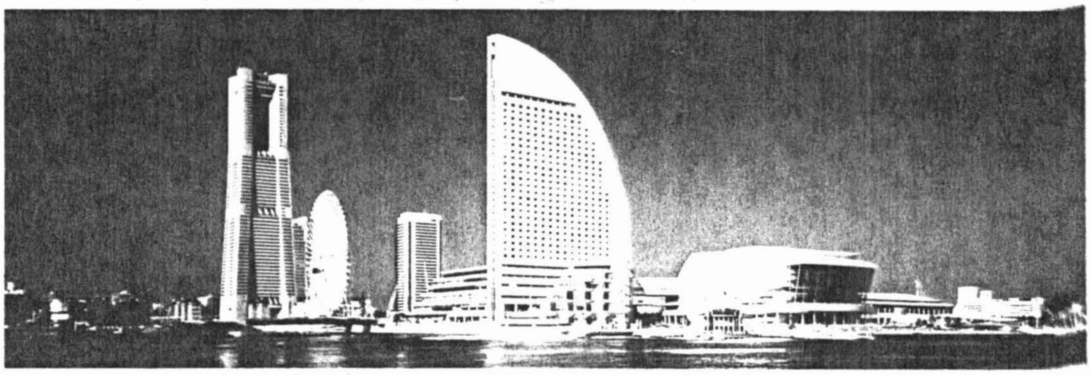
96-MO

Proceedings of the Multimedia Japan 96



March 18 - 20, 1996

Yokohama, Japan

Sponsored by Information Processing Society of Japan (IPSJ)

Multimedia Object Storage and Retrieval*

Cyril U. Orji, Naphtali Rishe
High Performance Database Research Center
School of Computer Science
Florida International University
Miami, FL. 33199
{orji,rishen}@fiu.edu

Kingley C. Nwosu
AT&T Bell Labs.
67 Whippany Road
Whippany
NJ 07981-0903
nwosuck@harpo.wh.att.com

Abstract

Object storage and retrieval is one of the challenging problems in object-based information systems. It requires the efficient and reliable satisfaction of the inherent and processing requirements of the objects. In this paper, we discuss different properties and requirements of objects which necessitate their decomposition for distribution and parallel retrievability. These include the object's size, data availability rate, and parallel processing requirements. 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. We also discuss the implications of some of these allocation techniques with respect to parity placement in RAID architecture and also their impact on certain interactive video on demand functionalities such as fast forward and rewind. We introduce Staggered RAID, a data placement scheme that provides both fault tolerance and load balancing in multimedia storage information systems.

1 Introduction

Numerous problems in the management of "objects" have emerged with the advent of multimedia information processing. Some of these include object storage, retrieval, synchronization, consistency, and transport. The storage allocation strategies for multimedia systems have attracted a lot of attention [5, 6, 10, 11]. Most of the techniques and strategies that have been proposed [3, 7, 9] are extensions to techniques used in traditional (non-multimedia) data processing environments. However, given the temporal characteristics of multimedia data, these techniques have serious implications on various functionalities of multimedia systems. Consequently, it is important that these performance implications be seriously considered in the design of algorithms for the storage and retrieval of multimedia objects.

Disk striping was studied in [13, 7] and found to be an effective technique for satisfying the high bandwidth requirements of certain supercomputing applications. However, Redundant Array of Inexpensive Disks (RAID) [12] was introduced to address the fault tolerance limitations of stripped disk arrays. In this technique, a parity disk is added to the stripped system so that fault tolerance can be maintained for a single disk failure.

Interactive video-on-demand (IVOD) applications are likely to become commonplace in the not too distant future. IVOD applications require high performance storage servers which support parallel access to data. Recent studies have shown that proper data placement and allocation techniques are central to the provision of certain IVOD functionalities [2, 8]. The data layout technique used greatly affects system performance when functionalities such as *fast forward* and *rewind* are supported.

