

Dynamic Reallocation of Multimedia Data Objects*

Cyril U. Orji[†], Naphtali Rishen[†], Donald A. Adjero[‡], Kingsley C. Nwosu[§]
{orji,rishen}@fiu.edu donald@cs.cuhk.hk nwosuck@harpo.wh.att.com

Abstract

The need for efficient and reliable satisfaction of the inherent and processing requirements of multimedia objects has made object storage and retrieval one of the challenging problems in object-based information systems. However, due to many other characteristics of multimedia objects, we need to be able to dynamically alter the storage locations of objects in order to maintain as often as possible, an optimal location for each object. Many conditions may influence object relocation. Some of these include a new size for an object due to object growth, change in object's access frequency, change in degree in utilization of storage devices, change in bandwidth requirements of the object, among others. In this paper we discuss the different properties and requirements of an object which necessitate its decomposition and reallocation. We present the conditions and criteria for determining the number and sizes of an object's segments and allocatability of each segment to a particular storage device. A detailed discussion on the relocation issues, strategies, and pertinent algorithms is also presented.

Keywords: access frequency, decomposition, multimedia, retrieval, reallocation, replication.

1 Introduction

Over the past few years, tremendous improvements in computer technology have made possible advanced applications utilizing disk arrays, high speed networks and high performance compression and coding algorithms. Some of these advanced applications include multimedia services in the form of video-on-demand systems, electronic news distribution and many of the numerous applications currently available over the internet. These new applications have, however, created new challenges for researchers. In particular, given the huge amount of data associated with multimedia applications, multimedia object management has emerged as one of the interesting and important topics for investigation within the framework of these new technologies. Issues in object management include but

are not limited to storage, retrieval, synchronization, consistency, and transport.

Storage allocation issues for multimedia systems have attracted a lot of attention [3, 4, 7, 9, 8]. Most of the techniques and strategies that have been proposed to address the issue [2, 5, 6] are extensions to techniques used in traditional (non-multimedia) data processing environments [1, 5, 10]. Most of these techniques only account for the size and/or bandwidth requirements of the objects while ignoring the dynamic access impacts on the utilization and efficiency of the storage devices.

Specifically, as the frequency of access to objects in a storage device changes, so does the efficiency of the utilization of the storage device. As access frequency changes, the latency of an I/O may be adversely affected, thereby impacting system performance. Consequently, it becomes important to relocate objects previously optimally placed, when some of their initial characteristics exceed or fall below some established thresholds. It should be possible to migrate objects from one location to another, and this migration should be done dynamically and transparently to the user, without performance loss.

In this paper, we discuss the initial allocation and dynamic reallocation issues of objects in multimedia information systems. The different properties and requirements of an object which necessitate its decomposition and reallocation for parallel retrievability are described. These include the size of the object, the data availability rate, and parallel processing requirements. We present the conditions and criteria for determining the number and sizes of the segments and allocatability of each segment to a particular storage device.

The remainder of the paper is organized as follows. In Section 2 factors that necessitate the decomposition of objects are discussed. Object decomposition in a heterogeneous storage device environment is also discussed. In Section 3 we discuss object reallocation issues and factors that necessitate object reallocation. In Section 4 we present the various object reallocation algorithms and briefly discuss replication and reallocation issues. Our conclusions are presented in Section 5.

2 Object Decomposition

Dynamic object reallocation requires some scheme for object decomposition. The decomposition of multimedia objects results in smaller objects and thus provides a finer granularity of control for both the objects to be reallocated and for the possible target devices. It

*This work has been supported in part by grants from NASA (NAGW-4080), ARO (BMDO grant DAAH04-0024) and NSF (IRI-9409661).

[†]High Performance Database Research Center, School of Computer Science, Florida International University, Miami, FL 33199.

[‡]Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

[§]Lucent Technologies, (AT&T Bell Labs.), 67 Whippany Road, Whippany, NJ 07981-0903.

is to be expected that some information on the usage of the different segments of an object could be helpful in determining the appropriate placement or location for the new objects. In this section we study the storage requirements of objects which foster their parallel retrievability. We discuss the properties, characteristics, or processing requirements of an object which impact its storage. We also present conditions for determining the number and sizes of the segments and allocation of each data/object segment to a particular storage device. The discussion in this section is based primarily on the object decomposition techniques proposed in [7].

2.1 Factors that influence object decomposition

Characteristics that necessitate the decomposition of a large object include the size of the object, the data availability rate, and the parallel processing requirements.

