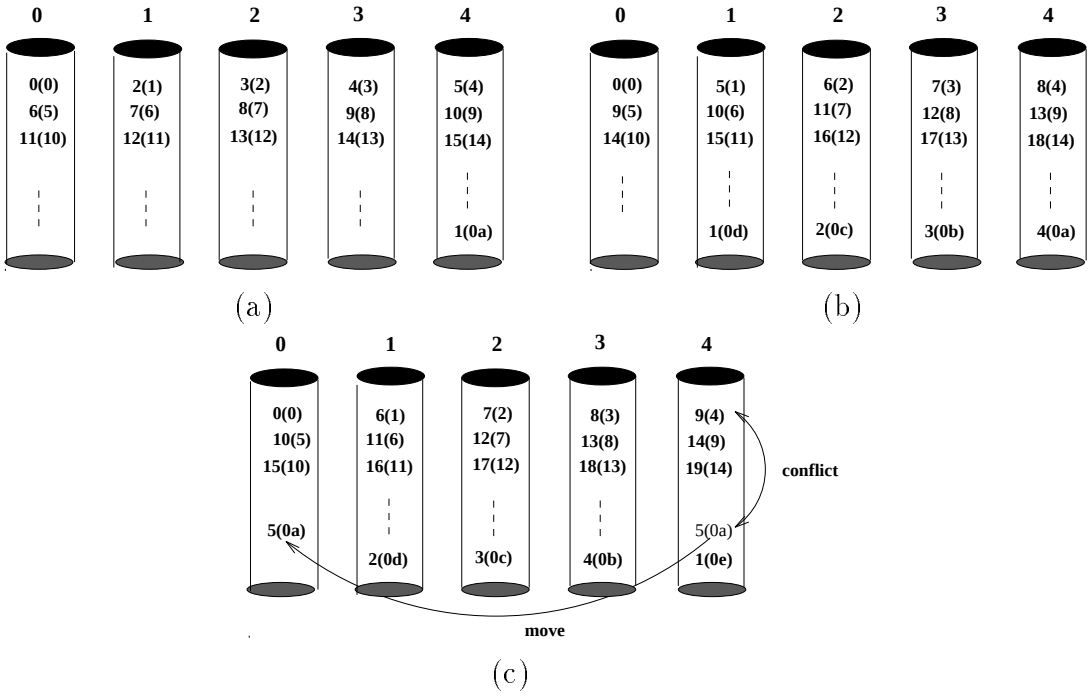

Fig. 3. An example of a series of insertion operations: (a) after X_{0a} is inserted; (b) after X_{0b}, X_{0c} and X_{0d} are inserted; (c) after X_{0e} is inserted.

Proof. Assume that no such a disk f exists; that is, for every disk, there is a subobject with an identification number = z such that $\lfloor \frac{k}{M} \rfloor = \lfloor \frac{z}{M} \rfloor = w$. Note that there are exact (M - 1) different values of z (excluding k) which satisfy the equation. By using the *pigeonhole principle*, there is at least one empty pigeonhole if there are (M - 1) pigeons that are put into N ($\geq M$) pigeonholes. That is, there is at least one disk, say disk f, where there is no subobject with an identification number = g on disk f such that $\lfloor \frac{k}{M} \rfloor = \lfloor \frac{g}{M} \rfloor = w$. Therefore, the previous assumption is wrong. Consequently, such a disk f exists at any time. □

4.2 Deletion

Similar to the case of an insertion, a deletion of subobject may disturb the property of continuous retrieval. The proposed algorithm for deleting a subobject with an identification number i is similar to the *insertion* algorithm with the potential requirement of an additional movement cost, but all the identification numbers of subobjects after subobject i are decreased by one. Since the deletion of a subobject may result in a new conflict between two subobjects stored in the same disk, we have to check the possible conflicts

which may occur between the subobjects with the identification numbers k ($k \geq i$) in each disk.

5 Performance Analysis and Simulation Results

In this section, we present the performance analysis and the simulation results of insertion/deletion operations for a striped object on a multi-disk drive by applying our proposed insertion/deletion algorithms. Since a deletion operation may cause the movement of the other subobjects as an insertion operation does, the performance analysis and the simulation results for a deletion operation is similar to those for an insertion operation. Therefore, we only discuss the performance of the insertion operations. In this analysis 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 striped on the multi-disk drive by applying the *simple striping* strategy and all the subobjects of an object have the same size. Each disk has approximately the same number of subobjects, that is, load balancing. The performance measure is the number of read/write operations (including the write operation for the data insertion) in the multi-disk drive to guarantee the real-time retrieval after an insertion operation.

Suppose N is the number of disks in a multi-disk drive, in which each disk has a fixed bandwidth d, a worst seek time WS, and a worst latency time WL. An object X with a required bandwidth C is striped on the multi-disk drive. The values of R, M and s can be obtained by applying the equations as described in Section 3, where R denotes the number of clusters, M denotes the number of subobjects in a cluster, and s denotes the subobject size. Assume that the the size of object X is V_X and there are Num subobjects of object X with $Num = \lceil \frac{V_X}{s} \rceil$. Logically, we can view the current state of the object striped on the multi-disk drive as L ($= \lceil \frac{Num}{M} \rceil$) groups numbered from 0 to (L - 1), where the subobjects in each group have to be retrieved, simultaneously. Each group has M subobjects except group (L - 1). There are FS ($= Num$ - $M \times (L$ - 1)) subobjects in group (L - 1). Figure 4-(a) shows a simple example of a logical view of an object X striped on a multi-disk drive with $N = 6$, where $Num = 10$, $M = 3$, $L = 4$ and $FS = 1$. (Note that, for those subobjects in the same logical group, we store them in different physical disks to guarantee the real-time retrieval.) In other words, the values of $\lfloor \frac{id}{M} \rfloor$ for these M consecutive subobjects are the same; i.e, they conflict with each other, where id denotes the identification number. In the above example, subobjects 0, 1 and 2 have the same value $\lfloor \frac{0}{M} \rfloor = \lfloor \frac{1}{M} \rfloor = \lfloor \frac{2}{M} \rfloor$

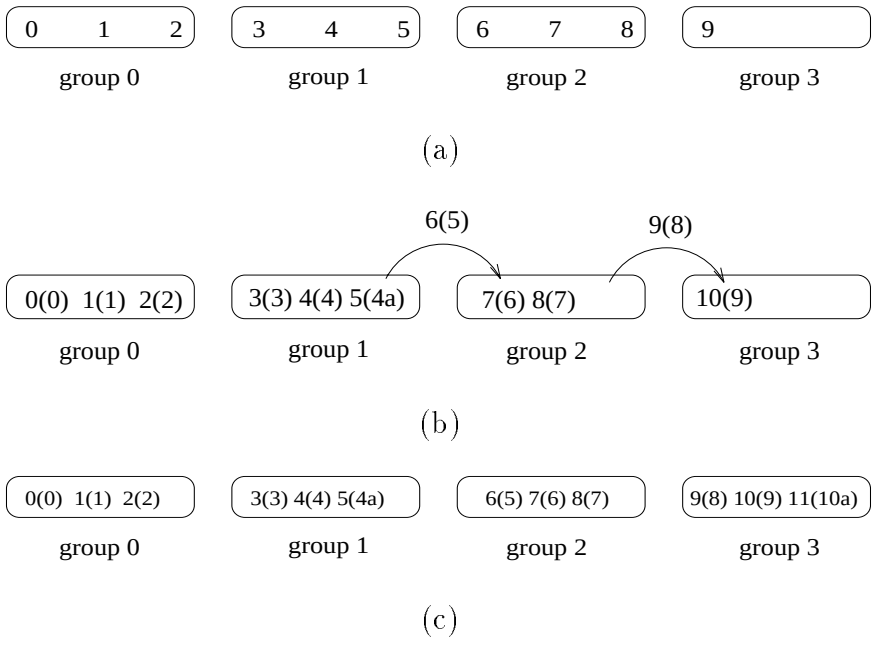

Fig. 4. An example of a logical view of an object X striped on a multi-disk drive: (a) before any insertion occurs; (b) after a new subobject is inserted after subobject 4; (c) after a new subobject is inserted after subobject 10.