In this paper, we discuss the placement or allocation and retrieval issues of objects in multimedia information systems. The different properties and requirements of an object which necessitate its decomposition for parallel retrievability are described. These include the object's size, 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. In Section 2 we discuss techniques for object decomposition in systems with heterogeneous and homogeneous storage devices. We develop a framework for efficient object decomposition in a heterogeneous storage device environment. In Section 3 we discuss stream control in interactive multimedia systems and discuss problems arising from certain data placement techniques. In Section 4 we introduce *Staggered RAID*, a data placement technique that provides both fault tolerance and load balance for IVOD environments. Our conclusions are presented in Section 5.

2 Storage Requirements for Objects

In this section we study the storage requirements of objects which foster their parallel retrievability. We also discuss the properties, characteristics, or processing requirements of an object which impact its storage.

2.1 Inherent properties and requirements

Characteristics that necessitate the decomposition of a large object include the size of the object, the

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

data availability rate, and the parallel processing requirements.

2.1.1 Object size

The problems associated with multimedia objects are often due to their continuous characteristics. For example video objects need to be digitized in order to store or transport them. The digitization of video signals results in objects of enormous sizes. Different encoding standards have been established to reduce the sizes of these objects. For example, the MPEG-I (Motion Picture Expert Group) encoding standard can encode 100 Mb to 1.5 Mb, while MPEG-II can encode the same original data to about 3 Mb. Therefore, given a 2 hour movie encoded with MPEG-II, a 21.6 Gb object results. Ignoring other requirements of the object, a storage device of at least the size of the object is needed to store it.

2.1.2 Data availability rates

Most of the 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. For example, if a television display requires x frames per second, and if each frame is y bits, then, the data utilization rate is $x.y$ bits/sec. It is, therefore, necessary that this amount of data be available when needed for the object to be meaningfully processed. Table 1 shows examples of some of the throughput requirements for some types of multimedia objects and standards such as compressed MPEG I & II, PAL (Phase Alternation by Line - European Television Standard), NTSC (National Television Standard) and HDTV (High Definition TV).

Data Type	Data Rate (per sec.)
Text	one page = 40 Kbits
Vector Graphics	500 lines = 24 Kbits
Bitmap Images	one page = 0.384 Mbytes
Digitized Voice	64 Kbits
Digitized Audio	640 Kbits
PAL Digitized Video	30 Mbytes
MPEG-I Video	1.5 Mbits to 3.0 Mbits
MPEG-II Video	3.0 Mbits to 6.0 Mbits
NTSC Digitized Video	25.2 MBytes
HDTV	144 MBytes

Table 1: **Examples of Multimedia Object Throughput 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.* An optimal allocation entails the minimization of seek and rotational latencies during data retrieval. To model the relationship between data availability and bandwidth of storage device, we assume that whenever necessary, each object can be optimally stored in a storage device. We let

O_i	= the ith multimedia object,
O_i^j	= the jth subobject or segment of O_i,
S_i	= the size of O_i (i.e., $\mid O_i \mid$),
S_i^j	= the size of O_i^j (i.e., $\mid O_i^j \mid$),
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 kth 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

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

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. For example, let Table 2 represent the decomposition matrix of an object, O_k, where $s_{i,j}$ (assuming row/major ordering) represents ith row and jth column, and m_{ss} represents the number of storage segments (target storage locations). Each row of O_k can be modeled as:

$$\forall j\ [j = 1 \ldots 4]\ s_{i,j} \in O_k^i, \quad i = 1, \ldots, m_{ss} \qquad (2)$$

and each column of O_k as:

$$\forall j\ [j = 1 \ldots 4]\ s_{j,i} \in O_k^i, \quad i = 1, \ldots, m_{ss} \qquad (3)$$

$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,4}$
$s_{2,1}$	$s_{2,2}$	$s_{2,3}$	$s_{2,4}$
$s_{3,1}$	$s_{3,2}$	$s_{3,3}$	$s_{3,4}$
$s_{4,1}$	$s_{4,2}$	$s_{4,3}$	$s_{4,4}$

Table 2: **An Example Decomposition for Parallel Processing**

All the segments composed from a given row or column or diagonal can be stored in a storage device attached to a different site or different storage device within a single machine. For example, it is possible to extract all the diagonal elements of the object, i.e., $s_{1,1}, s_{2,2}, s_{3,3}, s_{4,4}$, by concurrent issue of I/O requests to the applicable sites or devices.