2.1.1 Size of object

The digitization of video signals results in objects of enormous sizes. Different encoding standards have been established to reduce the sizes of these objects. Some of these include the MPEG-I (Motion Picture Expert Group) and MPEG-II encoding standards. Despite these encoding schemes, multimedia objects still have enormous sizes. For example a 2 hour movie encoded with MPEG-II results in a 21.6 Gb object.

2.1.2 Data availability rates

Most multimedia objects require a minimum amount of data utilization per unit time. The data utilization rate is the amount of data consumed per unit time to satisfy operating requirements. In this paper, we refer to *the throughput requirement of an object as its Data Availability Rate (DAR)*. Assuming that network latency is negligible, the satisfiability of an object's data availability rate depends primarily on the bandwidths of the storage devices.

The bandwidth of a device is the amount of data that can be retrieved from the storage device per unit time under an optimal allocation policy.

2.1.3 Parallel processing

In some parallel or distributed computing environments, different parts of an object may be needed by different machines or computing systems simultaneously. In other to localize object storage, the object must be decomposed and stored in the storage devices attached to different systems. Such a storage technique allows for parallel access to the data potentially improving the I/O rate by a factor of n , where n is the number of devices accessed in parallel.

2.1.4 Network implications

When satisfying the data availability rate of a multimedia object, we assume that the data is being trans-

ported only from permanent storage via the server to the client without any intervening transport medium. However, nowadays, the configuration usually involves various users connected to one or more servers via a Local, Metropolitan or Wide Area Network (LAN, MAN, WAN). In that case, even if the I/O devices to server data transmission satisfies the data availability rate, the network latency could present problems.

The bandwidth of a network is the amount of data that it can deliver from source to destination per unit time. The bandwidth of any network is affected by the amount of traffic (requests) generated per unit time. Retransmissions due to data loss while in transit over the network also affect the volume of traffic. For certain multimedia data types, especially those that are continuous, data retransmissions adversely affect on-time data delivery and utilization.

2.2 Determining storage units

For each object that must be decomposed, the number of Storage Units (SUs) and size of each SU must be determined. An SU is the amount of data that is allocated to a single storage device or site. SUs are analogous in some sense to fragments of an object. However, these fragments may be of unequal sizes. Given the heterogeneous storage device environment we have assumed, it is common to consider the *number* of SUs, or the number of different fragments of an object. Similarly, it is also common to consider the *size* of each SU. The *number* of SUs depends on the (i) size of an object, (ii) number of usable storage devices, (iii) DAR of an object, and (iv) degree of parallelism, while the *size* of each SU depends on the (i) bandwidth of the usable storage devices, (ii) free/available space on the storage devices, and (iii) degree of parallelism.

If the object cannot fit in a single storage device, the number of SUs and sizes of SUs solely depend on the size of the object and the number of usable storage devices.

Consider a heterogeneous environment with eight storage devices SD_1 to SD_8 . We group all identical devices into a storage device group (SDG), yielding (for our example), the following three groups:

- $SDG_1 = \{SD_1, SD_2, SD_3\}$
- $SDG_2 = \{SD_4, SD_5\}$
- $SDG_3 = \{SD_6, SD_7, SD_8\}$

The *storage set (ss)* of an object is the set of the *numbers of storage devices needed to achieve the object's DAR*. Given a heterogeneous environment, multiple storage sets are possible. Therefore, the size of an object's storage set is bounded by $2^w - 1$, where w is the number of different device storage groups, and hence different bandwidth values for the storage devices in the system. For our working example with $w = 3$, the size of the storage set = 7. Formally, $ss_k = \{y_1, y_2, \dots\}$ is the k th storage set of an object, and y_i is the number of storage devices from storage device group i (SDG_i) needed to satisfy the DAR of the object.

We enumerate below key aspects of our decomposition algorithm.

1. Given a storage set, every device group that belongs to the set must have at least a device in the set. In other words, a storage set cannot have a member with zero participation.
2. Given a storage set, we wish to minimize the amount of data transferred by the devices in each group with the constraint that the DAR is satisfied.
3. Given that the DAR of an object has been met, the amount of data retrieved should not exceed that required to satisfy the DAR by an amount greater than the amount of data contributed by any of the devices – in that event we could do without the device.
4. The storage device groups (SDGs) are arranged in order of decreasing bandwidth. Thus to satisfy a given DAR, the number of devices required are by implication arranged in increasing numerical order.
5. A storage set cannot have a device group that contains more than the available number of devices in the group.
6. A storage set is valid if removing any member of a storage device group causes any of the preceding minimization rules to be violated.