= 0; therefore, they have to be stored in different disks to avoid conflict, so do the other subobjects with the same value of $\lfloor \frac{id}{M} \rfloor$.

Whenever a new subobject is inserted after subobject i, this new subobject with identification number $(i + 1)$ is inserted into disk f, where no conflict occurs as described before. While the insertion may cause new conflicts between other subobjects behind the new inserted subobject to occur. The reason is that all the identification numbers greater than i have to be increased by one. Logically, all the subobjects in those groups which are behind group t $(= \lfloor \frac{i}{M} \rfloor)$ that subobject i belongs to, have to be re-clustered according to the order for displaying. Logically, the moved-out subobject in a group t will be the moved-in subobject in group $(t + 1)$. There are $(M$ - 1) subobjects that have not to be moved in each group. (Note that there is no conflict between any two subobjects of those $(M$ - 1) subobjects because, in the physical storage, these $(M$ - 1) subobjects are stored in different disks.) Therefore, a conflict can occur only between the moved-in subobject and one of those $(M$ - 1) subobjects in the same disk. Consequently, after an insertion operation occurs, the probability of a conflict that occurs between a moved-in subobject and one of those $(M$ - 1) subobjects in the same disk is $\frac{M-1}{N}$.

For example, Figure 4-(b) shows the state resulted from Figure 4-(a) after a new

subobject 4a is inserted after subobject 4, where the number in "()" denotes the original identification number before any insertion operation occurs. In this example, all the identification numbers of subobjects after subobject 4 have to be increased by one and the new subobject 4a is inserted into group 1 with an identification number 5. Since subobject 6(5) belongs to group 2 (= $\lfloor \frac{6}{M} \rfloor$), instead of group 1, subobject 6(5) has to be moved from group 1 to group 2. Therefore, a conflict can occur in group 2 only between subobject 6(5) and subobjects 7(6), 8(7) since subobject 9(8) has to be moved from group 2 to group 3 at the same time. Consequently, the probability of such a conflict is $\frac{2}{6}$. For the same reason, the probability of a conflict in group 3 between subobject 9(8) and subobject 10(9) is $\frac{1}{6}$. Moreover, Figure 4-(c) shows the state resulted from Figure 4-(b) after one more new subobject 10a is inserted after subobject 10(9). Since there are only 2 subobjects 10(9) and 11(10a) in group 3 after this new insertion operation is performed and no subobject has to be moved out of group 3, the probability of a conflict is 0.

There are only two cases of inserting a new subobject in the last group (i.e., group (L - 1)). One case is that when the number of subobjects in group (L - 1) $< M$ (i.e., $0 < FS < M$), the total number of subobjects in group (L - 1) is not greater than M after the new subobject is inserted; therefore, no subobject has to be moved out from group (L - 1). In this case, the probability of a conflict of such a new subobject insertion is 0 because no subobject has to be moved out. The other case is that when the number of subobjects in group (L - 1) = M (i.e., $FS = 0$), the total number of subobjects in group (L - 1) is (M + 1) after the new subobject is inserted; therefore, a subobject has to be moved out from group (L - 1) to a new group, say group L. In this case, the probability of a conflict of such a new subobject insertion is 0 because the moved-out subobject from group (L - 1) to group L does not conflict with the others since no subobject is stored in group L before this new subobject insertion. Therefore, the probability of a conflict of inserting a new subobject in the last group is always 0.

Since a conflict of each group occurs independently, the number of conflicts for an insertion after subobject i is obtained as

$$NC(i) = (\lfloor \tfrac{Num}{M} \rfloor - \lfloor \tfrac{i}{M} \rfloor - 1) \times \tfrac{M-1}{N} + \tfrac{FS}{N}, \qquad \text{when } \lfloor \tfrac{Num}{M} \rfloor > \lfloor \tfrac{i}{M} \rfloor \text{ or}$$
$$NC(i) = 0, \qquad \text{when } \lfloor \tfrac{Num}{M} \rfloor = \lfloor \tfrac{i}{M} \rfloor,$$

and the total cost for an insertion after subobject i is given by

$$INS(i) = NC(i) \times 2 + 1.$$

Then, the average cost for an insertion can be obtained as

$$AV\_INS(Num) = \frac{\sum_{i=0}^{Num-1} INS(i)}{Num}.$$

Since $NC(u) = NC(v)$ when $\lfloor \frac{u}{M} \rfloor = \lfloor \frac{v}{M} \rfloor$, the cost for the insertion after subobject u is equal to the cost for the insertion after subobject v, that is, $INS(u) = INS(v)$. In other words, the cost for any insertion in a group has the same insertion cost. Therefore, the average cost for an insertion can also be given by

$$\begin{aligned} AV\_INS(Num) &= \frac{\sum_{i=0}^{\lceil \frac{Num}{M} \rceil - 1} INS(i \times M)}{\lceil \frac{Num}{M} \rceil} \\ &= \frac{((\lfloor \frac{Num}{M} \rfloor - 1) \times \frac{M-1}{N} + \frac{FS}{N}) \times 2 + 1}{\lceil \frac{Num}{M} \rceil} + \frac{((\lfloor \frac{Num}{M} \rfloor - 2) \times \frac{M-1}{N} + \frac{FS}{N}) \times 2 + 1}{\lceil \frac{Num}{M} \rceil} + \ldots \\ &\quad + \frac{(0 \times \frac{M-1}{N} + \frac{FS}{N}) \times 2 + 1}{\lceil \frac{Num}{M} \rceil} + \frac{1}{\lceil \frac{Num}{M} \rceil} \\ &= \frac{(\lfloor \frac{Num}{M} \rfloor - 1) \times \lfloor \frac{Num}{M} \rfloor \times \frac{M-1}{N} + \frac{FS}{N} \times \lfloor \frac{Num}{M} \rfloor \times 2}{\lceil \frac{Num}{M} \rceil} + 1, \\ &\qquad\qquad \text{when } FS > 0, \text{ i.e., } \lfloor \frac{Num}{M} \rfloor < \lceil \frac{Num}{M} \rceil, \text{ or} \end{aligned}$$

$$AV\_INS(Num) = (\lfloor \frac{Num}{M} \rfloor - 1) \times \frac{M-1}{N} + 1, \qquad \text{when } FS = 0.$$

For a series of k insertion operations after subobjects i_1, i_2, ..., i_k, respectively, the total cost of these k insertion operations is obtained as the sum of the cost of an insertion after subobject i_j, $1 \leq j \leq k$. That is,

$$TOTINS(k) = \sum_{j=1}^{k} INS(i_j).$$

Note that the number Num of subobjects is increased by one after an insertion has been performed. The average cost for any series of k ($k > 0$) insertion operations can be obtained as

$$AV\_TOTINS(k) = \frac{\sum_{j=Num}^{Num+k-1} AV\_INS(j)}{k}.$$

Table 1 shows the results in terms of the number of read/write operations derived from the above formulas, where $N = 12$, $Num = 120$, $s = 1$ Mbits, $M = 2$, 3, 4, 6 and 12, respectively, and the number in "()" denotes the number of the additional movement operations. From this table, we observe that the additional movement cost is much smaller than the cost for reorganizing the whole object. (Note that the cost for the reorganization is $2 \times (Num - i - 1)$, when a new subobject is inserted after subobject i.) Moreover, the

Table 1. Analysis results.