2.2 Network implications

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

Even in the presence of efficient buffering strategy, the number of retransmissions and their temporal implications can become non-deterministic making it difficult for them to be accounted for by a buffering strategy. Consequently, network reliability becomes an important element in a multimedia delivery system.

Although studies such as that reported in [4] have demonstrated the effectiveness of statistical reservation schemes in certain multimedia environments, there is no doubt that for many multimedia applications, it is still necessary that the underlying transmission networks have the capability of guaranteed resource reservation. This implies that the bandwidth for the data transmission from source to destination must be guaranteed prior to data leaving its storage site. This may become a problem when heterogeneous networks are involved. An example is suggested in Figure 1 where the various networks $T_1, \ldots, T_4$ may be different types of networks with different protocols.

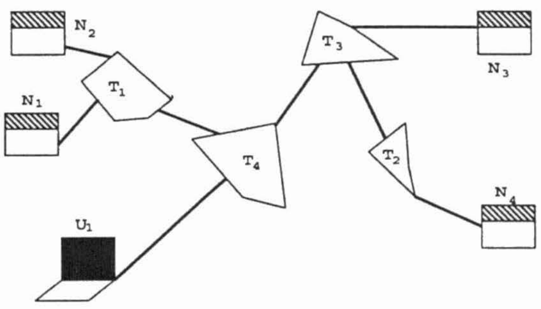


Figure 1: **An Example of a Heterogeneous Network Environment**

Figure 1 can be used to illustrate a temporal implication of multimedia delivery over heterogeneous networks. Let $N_1, \ldots, N_4$ be multimedia servers with permanent storage devices. Each N_i $(i = 1 \ldots 4)$ is connected to a network T_k $(k = 1 \ldots 4)$ as shown. U_1 is a client site where multimedia data requests are initiated.

Let $\phi_{i,j}$ be the network bandwidth between nodes i and j, and ρ_h^k the retrieval time of object O_k at N_h. We assume that each N_h achieves any multimedia object's data availability rate. Therefore, if t_{curr} is the current time, and O_1, O_2 (stored at N_2 and N_3, respectively) are required at U_1 at time $(t_{curr} + \delta)$ for synchronized presentation, then, the following temporal condition must be satisfied:

$$\begin{aligned} \mathcal{T} &= max(\rho_2^1, \rho_3^2) + max(\epsilon_2, \epsilon_3) + \\ & \quad max(\lceil \frac{Q_1}{\phi_{2,4}} \rceil, \lceil \frac{Q_2}{\phi_{3,4}} \rceil) < \delta \end{aligned} \tag{4}$$

where ϵ_2, ϵ_3 are the times for the requests to get to N_2, N_3, respectively. Note that concurrent processing at the various nodes is assumed, hence the use of the maximum time for each set of concurrent events.

2.3 Object decomposition

In this section, we present the conditions and criteria for determining the number and sizes of the segments and allocation of each data/object segment to a particular storage device.

2.3.1 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. Table 2 illustrates that 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, and (ii) degree of parallelism.

If the object cannot fit in a single storage device, the number of SUs and sizes of SU solely depend on the size of the object and the number of usable storage devices. Suppose an object, a, can be or is optimally allocated to device, b, denoted by $a \Rightarrow b$, and that there are m storage devices in a system. Given an object, O_i, we can decompose it into n object segments such that

$$\begin{aligned} &\forall j \; [j = 1 \ldots n] \; \exists \; SD_k \textit{ such that} \\ &O_i^j \Rightarrow SD_k, 1 \leq k \leq m. \end{aligned} \tag{5}$$