For example, given DAR_j and $ss_k = \{y_1, y_2, y_3\}$, the following conditions must hold:

1. $BW(SDG_1) > BW(SDG_2) > BW(SDG_3)$,
2. $\left[\frac{y_1 BW(SDG_1)}{y_3 BW(SDG_3)} + \frac{y_2 BW(SDG_2)}{y_3 BW(SDG_3)} \right] \geq DAR_j$ and
 - (a) $\left[\frac{(y_1 - 1)BW(SDG_1)}{y_3 BW(SDG_3)} + \frac{y_2 BW(SDG_2)}{y_3 BW(SDG_3)} \right] < DAR_j$,
 - (b) $\left[\frac{y_1 BW(SDG_1)}{y_3 BW(SDG_3)} + \frac{(y_2 - 1)BW(SDG_2)}{y_3 BW(SDG_3)} \right] < DAR_j$,
 - (c) $\left[\frac{y_1 BW(SDG_1)}{y_3 BW(SDG_3)} + \frac{y_2 BW(SDG_2)}{y_3 BW(SDG_3)} + \frac{(y_3 - 1)BW(SDG_3)}{y_3 BW(SDG_3)} \right] < DAR_j$.
3. $y_1 \leq y_2 \leq y_3$.

If any of the conditions above is violated, then the corresponding storage set is invalid. The above conditions are enshrined in the following integer linear programming problem:

$$\begin{aligned}
 y_1 BW(SDG_1) + y_2 BW(SDG_2) + y_3 BW(SDG_3) &\geq DAR_j, \\
 y_1 \leq y_2, \quad y_2 \leq y_3, \quad y_3 > 0. &\quad (1)
 \end{aligned}$$

An ss_k with $|ss_k| = g$ is *acceptable* if

- (1) $\sum_{i=1}^g y_i \leq m$, and
 - (2) $\forall j [j = 1 \dots g] y_j \leq |SDG_j|$
- where SDG_j is the set of homogeneous storage devices: $\{SD_1, \dots, SD_h\}$ for some h .

For example, consider an object of size 120KB and bandwidth requirement of 60KB/s. Moreover,

$SDG_1 = \{SD_1, SD_2, SD_3\}$, $SDG_2 = \{SD_4, SD_5\}$, $SDG_3 = \{SD_6, SD_7, SD_8\}$, $BW(SDG_1) = 30KB/s$ (i.e. the bandwidth of each of the devices in the group), $BW(SDG_2) = 20KB/s$, and $BW(SDG_3) = 10KB/s$. Since there are three storage device groups ($w = 3$), there are $2^3 - 1$ or 7 possible permutations of groups of storage devices numbered ss_1, \dots, ss_7 . These are respectively, $\{SDG_1\}$, $\{SDG_2\}$, $\{SDG_3\}$, $\{SDG_1, SDG_2\}$, $\{SDG_1, SDG_3\}$, $\{SDG_2, SDG_3\}$, and $\{SDG_1, SDG_2, SDG_3\}$. The possible storage sets are:

- $ss_1 = \{2\}$: any two of the three devices from SDG_1 .
- $ss_2 = \{3\}$: three devices from SDG_2 .
- $ss_3 = \{6\}$: six devices from SDG_3 .
- $ss_4 = \{1, 2\}$: one device from SDG_1 and two from SDG_2 .
- $ss_5 = \{1, 3\}$: one device from SDG_1 and three from SDG_3 .
- $ss_6 = \{2, 2\}$: two devices from SDG_2 and two from SDG_3 .
- $ss_7 = \{1, 1, 1\}$: one from each of the groups.

Obviously, ss_2 , and ss_3 are invalid. Each requires more than the number of available devices. Furthermore, without the constraints discussed above, it is evident that given $\{SDG_2, SDG_3\}$, the storage sets $\{3, 0\}$, $\{1, 4\}$, $\{0, 6\}$, $\{2, 3\}$, and $\{1, 5\}$ can achieve the I/O or display requirements. However, applying the constraints limits the option to $\{2, 2\}$. $\{3, 0\}$, $\{0, 6\}$ violate the zero membership constraint, $\{1, 4\}$, $\{1, 5\}$ require more than the available number of devices in SDG_3 , and $\{2, 3\}$ uses one unit from SDG_3 that is unnecessary. If none of the storage sets of an object is acceptable, then we cannot allocate the object.