N	M	Num	INS(1)	INS(60)	INS(118)	AV_INS(Num)	AV_TOTINS(120)/120
12	2	120	10.8(4.9)	5.8(2.4)	1(0)	5.9(2.5)	8.3(3.7)
12	3	120	14.0(6.5)	7.3(3.2)	1(0)	7.5(3.3)	10.7(4.9)
12	4	120	15.5(7.3)	8.0(3.5)	1(0)	8.2(3.6)	11.8(5.4)
12	6	120	16.8(7.9)	8.5(3.8)	1(0)	8.9(4.0)	12.8(5.9)
12	12	120	17.5(8.3)	8.3(3.7)	1(0)	9.2(4.1)	13.3(6.2)

Table 2. Performance: (a) simulation results; (b) analysis results.

N	M	Num	INS(1)	INS(7)	INS(10)	AV_INS(Num)	AV_TOTINS(12)/12
4	2	12	2.0(0.5)	1.46(0.23)	1(0)	1.5(0.25)	1.76(0.38)
4	4	12	3.96(1.48)	2.52(0.76)	1(0)	2.49(0.745)	3.2(1.1)

(a)

N	M	Num	INS(1)	INS(7)	INS(10)	AV_INS(Num)	AV_TOTINS(12)/12
4	2	12	2.0(0.5)	1.5(0.25)	1(0)	1.5(0.25)	1.75(0.375)
4	4	12	4.0(1.5)	2.5(0.75)	1(0)	2.5(0.75)	3.25(1.125)

(b)

cost for inserting a new subobject after subobject i is increased as i is decreased, because the probability of a conflict is increased. Since there are $(\mathrm{N!})^{\lceil \frac{Num}{N} \rceil}$ kinds of initial states of the multi-disk drive, it requires a lot of time to simulate all the cases if N or Num are too large. Therefore, in the simulation study, we consider a case with $N = 4$, $Num = 12$, $s = 1$ Mbits, $M = 2$ and 4, respectively. Table 2 shows the simulation results and analysis results under the same condition, where the number in "()" denotes the number of the additional movement operations. (Note that in this case, we have to simulate all of the $(4!)^3$ $(= 13824)$ initial states of the multi-disk drive and calculate the average results.) Compared with the analysis results shown in Table 2-(b), the simulation results shown in Table 2-(a) are very close to those shown in Table 2-(b). Note that the 12 new added subobjects are generated randomly.

6 The Deferring Approach

In this section, we extend the *conflict-resolution* approach with a *deferring* approach to a sequence of subobject insertions to reduce the additional movement cost. Based on this approach, two strategies are proposed: the *conflict-resolved-first-deferring* strategy (the $CRFD$ strategy) and the *conflict-resolved-last-deferring* strategy (the $CRLD$ strategy). Moreover, from the study of performance analysis, we will show an optimal strategy based on the *deferring* approach.

6.1 An Interesting Observation

Consider an example for inserting two new subobjects after subobject 1 and subobject 8, respectively, as shown in Figures 5, where object X with bandwidth requirement = 80 Mbps, $N = 5$, $d = 20$ Mbps, $WS = 30$ ms, $WL = 10$ ms, $M = 5$, $R = 1$, $s = 3.2$ Mbits, $Num = 15$ and the number in "()" denotes the original identification number before any insertion operation. Figure 5-(a) shows the state of the multi-disk drive before any insertion occurs. Figure 5-(b) shows the case of the multi-disk drive after the insertion of the new subobject after subobject 1 and then the insertion of the other new subobject after subobject 9(8) are executed. The total insertion cost for these two new added subobjects is 6. On the other hand, Figure 5-(c) shows the case of the multi-disk drive after the insertion of the new subobject after subobject 8 and then the insertion of the other new subobject after subobject 1(1) are executed. The total insertion cost for these two new added subobjects is 8. Therefore, we observe an interesting result: an insertion sequence will affect the total insertion cost when a series of new subobjects are inserted one after one based on our proposed *insertion* algorithm. For the example shown in Figure 5, a minimum insertion cost is obtained if those two new subobjects are inserted one after one according to the ascending order of logical positions.

Consider another example for inserting two new subobjects after subobject 1 and subobject 4, respectively, as shown in Figures 6, where object X with bandwidth requirement = 80 Mbps, $N = 5$, $d = 20$ Mbps, $WS = 30$ ms, $WL = 10$ ms, $M = 5$, $R = 1$, $s = 3.2$ Mbits, $Num = 15$ and the number in "()" denotes the original identification number before any insertion operation occurs. Figure 6-(a) shows the state of the multi-disk drive before any insertion occurs. The total insertion cost for these two new added subobjects according to an insertion sequence = (1, 5(4)) is 8 as shown in Figure 6-(b). On the other hand, the total insertion cost for these two new added subobjects according to the other insertion sequence = (4, 1(1)) is 6 as shown in Figure 6-(c). For the example shown in

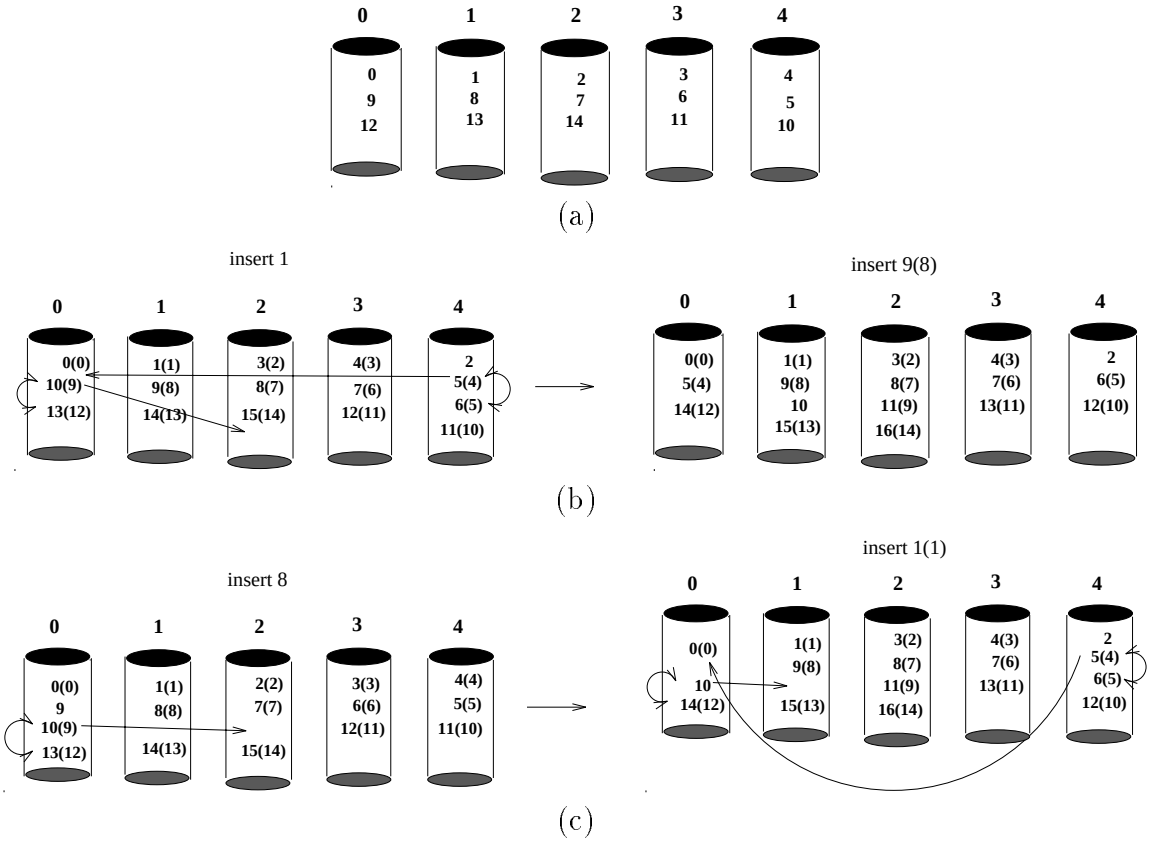

Fig. 5. An example of inserting two new subobjects: (a) initial state of object X; (b) the state after following the insertion sequence: 1, 9(8); (c) the state after following the insertion sequence: 8, 1(1).