If the DAR of the object cannot be satisfied by a single storage device, then the *number* of SUs depends on the DAR of the object and the bandwidths of the storage devices, while the *size* of each SU depends on the bandwidth of the target storage device. The criteria are that (1) no storage device should be required to retrieve an amount of data, per unit time, that is greater than its bandwidth; and, (2) the cumulative bandwidths of the applicable storage devices must be as large as the DAR. Therefore, given O_i we can decompose it into n segments $(n \leq m)$ such that

$$\begin{aligned} &\forall j \; [j = 1 \ldots n] \; \exists \; SD_k \textit{ such that} \\ &BW(SD_k) \; \geq \; DAR_i^j \textit{ and } O_i^j \Rightarrow SD_k \end{aligned}$$

$$and \sum_{k=1}^{n} BW(SD_k) \geq DAR_i \qquad (6)$$

In a homogeneous storage device environment, we can simply determine n as

$$n = \frac{DAR_i}{BW(SD_1)} \qquad (7)$$

However, for the remainder of this section we focus on a heterogeneous (storage device) environment. In this type of environment where the storage devices may have different bandwidths, there are usually many decomposition alternatives. There is, therefore, the need to impose some constraints on permissible decompositions. A number of factors such as homogeneity of objects, the device types, and the number of each device type available may influence allowable decomposition alternatives. We describe our heterogeneous decomposition algorithm with an example.

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, $ss_k = \{y_1, y_2, \ldots\}$ is the kth 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. $[y_1 BW(SDG_1) + y_2 BW(SDG_2) + y_3 BW(SDG_3)] \geq DAR_j$ and
   (a) $[(y_1 - 1)BW(SDG_1) + y_2 BW(SDG_2) + y_3 BW(SDG_3)] < DAR_j$,
   (b) $[y_1 BW(SDG_1) + (y_2 - 1)BW(SDG_2) + y_3 BW(SDG_3)] < DAR_j$,
   (c) $[y_1 BW(SDG_1) + y_2 BW(SDG_2) + (y_3 - 1)BW(SDG_3)] < DAR_j$.
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 enshrouded in the following integer linear programming problem:

$$y_1 BW(SDG_1) + y_2 BW(SDG_2) + y_3 BW(SDG_3) \geq DAR_j,$$

$$y_1 \leq y_2,\ y_2 \leq y_3,\ y_3 > 0. \qquad (8)$$

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

For example, consider an object of size $120KB$ and bandwidth requirement of $60KB/s$. Given that $SDG_1 = \{SD_1, SD_2, SD_3\}$, $SDG_2 = \{SD_4, SD_5\}$, $SDG_3 = \{SD_6, SD_7, SD_8\}$, $BW(SDG_1) = 30KB/s$ (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 $ss_1, \ldots ss_7$. 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. In the case where an object is decomposed for parallel processing, an additional constraint is that the degree of parallelism (*dp*) must equal the number of SUs.

2.3.2 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. Although a detailed treatment of the factors that influence these choices cannot be undertaken here (see [9]), 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.

In the remainder of this paper, we return to the more common case of homogeneous storage device environment. First we briefly describe the RAID technique which has become the acceptable method of stripping data across multiple disks and provides fault tolerance for a single disk failure. Next we describe some important functionalities of interactive video-on-demand systems and show the problem presented by the RAID technique in these situations. We then suggest *Staggered RAID* as a possible solution to the problem.

2.4 Object decomposition in a homogeneous storage device environment: The RAID scheme

In a homogeneous storage device environment, data is often decomposed to facilitate parallel access. This technique is referred to as data or disk stripping [13]. In theory, the I/O bandwidth of a system can be increased N times if data is striped across N disks in the system. One of the major problems with disk stripping is that it has poor reliability since the failure of any of the disks in the array results to a loss of data since the data in the failed disk is unavailable. As a result, while it is attractive to increase the degree of parallelism, this also increases the probability of data loss.