Note that using storage set six (ss_6) means that the object fragments are stored such that they can be retrieved in units of 20KB in SDG_2 and units of 10KB in SDG_3 . Clearly, adequate meta information on storage unit allocation must be kept to prevent data loss.

2.3 Allocating storage units

After the storage devices have been determined, there is often the need to make judicious choices of the specific devices that would be used in a storage set. In the example of the preceding paragraph two devices out of the three devices that belong to SDG_3 are needed. Which two to choose will very likely impact system performance. Although a detailed treatment of the factors that influence these choices cannot be undertaken here (see [6]), one important requirement for storing an object in a storage device is the availability of contiguous space to accommodate the object. In some conventional techniques, an object may be fragmented into blocks of static or variable sizes and stored in different locations in storage device wherever space is available. Some techniques for

increasing system performance by minimizing I/O latencies, require that object be contiguously allocated at some optimally determined locations in the storage device.

3 Object Reallocation Issues

Having stored an object, certain conditions might necessitate its reallocation to one or more storage devices. During reallocation, a determination must be made whether the whole object or parts of it have to be reallocated. Furthermore, it must be determined where to reallocate the object (or its subparts), and how to do so efficiently.

A number of factors may necessitate object reallocation. These include:

- object growth or change in size of object,
- change in frequency of access of object or parts of it,
- change in device utilization, and
- change in display (bandwidth) requirement of the object.

For a class of multimedia applications, such as VOD systems, object reallocation must be done dynamically for a number of reasons. For instance to avoid possible system failure due to one or more reasons as enumerated for reallocation - high access frequency, over utilization, etc. Another could be to ensure that the reallocation is transparent to the user - that is the user's satisfaction should not be compromised because of the reallocation process, thus the user should not be kept waiting for some objects (because the objects are being reallocated) before using the services provided by the multimedia system. This second point is necessitated by the need to prevent performance degradation during object reallocation. Moreover, one characteristic property of VOD systems is "non-stop processing". In other words, the system is never brought down for backup, or other administrative activities. In such an environment, dynamic object reallocation becomes a necessity.

Object growth

By object growth is meant the possible change in size of an object. Typically, one reason for which objects may grow could be the need to satisfy different QoS requirements, as may be required by the user. Since the user is allowed dynamic interaction, it means that he could dynamically change his quality of service requirements, and thus objects (or some segments of an object) may grow dynamically. When an object grows, the resultant size of the affected SU may require a reallocation. To this end, the allocation algorithm must be re-run to determine the new optimal allocation. For example, if an object grows such that a new segment is generated, then the size of the applicable SU has increased and so the current storage device for the SU may not provide an optimal allocation.

Access frequency

Besides growth, another prevalent condition for object reallocation is frequency of access. When the current access frequency of an object has exceeded some defined threshold in a system, relative to its frequency at last allocation, then the object may have to be reallocated. The reallocation may also be applicable when the access frequency has fallen below a threshold. Exceeding or falling below a threshold indicates that the current allocation may not be optimal anymore. Realistically, in the case of falling below a threshold, the object may need to be moved to an archival storage such as a tape device.

Device utilization

On the other hand, it may happen that none of the objects stored in a storage device has exceeded the threshold, however, the utilization level of the storage device may be relatively high. It may be useful to note how reallocation needs arising from device utilization differ from those due to access frequency. A storage device could have many SUs, with some possibly belonging to different objects. The problem of device utilization then results from the cumulative effect of high access frequency over all the SUs stored on the device. In order to balance the load and hence the utilization of storage devices, it is necessary that one or more objects from a storage device be moved to other storage devices. The problem again becomes the criteria for selecting objects for migration. Migration issues are beyond the scope of this paper.

This kind of problem can also be solved by object replication. In such cases, the load of a storage device is reduced by directing accesses to the target object's replicas in other storage devices. Based on the number of current accesses to the object, the storage devices for the replicas may also be over-utilized. Redirecting requests to replica locations may exacerbate the situation. Invariably, one or more objects must be reallocated. To minimize the number of objects that may need to be reallocated, the Most Frequently Accessed Objects (MFAOs) have to be considered for migration and, consequently, normalize the utilization level. Clearly, other migration policies such as those based on Least Frequently Accessed Objects (LFAO), etc. could be used, depending on the cost model adopted. So far, the MFAO have been found to be most appealing for the purpose. With an efficient migration and reallocation algorithm, such strategies for improving the utilizations of storage devices also apply to load distribution to single nodes in distributed or clustered systems.