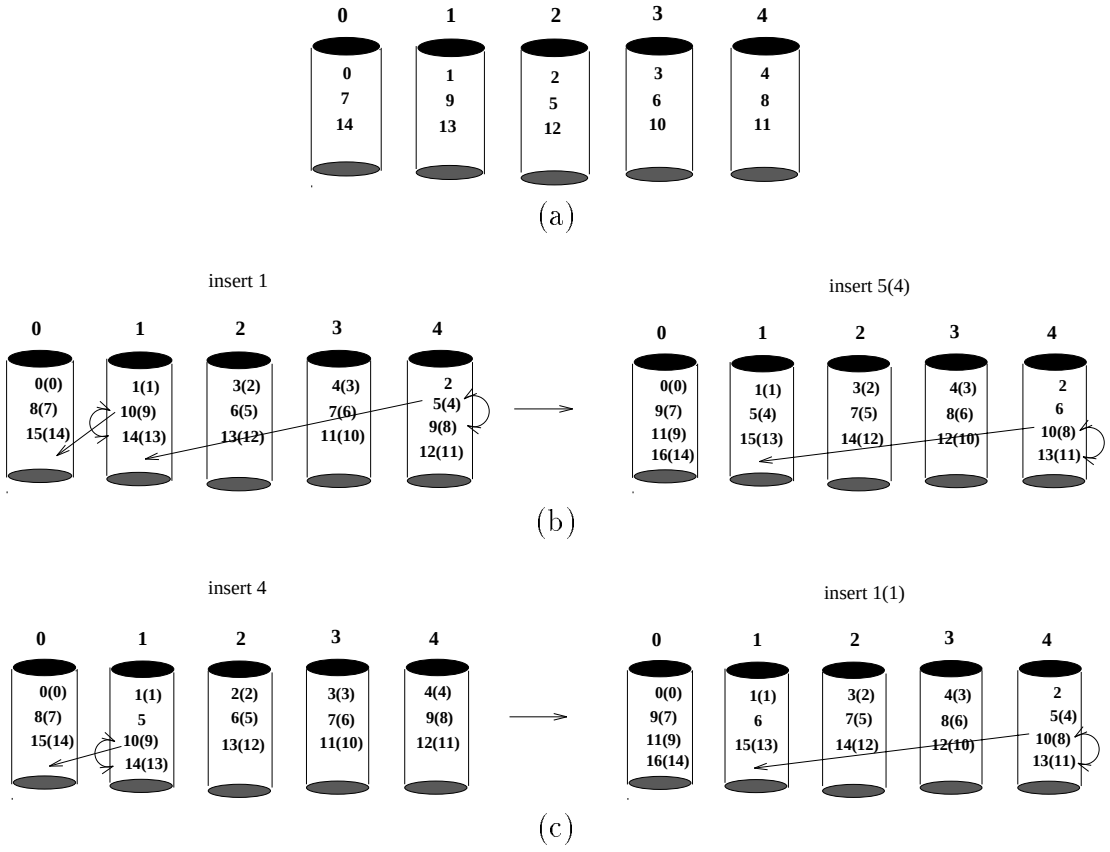

Fig. 6. An example of inserting two new subobjects: (a) initial state of object X; (b) the state after following the insertion sequence: 1, 4(5); (c) the state after following the insertion sequence: 4, 1(1).

Figure 6, a minimum insertion cost is obtained if those two new subobjects are inserted one after one according to the descending order of logical positions. From the above two examples, we observe that an insertion sequence can affect the total cost and a series of insertion operations according to a certain insertion sequence will require the minimum cost. To find an optimal insertion sequence for a series of k new subobjects, we have to test all the possible insertion sequences. The computation complexity is $O(k!)$. However, if k is not too large and a buffer is used, we can make use of the idea of the optimal insertion sequence to reduce the movement overhead of the *insertion* algorithm described in the previous section. This is the motivation of the *deferring* approach, in which we defer k incoming insertion operations and store those k new subobjects in a buffer, and then insert those k subobjects by following an optimal insertion sequence.

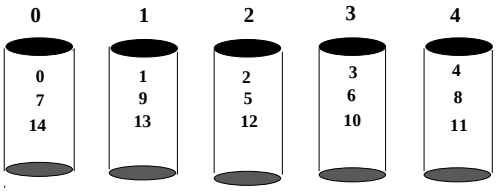

Fig. 7. The initial state.

Table 3. The insertion costs of 24 different insertion sequences.

insertion sequnce	insertion cost	insertion sequnce	insertion cost	insertion sequnce	insertion cost	insertion sequnce	insertion cost
1 4 24 37	14	4 1 24 37	12	24 1 4 37	18	37 1 4 24	22
1 4 37 24	16	4 1 37 24	14	24 1 37 4	20	37 1 24 4	24
1 24 4 37	16	4 24 1 37	14	24 4 1 37	16	37 4 1 24	20
1 24 37 4	18	4 24 37 1	16	24 4 37 1	18	37 4 24 1	22
1 37 4 24	18	4 37 1 24	16	24 37 1 4	22	37 24 1 4	26
1 37 24 4	20	4 37 24 1	18	24 37 4 1	20	37 24 4 1	24

For example, given an initial state of the multi-disk drive shown in Figure 7 and four new subobjects which will be inserted after subobjects 1, 4, 24 and 37, respectively, Table 3 shows the insertion costs of 24 different insertion sequences, where $N = 5$, and an object X with $M = 5$, $R = 1$, $Num = 15$ and $s = 1$ Mbits. Among these 4! (=24) insertion sequences, we observe that the series of insertion operations according to the insertion sequence (4(4), 1(1), 26(24), 40(37)) will require the minimum cost 12. To find such an insertion sequence which requires the minimum insertion cost, we only have to prepare a buffer with a size of 4 subobjects in the *deferring* approach.

6.2 The CRFD Strategy

The first strategy to implement the *deferring* approach is called the *conflict-resolved-first-deferring* strategy (or the *CRFD* strategy), in which it chooses an optimal insertion sequence for every n new incoming subobjects with a computation complexity $O(n!)$ and then, inserts these new subobjects one after one according to the optimal sequence for those n new subobjects by applying the *insertion* algorithm described before, respectively. Therefore, given a series of k data insertion operations, the total time complexity to choose all of the optimal insertion sequences for every n incoming insertion operations is ($\lceil \frac{k}{n} \rceil \times$ O($n!$)), which is much smaller than O($k!$). Figures 6 is such an example. Note that in this

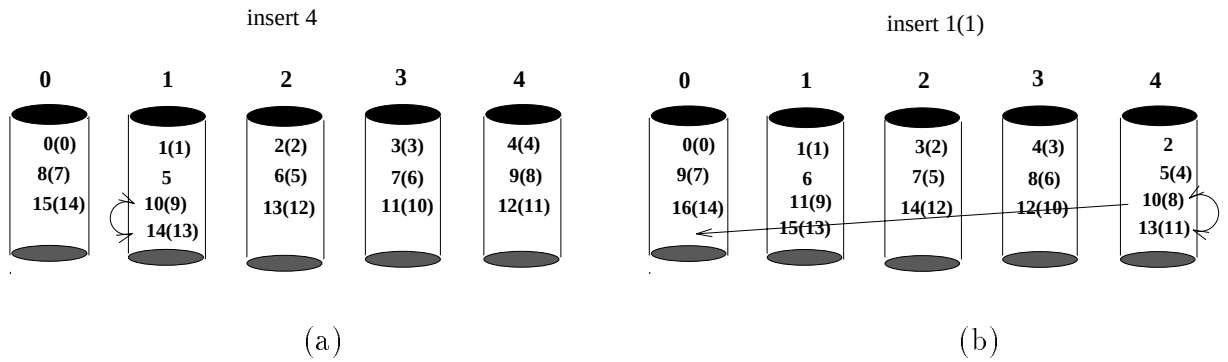

Fig. 8. An example of the insertion of the two new subobjects: (a) a conflict between subobjects 10(9) and 14(13); (b) the conflict disappears.

strategy, a conflict is resolved immediately after a new subobject is inserted if it occurs.

6.3 The CRLD Strategy

In the $CRFD$ strategy, a moved-out subobject (due to a conflict) during the execution of the *insertion* algorithm may not have to be moved out after some other subobject is inserted since the insertion of this new inserted subobject changes some of the identification numbers again, which may make the previous conflict disappear. In other words, we may save a movement operation if we do not resolve a conflict immediately. Such an example is shown in Figure 8, which has the same initial state as in Figure 6-(a). We do not resolve a conflict between subobjects 10(9) and 14(13), immediately, after subobject 5 is inserted. After one more new subobject (i.e., subobject 2) is inserted after subobject 1(1), the previous conflict between subobjects 11(9) and 15(13) disappears. (Note that the identification numbers of subobject 10(9) and 14(13) are increased by one after the new subobject (i.e., subobject 2) is inserted after subobject 1(1).) Compared with the total cost (= 6) shown in Figure 6-(c) in which it shows the results of the same data insertions according to the $CRFD$ strategy, the total cost of the insertion for these two subobjects shown in Figure 8 according to the second strategy is reduced to 4. Therefore, the second strategy to implement the *deferring* approach is proposed, which is called the *conflict-resolved-last-deferring* strategy (the $CRLD$ strategy). The $CRLD$ strategy inserts all these new subobjects stored in the buffer and increases the identification numbers of related subobjects, but does not resolve the conflicts until all the new subobjects have been inserted.

Table 4 shows the insertion costs of the different insertion sequences by using the $CRLD$ strategy, where $N = 5$ and an object X with $M = 5$, $R = 1$, $Num = 15$, $s =$

Table 4. The insertion costs of 24 different insertion sequences in the $CRLD$ strategy.

insertion sequnce	insertion cost	insertion sequnce	insertion cost	insertion sequnce	insertion cost	insertion sequnce	insertion cost
1 4 24 37	10	4 1 24 37	10	24 1 4 37	12	37 1 4 24	12
1 4 37 24	12	4 1 37 24	12	24 1 37 4	14	37 1 24 4	14
1 24 4 37	12	4 24 1 37	12	24 4 1 37	12	37 4 1 24	12
1 24 37 4	14	4 24 37 1	14	24 4 37 1	14	37 4 24 1	14
1 37 4 24	12	4 37 1 24	12	24 37 1 4	14	37 24 1 4	14
1 37 24 4	14	4 37 24 1	14	24 37 4 1	14	37 24 4 1	14

1 Mbits and $n = 4$. The initial state of the multi-disk drive is shown in Figure 7. Four new subobjects are to be inserted after subobjects 1, 4, 24 and 37, respectively. We input these four new subobjects into disks according to all the possible insertion sequences. From Tables 3 and 4, we observe that the $CRLD$ strategy has a lower insertion cost than the $CRFD$ strategy in all of the insertion sequences. Moreover, from Table 4, we find an interesting observation that the series of insertion operations according to an ascending order of identification numbers (1(1), 5(4), 26(24), 40(37)) requires the minimum insertion cost (10) among these 4! (=24) insertion sequences.

Recall that the $CRLD$ strategy still requires the time to decide the local optimal insertion sequence for every n incoming new subobjects with the time complexity O(n!). From the study of performance analysis in the following subsection, we will prove that the cost for inserting all the new subobjects stored inside the buffer according to an *ascending* order of identification numbers in the $CRLD$ strategy is always lower than the one according to the other insertion sequences. Therefore, the time complexity O(n!) to find an optimal insertion sequence in the *deferring* approach is no longer required in the $CRLD$ strategy.

6.4 Performance Analysis and Simulation Study of the $CRLD$ Strategy

Assume that there are Num subobjects that are striped among N disks and require an aggregation bandwidth with M disks ($M \leq N$), and each of those Num subobjects has a size = s Mbits. We prepare a buffer with size of ($n \times s$) Mbits, where n is the number of deferred subobjects and $n > 0$. Recall that logically, we can view the current state of the object striped on the multi-disk drive as L ($= \lceil \frac{Num}{M} \rceil$) groups numbered from 0 to (L - 1), where the subobjects in each group have to be retrieved, simultaneously. In this section, we show that the optimal insertion sequence in the $CRLD$ strategy is the one according to

the ascending order of identification numbers. First, we consider a simple case with $n = 2$. Next, we consider a general case.

6.4.1 A Simple Case

Let's consider a simple example with $n = 2$ in the $CRLD$ strategy. There are two new subobjects stored inside the buffer when the buffer is full. Suppose these two new subobjects will be inserted after subobjects i_1, i_2, respectively, and $i_1 < i_2$. Consequently, there are 2! (= 2) insertion sequences that are (i_1, i_2) and (i_2, i_1). Let us consider the case with an insertion sequence (i_2, i_1), i.e., a sequence according to the descending insertion order of identification numbers. First, a new subobject is inserted after subobject i_2. Then, the other new subobject is inserted after subobject i_1 (i_1), where the number in "()" denotes the original identification number before any insertion operation occurs. Before group v (= $\lfloor \frac{i_2+2}{M} \rfloor$) (including group v), the probability of a conflict that occurs between a moved-in subobject and one of subobjects in the same group is $\frac{M-1}{N}$ as described before. While after group v (excluding the last group (L - 1)), there are two moved-out subobjects that are moved into each group. The probability of a conflict that occurs between one of these two moved-in subobjects and one of the other (M - 2) subobjects in the same group is $\frac{M-2}{N}$. Let there be FS subobjects stored in the last group (L - 1), the probability of a conflict that occurs between one of these two moved-in subobjects and one of the other FS subobjects is $\frac{FS}{N}$.

Figure 9 shows a logical view of a simple example for inserting two new subobjects 9a and 3a after subobjects 9 and 3, respectively, with an insertion sequence according to the descending order of identification numbers (9, 3). Suppose object X is striped on a multi-disk drive with $N = 6$, where $Num = 16$, $M = 3$, $L = 6$ and $FS = 1$ and the number in "()" denotes the original identification number before any insertion operation occurs. Note that, first, the new subobject 9a is inserted into group 3 with an identification number = 10, which then will be increased by one (i.e., 11) after the other new subobject 3a is inserted into group 1 with an identification number = 4. From this figure, we observe that in group 2 and group 3, the probability P_1 of a conflict which occurs between a moved-in subobject (such as subobject 6(5) in group 2 or subobject 9(8) in group 3) and one of the other two subobjects that are stored in the same group is $\frac{2}{6}$ (= $\frac{M-1}{N}$). (Note that subobject 11(9a) has been stored in group 3 before subobject 4(3a) is inserted.) While in group 4, there are two moved-in subobjects 12(10), 13(11) moved from group 3 and two moved-out subobjects 15(13), 16(14) moved to group 5. Therefore, the probability of a conflict which