Bandwidth requirement

Like the object growth, changing the QoS specification could result in changes in the object's data availability rate during presentation. For instance, when the same object is requested but with a different QoS specification as compared with previous request(s), this may affect the bandwidth. This could be typical in a VOD system that supports multiple levels of service, such as serving video at different resolutions (possibly both temporal and spatial), say in response to requests for special user interaction (fast

forward, etc.). Another reason for possible dynamic changes in bandwidth requirement is when there is an increase in the number of concurrent users who cannot be serviced with the same copy of the video.

Reconsider the example of Section 2.2. For ease of reference, there are eight storage devices SD_1, SD_2, \dots, SD_8 distributed in three storage groups SDG_1, SDG_2, SDG_3 . SD_1, SD_2, SD_3 belong to SDG_1 , and each supports a bandwidth of $30KB/s$. SDG_2 has SD_4, SD_5 and each has a bandwidth of $20KB/s$. SDG_3 has SD_6, SD_7, SD_8 and each has a $10KB/s$ bandwidth. We found that to support a multimedia object with a display requirement of $60KB/s$, the best allocation was to use two devices each from SDG_2 and SDG_3 .

Suppose now that the bandwidth requirement for this object changes to $80KB/s$. The current allocation is clearly inappropriate. The allocation algorithm shows that only three of the seven storage sets are valid. These are:

- ss_1 : three devices from SDG_1 ,
- ss_5 : two devices each from SDG_1 and SDG_3 ,
- ss_7 : one device from each of SDG_1 and SDG_2 , and three from SDG_3 .

The need to reallocate the object is thus demonstrated.

4 Reallocating Objects

We now describe in more detail the various models and algorithms for object reallocation as discussed above. The following parameters and notations are used.

O_i	= the i th multimedia object,
O_i^j	= the j th subobject or segment of O_i ,
S_i	= the size of O_i (i.e., $ O_i $),
S_i^j	= the size of O_i^j (i.e., $ O_i^j $),
DAR_i	= the data availability rate of O_i ,
DAR_i^j	= the data availability rate of O_i^j ,
Q_i	= the amount of data associated with DAR_i ,
Q_i^j	= the amount of data associated with DAR_i^j ,
SD_k	= the k th storage device, and
$BW(SD_k)$	= the bandwidth of SD_k .

Therefore, an object O_i can be stored in m storage devices (assuming the existence of m subparts/subobjects of O_i) if

$$\sum_{k=1}^m BW(SD_k) \geq DAR_i \quad (2)$$

Other parameters and notations used include:

$O_i^{\#sus}$:	Number of SUs associated with O_i .
SU_i^j :	j th SU of O_i .
$SIZE(SU_i^j)$:	Size of SU_i^j (current size at most recent allocation)
$N_SIZE(SU_i^j)$:	New size of SU_i^j after growth.
$AF(O_i)$:	Access frequency of O_i at last allocation.
$N_AF(O_i)$:	Current access frequency of O_i .
\mathcal{T}_o^+ :	Threshold for an object's access frequency when the object is frequently accessed.
\mathcal{T}_o^- :	Threshold for an object's access frequency when the object is infrequently accessed.
SD^{heat} :	Current utilization level of storage device. This is expressed as a function of the average number of requests satisfied per unit time and the size of the waiting queue over a unit time.
\mathcal{T}_d :	Threshold for current utilization of a storage device. \mathcal{T}_d is the expected maximum SD^{heat} of a storage device.

When one or more objects need to be reallocated, the target storage devices for the objects are determined such that the resultant load distribution among the devices is relatively balanced. The balancing strategy assumed here is the one described in [6] which uses a minimization of the cumulative allocation cost of the SUs via bipartite matching. In other words, the minimal possible cost of the allocation based on the permutations of the SUs to different storage devices is used as the optimal solution. All the SUs of an object that need to be reallocated are done concurrently. In the rest of the paper, we denote the function that determines the new optimal allocation of a set of SUs as $\mathcal{F}(SU_1, SU_2, \dots)$. The function \mathcal{F} returns a set of storage devices \mathcal{R} . When an SU's current allocation device is different from the newly determined optimal device, then the new SU is stored before the old copy is deleted. The deletion involves both the actual data and meta data such as its effect on the utilization factor of the storage device. We use the function $RE_STORE(SU_1, SU_2, \dots, SD_1^{cur}, SD_2^{cur}, \dots, SD_1^{new}, SD_2^{new}, \dots)$ (where SD_1^{cur}, SD_1^{new} are the current and new storage devices for SU_1 , respectively) to indicate the actual storing of new SUs and deletion of existing ones.