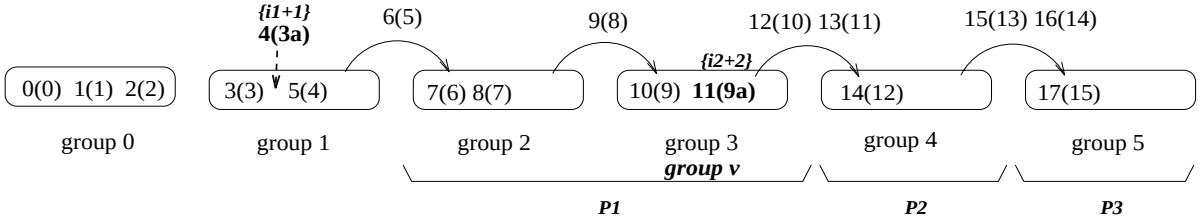

Fig. 9. An example of a logical view of an object X striped on a multi-disk drive after two new subobjects 3a and 9a are inserted with the insertion sequence (9, 3).

occurs between one of these two moved-in subobjects and subobject 14(12) in group 4 is $\frac{1}{6}$ ($=\frac{M-2}{N}$). In the last group, i.e., group 5, there is only one subobject, i.e., subobject 17(15), before any insertion operation occurs. The probability of a conflict which occurs between one of these two moved-in subobjects and subobject 17(15) is $\frac{1}{6}$ ($= \frac{FS}{N}$).

Note that the conflicts between any one of those moved-in subobjects and any one of the subobjects that are stored in the same group are independently. Therefore, the probability P_2 of a conflict is ($2 \times \frac{1}{6}$) ($= \frac{2(M-2)}{N}$) and the probability P_3 of a conflict is ($2 \times \frac{1}{6}$) ($= \frac{2FS}{N}$). In general, the number of conflicts for inserting two new subobjects after subobjects i_1 and i_2, respectively, in the $CRLD$ strategy with the insertion sequence (i_2, i_1), where $i_1 < i_2$, is obtained as

$$D\_NC(i_2, i_1) = (\lfloor \tfrac{i_2+2}{M} \rfloor - \lfloor \tfrac{i_1+1}{M} \rfloor) \times \tfrac{M-1}{N} + (\lfloor \tfrac{Num}{M} \rfloor - \lfloor \tfrac{i_2+2}{M} \rfloor - 1) \times \tfrac{2(M-2)}{N} + 2 \times \tfrac{FS}{N}.$$

That is, the probability P_1 of a conflict in each of those groups (i.e., group ($\lfloor \frac{i_1+1}{M} \rfloor$ + 1), ..., group $\lfloor \frac{i_2+2}{M} \rfloor$) is equal to $\frac{M-1}{N}$, the probability P_2 of a conflict in each of those groups (i.e., group ($\lfloor \frac{i_2+2}{M} \rfloor$ + 1), ..., group $\lfloor \frac{Num}{M} \rfloor$) is equal to $\frac{2(M-2)}{N}$ and the probability P_3 of a conflict in the last group is equal to $\frac{2 \times FS}{N}$.

Let us consider the previous example in the $CRLD$ strategy with the other insertion sequence (i_1, i_2), where $i_1 < i_2$, i.e., a sequence according to the ascending order of identification numbers. Like the previous case, the probability P_1 of a conflict between a moved-in subobject and one of the others (M - 1) subobjects in the same group is $\frac{M-1}{N}$ before group v ($= \lfloor \frac{i_2+2}{M} \rfloor$) (excluding group v), the probability P_2 of a conflict in a group after group v is $2 \times \frac{M-2}{N}$, and the probability P_3 of a conflict in the last group is $2 \times \frac{FS}{N}$, Note that before the second new subobject is inserted after subobject i_2, there are one moved-in subobject k and one moved-out subobject h in group v after the first new subobject is inserted after subobject i_1. During the process of inserting the second new subobject, we will select a

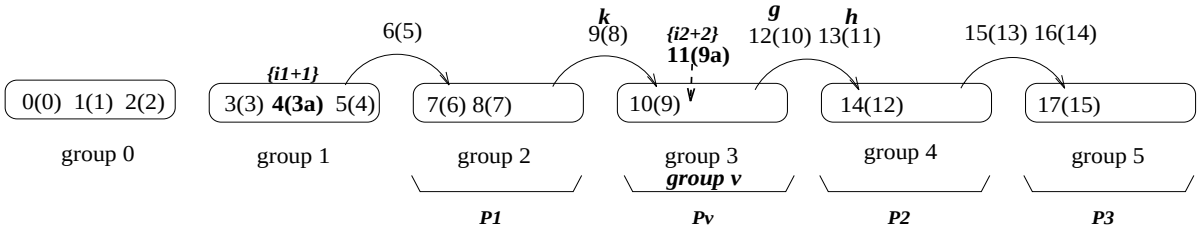

Fig. 10. An example of a logical view of an object X striped on a multi-disk drive after two new subobjects 3a and 9a are inserted with the insertion sequence (3, 9).

disk to store this new subobject such that no conflict occurs between this new subobject (i + 2) and the moved-in subobject k (or one of the subobjects that are stored in group v). Moreover, this insertion causes that one more subobject g has to be moved-out. Therefore, in this case, the moved-in subobject k may conflict only with one of the (M - 2) subobjects that are stored in group v. The probability of such a conflict in group v is $\frac{M-2}{N}$ that is smaller than $\frac{M-1}{N}$ by using the insertion sequence (i_2, i_1) as described before.

Figure 10 shows a logical view of the same example in Figure 9 while those two new subobjects are inserted with an insertion sequence according to the ascending order of identification numbers, i.e., (3, 9). From this figure, we observe that the probabilities P_1, P_2 and P_3 are same as those of the case in Figure 9. Note that in group 3, the moved-in subobject 9(8) may conflict only with subobject 10(9) because that a conflict between the new subobject 11(9a) and subobject 9(8) (or, subobject 10(9)) can be avoided by choosing a disk in which subobjects 9(8) and 10(9) are not stored to store subobject 11(9a). Therefore, the probability of a conflict P_v in group 3 is $\frac{1}{6}$ ($= \frac{M-2}{N}$) that is smaller than $\frac{2}{6}$ ($= \frac{M-1}{N}$) in Figure 9.

Therefore, the number of conflicts for inserting the two new subobjects in the $CRLD$ strategy with the insertion sequence (i_1, i_2) is obtained as

$$D\_NC(i_1, i_2) = (\lfloor \tfrac{i_2+2}{M} \rfloor - \lfloor \tfrac{i_1+1}{M} \rfloor - 1) \times \tfrac{M-1}{N} + \tfrac{M-2}{N} + (\lfloor \tfrac{Num}{M} \rfloor - \lfloor \tfrac{i_2+2}{M} \rfloor - 1) \times \tfrac{2(M-2)}{N} + 2 \times \tfrac{FS}{N}, \quad \text{when } \lfloor \tfrac{i_1+1}{M} \rfloor < \lfloor \tfrac{i_2+2}{M} \rfloor, \text{ or}$$

$$D\_NC(i_1, i_2) = (\lfloor \tfrac{Num}{M} \rfloor - \lfloor \tfrac{i_2+2}{M} \rfloor - 1) \times \tfrac{2(M-2)}{N} + 2 \times \tfrac{FS}{N}, \quad \text{when } \lfloor \tfrac{i_1+1}{M} \rfloor = \lfloor \tfrac{i_2+2}{M} \rfloor.$$

From the above discussion for the simple case of $n = 2$, we observe that the number

of conflicts for the insertion in the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers is smaller than the one with the descending insertion order.

6.4.2 A General Case

In general, we have the following theorem to prove that the number of conflicts for the insertion in the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers is always smaller than those according to all the other insertion orders when $n \geq 2$.

Theorem 2 *Suppose there are n new inserted subobjects that are stored in a buffer, which will be inserted after subobjects* i_1, i_2, ..., i_n, *respectively, where* $i_j < i_{j+1}$ *and* $1 \leq j < n$. *The number of conflicts for inserting these n new subobjects in the CRLD strategy with an insertion sequence according to the ascending order of identification numbers, i,e,* (i_1, i_2, ..., i_n), *is smaller than those according to all the other insertion sequences.*

Proof. We prove this theorem by induction. First, when $n = 2$, we have shown that the number of conflicts for the insertion in the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers is smaller than the one with the descending insertion order as described before. Next, we assume that the optimal insertion order is (i_1, i_2, ..., i_g) when $n = g$. Then, let us consider the case with $n = (g + 1)$ and there are $(g + 1)$ new subobjects that will be inserted after subobjects i_1, i_2, ..., i_g, i_{g+1}, respectively. Suppose the new subobject, say subobject w, that will be inserted after subobject i_{g+1} is the pth performed insertion operation, where $1 \leq p \leq (g + 1)$. Since there are already $(p - 1)$ subobjects that have been inserted before the new subobject w is inserted and all the identification numbers of those $(p - 1)$ subobjects are smaller than the one of subobject w, there are $(p - 1)$ moved-in subobjects in the group where subobject w will be inserted. Therefore, a conflict between any one of the $(p - 1)$ moved-in subobjects and subobject w can be avoided. From the view point of subobject w, the probability of such a conflict is decreased as p is increased. When $p = (g + 1)$, the probability is 0. In this case, all the other g new subobjects have to be inserted before subobject w is inserted. Therefore, the probability of a conflict between the subobject w and one of these g moved-in subobjects is 0. Since the optimal insertion sequence of those g new subobjects is (i_1, i_2, ..., i_g), the optimal insertion sequence for those g new subobjects and subobject w should be (i_1, i_2, ..., i_g, i_{g+1}). Therefore, this theorem is correct. □

6.4.3 Performance Analysis

Consider the case with $n \geq 1$ and $n \leq M$. In the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers, the number of conflicts for inserting the n new subobjects that are inserted after subobjects i_1, i_2, ..., and i_n, respectively, can be obtained as

$$\begin{aligned} DA\_NC_{Num}(n) = & (\lfloor \tfrac{i_2+2}{M} \rfloor - \lfloor \tfrac{i_1+1}{M} \rfloor - 1) \times \tfrac{M-1}{N} + \tfrac{M-2}{N} \\ & + (\lfloor \tfrac{i_3+3}{M} \rfloor - \lfloor \tfrac{i_2+2}{M} \rfloor) \times \tfrac{2(M-2)}{N} + \tfrac{2(M-3)}{N} \\ & + ... \\ & + (\lfloor \tfrac{i_{j+1}+j+1}{M} \rfloor - \lfloor \tfrac{i_j+j}{M} \rfloor) \times \tfrac{j(M-j)}{N} + \tfrac{j(M-j-1)}{N} \\ & + ... + (\lfloor \tfrac{Num}{M} \rfloor - \lfloor \tfrac{i_n+n}{M} \rfloor - 1) \times \tfrac{n(M-n)}{N} + n \times \tfrac{LS}{N}, \\ & \text{where } \lfloor \tfrac{i_j+j}{M} \rfloor < \lfloor \tfrac{i_{j+1}+j+1}{M} \rfloor, \; i_j < i_{j+1}, \; 1 \leq j < n. \end{aligned}$$

That is, the probabilities of a conflict in those groups (after group $\lfloor \frac{i_j+j}{M} \rfloor$ and before $\lfloor \frac{i_{j+1}+j+1}{M} \rfloor$) are $\frac{j(M-j)}{N}$; while the probability of a conflict in group $\lfloor \frac{i_{j+1}+j+1}{M} \rfloor$ is $\frac{j(M-j-1)}{N}$. Note that LS denotes the number of subobjects that are still stored in the last group, say group $(L$ - $1)$, after the n new subobjects have been inserted. When $(n + FS) \leq M$, the n moved-in subobjects may conflict with those LS $(= FS)$ stored subobjects. Therefore, the probability of such a conflict is $(n \times \frac{FS}{N})$. On the other hand, since the number of subobjects in the last group $(L$ - $1)$ is large than M when $(n + FS) > M$, there are $(n +$ FS - $M)$ subobjects have to be moved into the new last group, say group L. In such a case, there are $LS = (FS$ - $(n + FS$ - $M)) = (M$ - $n)$ subobjects that are still stored in group $(L$ - $1)$ and $(n + FS$ - $M)$ subobjects that have to be moved from group $(L$ - $1)$ to the new last group L. Therefore, the probability of a conflict between one of the n moved-in subobjects and any one of those $(M$ - $n)$ subobjects in group $(L$ - $1)$ is $\frac{M-n}{N}$ and the probability of a conflict in group $(L$ - $1)$ is $(n \times \frac{M-n}{N})$. Note that in group L, the probability of a conflict is 0. The reason is that one of those $(n$ - FS - $M)$ moved-in subobjects does not conflict with any one of the subobjects that are stored in group L because there is no subobject in group L, originally.

Moreover, let us consider a case where $n > M$. Since in a group, the probability of a conflict between one of the moved-in subobjects and any one of the subobjects still stored in this group is 0 when the number of the moved-in subobjects is equal to M. The reason is that the M moved-in subobjects will cause M subobjects to be moved out. In this case,

there is no subobject that is still stored in this group because all those M originally stored subobjects have become the M moved-out subobjects. In other words, for every M inserted subobjects, such a group v will appear and the probabilities of a conflict in those groups after group v (and including group v) are 0. Therefore, for every M inserted subobjects, we only consider the probability of a conflict in those groups between group f and group l ($f \leq l$), where the first new subobject is inserted in group f and the Mth new subobject is inserted in group l. Such an independent characteristic per M new data insertions has simplified the way to calculate the total number of conflicts for inserting n ($> M$) new subobjects in the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers. A general formula to compute the number of conflicts for inserting n new subobjects in the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers can be obtained as follows, where $n = kM + r$, $k \geq 0$ and $0 \leq r < k$.

$$GDA\_NC_{Num}(n) = (\sum_{g=0}^{k-1} DA\_NC_{Num+gM}(M)) + DA\_NC_{Num+kM}(r).$$

6.4.4 Simulation Results

Figure 11 shows the relationship between the insertion cost for 50 new subobjects and the size of buffer ($n \times s$) Mbits by using the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers, where $N = 5$, an object X with $M = 5$, $R = 1$, $Num = 50$ and $s = 1$ Mbits. From this figure, in general, as n is increased, the insertion cost is decreased. The reason is that as n is increased, the number of conflicts is decreased. However, from the figure, we also observe that the curve has an oscillation between $n = 5$ and $n = 20$. This anomalous phenomenon is due to the fact that the global insertion sequence for the k ($k > n$) new incoming subobjects is aggregated by the $\lceil \frac{k}{n} \rceil$ local insertion sequences, where each local sequence is the ascending order of identification numbers of every n new incoming subobjects. For example, suppose there are $k = 6$ new incoming subobjects will be inserted after subobjects 1, 2, 4, 3, 5 and 6, respectively. In the case with a buffer with $n = 3$, these 2 local insertion orders are (1, 2, 4) and (3, 5, 6). Therefore, the global insertion sequence with $n = 3$ for these k new subobjects is (1, 2, 4, 3, 5, 6). While in the case with a buffer with $n = 2$, these 3 local insertion orders are (1, 2), (3, 4) and (5, 6) and the global insertion sequence is (1, 2, 3, 4, 5, 6). In the above two cases, we observe that the insertion cost is decreased as n is increased. Specifically, the insertion cost with a large buffer $n = LB$ will be always lower than the one with a