4.1 Growth of object

When an object grows, one or more of the SUs may be affected. It is also possible for all the SUs that comprise the object to be affected due to growth. Under such circumstances, only the SUs that are affected by an extension or growth are considered for reallocation. The reallocation algorithm (Algorithm 4.1) reallocates some SUs of an object, O_i , when the object experiences growth.

```

Begin
 $\mathcal{G}, \mathcal{T} = \emptyset$  //  $\mathcal{G}$  is the set of the SUs of  $O_i$  which have
// grown.  $\mathcal{T}$  is the set of the storage devices
// where elements of  $\mathcal{G}$  are stored.
 $\forall j [j = 1, \dots, O_i^{\#sus}]$  do
  If ( $N\_SIZE(SU_i^j) > SIZE(SU_i^j)$ )
     $\mathcal{G} = \mathcal{G} \cup SU_i^j$  // Collect applicable SUs
     $\mathcal{T} = \mathcal{T} \cup SD_k$  such that  $SU_i^j \Rightarrow SD_k$ 
    // Save SUs' current allocations
  Endif
Enddo
 $\mathcal{R} = \mathcal{F}(\mathcal{G})$  // Determine new optimal allocations.
//  $\mathcal{R}$  is the set of the optimal storage
// the optimal storage devices for  $\mathcal{G}$ 
 $RE\_STORE(\mathcal{G}, \mathcal{T}, \mathcal{R})$  // Store and delete SUs
Endbegin

```

Algorithm 4.1: **Reallocation as a result of growth of object**

We assume that at initial allocation, $N_SIZE(SU_i^j) = SIZE(SU_i^j)$ for $1 \leq j \leq O_i^{\#sus}$.

4.2 Change in access frequency of object

The need to reallocate an object may be determined by a *daemon*¹ process which periodically checks the current access frequencies of the objects and determines which levels of access have either exceeded or fallen below the given thresholds since the last allocation. At specific time intervals, the daemon process compares the difference between an object's access frequency at last allocation and current access frequency with the established thresholds. If the difference is greater than the applicable threshold, then the object has been experiencing considerable number of accesses. On the other hand, if the currently computed access frequency is less than the access frequency at last allocation, and the difference is less than the applicable threshold, then the object is infrequently used. It, therefore, may need to be stored on a tape.

It may not be reasonable to reallocate an entire object after satisfying one of the above conditions. Sometimes, certain parts of an object are more frequently accessed than others, e.g., a digital video object may be descriptively captivating, but that may only apply to the first several minutes of the movie. As a result, most of the viewers may not have the patience to see the middle or ending of the movie. In this example, the access frequencies of the different segments of the video object may be used to determine the granularity of reallocation. Although it may be very beneficial to associate access frequencies to SUs, the potential performance implications may be very consequential as a result of the amount of data to store and maintain. In a system with hundreds of objects, that approach may not be costly; however, with thousands of objects, it may pose a problem. In developing the algorithm, we assume that an entire object is reallocatable. Once it is determined that an object needs to be reallocated, its reallocation is initiated independent of other reallocations. The reallocation of objects may be handled

¹ a background process either waiting for some event to occur, or waiting to perform some specified task on a specific basis[11].