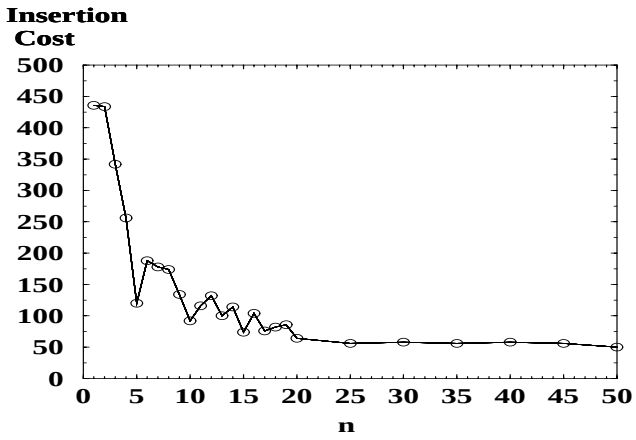

Fig. 11. The relationship between the insertion cost and the size of the buffer (n).

small buffer $n = SB$ when $LB = 2^r \times SB$ and $r \geq 1$. The reason is that every 2^r local insertion sequences with a buffer size $= SB$ will be covered by a local insertion sequence with a buffer size LB, where $LB = 2^r \times SB$ and $r \geq 1$.

Figure 12 shows the relationship between the insertion cost for 50 new subobjects of object X and the size of buffer $(n \times s)$ Mbits by using the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers, where $N = 12$, and an object X with $Num = 120$, $s = 1$ Mbits and $M = 12$, $M = 6$ or $M = 2$, respectively. In all of the cases of three different values of M, as n is increased, the insertion cost is decreased. Moreover, given a fixed n, the insertion cost is increased as M is increased. The reason is that as M is increased, the probability of a conflict in all logical groups is increased, which is $\frac{M-k}{N}$ and $1 \leq k \leq n$, resulting in an increase of the insertion cost.

For the case of a sequence of data deletion, the $CRLD$ strategy will also have better performance than the $CRLD$ strategy. Moreover, different sequences of data deletion will require the same deletion cost. The reason is that the probability of a conflict for the latter deleted subobject will not be affected by the previously deleted subobjects although the corresponding identification numbers of subobjects have to been decreased. In other words, the final state of the multi-disk drive after a series of deletion operations according to any deletion sequence is always the same.

7 Conclusion

In this paper, we have proposed an efficient *conflict-resolution* approach to the insertion/deletion operations on continuous media that are striped into a multi-disk drive with-

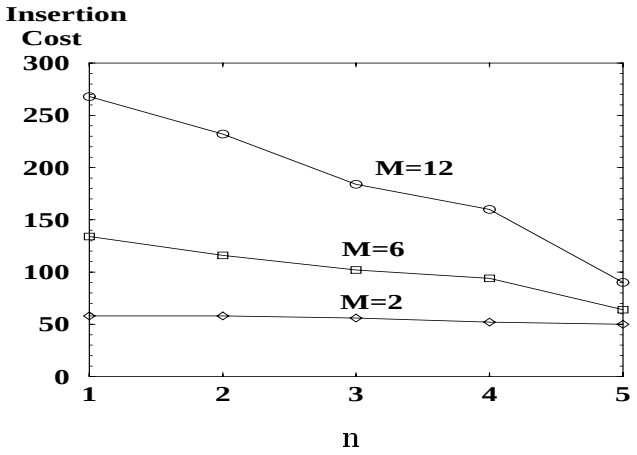

Fig. 12. The relationship between the insertion cost and the size of the buffer (n) with different values of M.

out reorganization of the whole data. Only when a conflict occurs, one movement operation is required. From our performance analysis and simulation results, we have shown that the average number of additional movements for any subobject insertion is no more than 5, when there is an object that is divided into 120 subobjects and is striped on a multi-disk drive with 12 disks. Moreover, a *deferring* approach has been proposed to reduce those additional movement cost at the cost of an additional buffer. Based on this approach, two strategies are proposed. One is called the *conflict-resolved-first-deferring* strategy (the $CRFD$ strategy), and the other one is called the *conflict-resolved-last-deferring* strategy (the $CRLD$ strategy). From the performance analysis and simulation results, we have shown that the $CRLD$ strategy has a lower insertion cost than the $CRFD$ strategy. We also have proved that the the $CRLD$ strategy with an insertion sequence according to the ascending order of identification numbers is an optimal strategy based on the proposed *deferring* approach. Moreover, in the $CRLD$ strategy, in general, as n is increased, where n is the number of deferred subobjects, the insertion cost is decreased. Furthermore, given a fixed n, the insertion cost is decreased as M is decreased, where M is the number of disks which have to operate synchronously to support the required bandwidth of an object. One of the most important challenges in a video server is to support *interactive browsing* functions such as *fast forward* and *fast backward*. How to support the continuous display of multiple objects at different display speed rates, simultaneously, without any additional resource, will be an important research direction.

References

[1] S. Berson, S. Ghandeharizadeh, R. Muntz and X. Ju, "Staggered Striping in Multimedia Information Systems", *ACM SIGMOD*, 1994, pp. 79-90.

[2] J. F. K. Buford, "Multimedia File Systems and Information Models", in *The Book of Multimedia Systems*, Ed. J. F. K. Buford, Addison-Wesley, 1994.

[3] M. S. Chen, D. D. Kandlur and P. S. Yu, "Storage and Retrieval Methods to Support Fully Interactive Playout in a Disk-Array-Based Video Server", *ACM Multimedia Systems* **3** (1995) 126-135.

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

[5] J. Gemmell and S. Christodoulakis, "Principles of Delay-Sensitive Multimedia Data Storage and Retrieval", *ACM Transactions on Information Systems* **10**, 1 (1992) 51-90.

[6] D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan and L. A. Rowe, "Multimedia Storage Servers: A Tutorial", *IEEE Computer* (May 1995) 40-49.

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

[8] S. Ghandeharizadeh and L. Ramos, "Continuous Retrieval of Multimedia Data Using Parallelism", *IEEE Transactions on Knowledge and Data Engineering* **5**, 4 (1993) 658-669.

[9] K. Keeton and R. H. Katz, "Evaluating Video Layout Strategies for a High-Performance Storage Server", *ACM Multimedia Systems* **3** (1995) 43-52.

[10] J. C. L. Liu, D. H. C. Du and J. A. Schnepf, "Supporting Random Access on Real-Time Retrieval of Digital Continuous Media", *Computer Communications* **18**, 3 (1995) 145-159.

[11] P. Lougher and D. Shepherd, "The Design of a Storage Server for Continuous Media", *The Computer Journal* **36**, 1 (1993) 33-42.

[12] B. Ozden, R. Rastogi and A. Silberschatz, "On the Design of a Low-Cost Video-On-Demand Storage System", *ACM Multimedia Systems* **4** (1996) 40-54.

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

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

[15] P. V. Rangan, H. M. Vin and S. Ramanathan, "Designing an On-Demand Multimedia Service", *IEEE Communications Magazine* (July 1992) 56-64.

[16] P. V. Rangan and H. M. Vin, "Efficient Storage Techniques for Digital Continuous Multimedia", *IEEE Transactions on Knowledge and Data Engineering* **5**, 4 (1993) 564-573.

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

[18] R. Steinmetz, "Multimedia File Systems Survey: Approaches for Continuous Media Disk Scheduling", *Computer Communications* **18**, 3 (1995) 133-144.

[19] H. M. Vin and P. V. Rangan, "Designing a Multiuser HDTV Storage Server", *IEEE Journal on Selected Areas in Communications* **11**, 1 (1993) 153-164.

[20] C. Yu, W. Sun, D. Bitton, Q. Yang, R. Bruno and J. Tullis, "Efficient Placement of Audio Data on Optimal Disks for Real-Time Applications", *Communications of ACM* **32**, 7 (1989) 862-871.