by lightweight² processes where each process is responsible for a single object's reallocation. Algorithm 4.2 shows the description of the daemon process where *timeint* is the time interval between each iteration of the daemon. The sets of objects and their current allocations are saved in global data structures which can be accessed by other functions.

```

Begin
  sleep(timeint)
   $\mathcal{G}, \mathcal{T}, \mathcal{G}', \mathcal{T}' = \emptyset$ 
  //  $\mathcal{G}$  is the set of the SUs of an object which are frequently
  // used.  $\mathcal{T}$  is the current storage locations of  $\mathcal{G}$ ;
   $\mathcal{G}'$  is the set of SUs of an object that are infrequently used;
  //  $\mathcal{T}'$  is the current storage locations of  $\mathcal{G}'$ 
   $\forall i [i = 1, \dots, O_i^{\#sus}]$  do
     $temp = N\_AF(O_i) - AF(O_i)$ 
    // Compute difference in access frequency
    If ( $temp > 0 \ \& \ temp > \mathcal{T}_\sigma^+$ )
      // Is difference > threshold?
       $\mathcal{G} = \mathcal{G} \cup_{j=1}^{O_i^{\#sus}} SU_i^j$ 
      // Collect all SUs of an object
       $\mathcal{T} = \mathcal{T} \cup_{k=1}^{O_i^{\#sus}} SD_k$  where  $SU_i^j \Rightarrow SD_k$ 
      // Obtain their current allocations
    Elseif ( $temp < 0 \ \& \ temp > \mathcal{T}_\sigma^-$ )
      // Object is infrequently used
       $\mathcal{G}' = \mathcal{G}' \cup_{j=1}^{O_i^{\#sus}} SU_i^j$ 
      // Collect all SUs of an object
       $\mathcal{T}' = \mathcal{T}' \cup_{k=1}^{O_i^{\#sus}} SD_k$  where  $SU_i^j \Rightarrow SD_k$ 
      // Obtain their current allocations
    Endif
  Enddo
   $\mathcal{R} = \mathcal{F}(\mathcal{G})$  // Determine new optimal allocations.
  //  $\mathcal{R}$  is the set of the optimal storage
  // the optimal storage devices for  $\mathcal{G}$ 
   $RE\_STORE(\mathcal{G}, \mathcal{T}, \mathcal{R})$  // Store and delete SUs
Endbegin

```

Algorithm 4.2: **Reallocation due to changes in access frequency**

4.3 Change in utilization level of storage device

The allocation algorithm for the objects takes into account the cumulative access factors of all the objects stored in a storage device. However, as the access frequencies of the objects change, the cumulative effect may render a storage device over-utilized relative to others; while, none of the objects may have exceeded the threshold. Therefore, we need to identify the objects which have mostly contributed to this imbalance and, subsequently, reallocate them. We have established that it is efficient to identify and reallocate the MFAOs by extracting the objects whose access frequencies, since last allocation, have increased the most. The objects, in this case, may refer to SUs or whole objects (for very small objects, when applicable). The reallocation process initially determines

² a number of processes running concurrently with little non-shared space.

that a particular storage device have exceeded the required threshold. Consequently, the most current results from the daemon process are used to determine the objects that have contributed to the imbalance. The objects that have exceeded the threshold are reallocated. When none of the objects has exceeded the threshold, then the MFAOs for the storage device are reallocated. Algorithm 4.3 shows the description of the algorithm that reallocates objects to maintain relative balance among the storage devices in a system.

```

Begin
   $\forall j [j = 1, \dots, m]$  do
    If  $(SD_j^{heat} > \mathcal{T}_d)$ 
      // Compare current utilization with threshold
       $\mathcal{G}', \mathcal{T}'' = \emptyset$ 
      //  $\mathcal{G}$  is the set of all the SUs in the system
       $\forall SU \in \mathcal{G}$  if  $(SU \Rightarrow SD_j)$ 
        // Are there SUs stored in this device?
         $\mathcal{G}' = \mathcal{G}' \cup SU$  // Collect the SUs
         $\mathcal{T}'' = \mathcal{T}'' \cup SD_j$  // Save current locations
      If  $(\mathcal{G}' \neq \emptyset)$  // If we found some SUs...
         $\mathcal{R} = \mathcal{F}(\mathcal{G}')$  // Determine new optimal locations
         $RE\_STORE(\mathcal{G}', \mathcal{T}'', \mathcal{R})$  // Store and delete SUs
      Else // No SU of the SD exceeded the threshold
        While  $(SD_j^{heat} > \mathcal{T}_d)$  do
          // Balance SD's utilization
           $temp = SU_k^l$  such that  $\forall SU \Rightarrow SD_j$ 
             $N\_AF(SU_k^l) > N\_AF(SU)$ 
          If  $(temp = \emptyset)$  return
            // No more SD to process
             $\mathcal{R} = \mathcal{F}(temp)$ 
            // Determine new optimal locations
          If  $(temp = \mathcal{R})$  continue
            // Current location is still optimal
             $RE\_STORE(temp, SD_j, \mathcal{R})$ 
            // Store and delete SUs
            // Re-compute  $SD_j^{heat}$ 
        Endwhile
      Endelse
    Endif
  Enddo
Endbegin

```

Algorithm 4.3: **Reallocations to balance storage devices**

4.4 Object replication and object reallocation

Replication of an object in any computing system serves multiple purposes. First, from the user's point of view, multiple copies of an object provide the opportunity for increased availability. From the system point of view, some form of replication is more than convenient; it is essential for both system availability and performance.

The overriding requirement for replication is that different replicas of the same object or part of an object reside on failure-independent locations. In other words, the availability of one copy is not affected by the availability of other copies, but that each copy's

availability is location dependent. Above all, it is imperative that object replication be transparent to the users.

The inherent problem with replicated objects is their update. Replicas of an object denote the same entity, consequently, an update to one replica must be reflected on all other replicas. There is a trade-off between object availability and consistency. The consistency of objects cannot be compromised, as a result, complicated update operations are the price paid for availability and consistency. The complexity of update operations can be ameliorated by employing variations of replication strategies such as *read-only replication*. Although the price of storage devices has continued to decline, storage space is still a resource that must be managed. In spite of the advantages of replication, this technique may not always be appropriate for multimedia data for at least two reasons. First, replication does not adequately address the issue of varying access frequencies, and hence, different utilization, of storage devices. Second, multimedia data is enormous and replication or mirroring of such huge amount of data may be economically inappropriate.

On the other hand, dynamic object reallocation is not a perfect solution since it requires intelligent, reliable, and efficient algorithms. We need to keep and constantly update a considerable amount of meta data in order to perform cost-effective object reallocations.

Taking cognizance of the characteristics of object replication and reallocation, and recognizing the diverse properties and sizes of multimedia objects, it is obvious that object reallocation is very important. It affords us the opportunity to utilize other storage devices that are relatively under-utilized at any time without sacrificing space or redundancies.

5 Conclusions

In this paper, we have discussed and also proposed efficient strategies and techniques for the reallocation of multimedia objects. An optimal allocation of objects is important in maintaining system's performance. In the presence of non-decomposable data, the size of an object is the primary parameter for determining its allocation for optimization. In the presence of multimedia information systems and multimedia objects, due to their diverse properties and requirements, it has become necessary to account for more than one property of an object. The discussions in this paper highlight the important and necessary conditions for satisfying the objects' processing requirements and reallocation issues. The paper outlined criteria for decomposing large objects for distributed and parallel retrievability, which include such factors as size, data availability rate, parallel processing requirement, decomposition strategies, object allocatabilities, and reallocation criteria.

We have also discussed the general network implications for efficient and reliable multimedia data delivery. Issues such as the resource reservation, data retransmission and admission criteria and control are discussed.

References

- [1] D. Bitton and J. Gray. Disk Shadowing. In *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 331–338, Los Angeles, California, September 1988.
- [2] C. Chen, K. Nwosu, and P. Bruce Berra. Multimedia Object Modeling and Storage Allocation Strategies for Heterogeneous Parallel Storage Devices in Real Time Multimedia Computing Systems. In *Proceedings IEEE 17th Annual International Computer Software and Applications Conference (COMPSAC)*, pages 216–223, 1993.
- [3] B. Furht, D. Kaira, F. Kitson, A. Rodriguez, and W. Wall. Design Issues for Interactive Television Systems. *IEEE Computer*, pages 25–38, May 1995.
- [4] D. Gemmell, H. Vin, D. Kandlur, P. Rangan, and L. Rowe. Multimedia Storage Servers: A Tutorial. *IEEE Computer*, pages 40–49, May 1995.
- [5] M. Kim. Synchronized Disk Interleaving. In *IEEE Transactions on Computers, Vol. C-35, No. 11*, November 1986.
- [6] K. Nwosu. *Data Storage Modeling and Management for Multimedia Information Systems*. PhD thesis, Syracuse University, Syracuse, New York, December 1993.
- [7] C. Orji, P. Bobbie, and K. Nwosu. Decomposing Multimedia Data Objects for Parallel Storage Allocation and Retrieval. Submitted to Journal of Intelligent Information Systems (JIIS).
- [8] C. Orji, P. Bobbie, and K. Nwosu. Spatio-Temporal Effects of Multimedia Objects Storage and Delivery for Video-On-Demand Systems. To appear in Multimedia Systems Journal (MSJ).
- [9] C. Orji, K. Nwosu, and N. Rishe. Multimedia Object Storage and Retrieval. In *Proceedings of International Symposium on Multimedia Systems*, pages 368–375, Yokohama, Japan, 1996.
- [10] D. Patterson, P. Chen, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the International Conference of the ACM SIGMOD*, pages 109 – 116, Chicago, Illinois, June 1988.
- [11] R. Stevens. *UNIX Network Programming*. Prentice Hall, New Jersey, 1990.