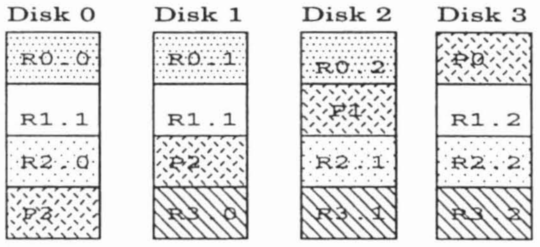


Figure 2: **RAID-5**

In a RAID system [12], redundant parity information is used to recover from a single disk failure. A number of RAID levels were defined in [12], but we use the term RAID to refer to the level 5 or rotated parity RAID. This scheme does not replicate data as in mirroring and hence has lower storage overhead, but depending on the workload it could have lower performance as well. In a RAID scheme a unit of data is ***striped*** across multiple drives and can be accessed in parallel, resulting in a significant increase in transfer rate. Figure 2 illustrates a RAID scheme with four disks. *Ri.j* is the j^{th} fragment of *Ri*. *Ri*.0, *Ri*.1, and *Ri*.2 together form a ***stripe***. *Pi* is the parity associated with *Ri*. If a disk fails, a missing sector can be reconstructed by computing the exclusive OR ("xor") of all data sectors in the stripe. Observe that the parity sectors are spread evenly over all the disks to prevent a parity disk from becoming a bottleneck during updates. A detailed treatment of RAID is beyond our scope here.

3 Stream Control in Interactive Multimedia Systems

Multimedia applications require interactivity in the form of stream playout control that allows a user to do ***fast forward (ff)***, ***rewind (rw)***, ***slow play***, ***slow rewind***, ***pause***, and ***stop-and-return*** on a media stream as is done in video cassettes. A user may also access the media streams in a random manner. These operations have implications for data layout and scheduling in an interactive video-on-demand (IVOD) environment. Two ways of implementing stream control for some of these operations are [2]:

- **Rate Variation Scheme (RVS)**: In this technique, the rate of display at the client, and hence the rate of data retrieval at the server, is changed. For *ff* the data retrieval at the server is increased. A performance study of this type of scheme is presented in [4].

- **Sequence Variation Scheme (SVS)**: In this technique, the sequence of frame display, and hence the sequence of data retrieval and transmission from the server, is varied. The display rate at the client side is unaltered.

There are three disadvantages of the RVS implementation for *ff* and *rw* [2]. These include (1) Increased network and storage bandwidth requirement since the data rate is increased; (2) Increased data handling requirement for real-time decoders which may be unable to handle the increased throughput they now face; and (3) Increased buffer requirement at the client since the arrival rate of data has increased.

Because of these drawbacks, the SVS implementation appears to be the preferred alternative. However, as noted in [2], scheduling for *ff (rw)* is problematic with the SVS approach. Consider a system with storage nodes $D = 6$ and a *ff* implementation that skips alternate frames. In normal playout, the frame sequence is $\{0, 1, 2, 3, 5, 6, \ldots\}$, whereas for the *ff* the same sequence is $\{0, 2, 4, 6, 8, 10, \ldots\}$. If data layout is simple round robin (modulo D) algorithm, then the set of nodes visited during normal playout is $\{0, 1, 2, 3, 4, 5, \ldots\}$, whereas in *ff* mode the nodes visited are $\{0, 2, 4, 0, 2, 4, \ldots\}$. There are two problems with this simple example. First, the stream control alters the sequence of node-visits from the normal linear (modulo D) sequence. Second, it creates "hot-spots" and in turn requires bandwidth to be reserved at each node to deal with the overloads.

To address this problem a number of data layout algorithms have been suggested. In [8] the prime round robin (PRR) layout policy was introduced. The PRR uses arbitrary number of disks (N) with uniform load balance for fast retrievals as well as display and slow retrievals, but the rounding distance is the biggest prime number N_p ($\leq N$) instead of N. Using their model, the j^{th} segment of the i^{th} object would be stored in disk k of N, where k is given by

$$k = \begin{cases} (\alpha + j \bmod (N - N_p + 1)) \bmod N & \text{if } j = cN_p \\ (\alpha + N - N_p + (j \bmod N_p)) \bmod N & \text{otherwise} \end{cases}$$

where $\alpha = (N - N_p + 1) \times i$.

PRR allows fast retrieval as well as play at any speed $s \neq cN$ to access N distinct disks provided N is prime. There are, however, a number of problems with the PRR algorithm. First, it is wasteful of disk space. For example, an installation that has an array of 10 disks uses only 7 of them to store a stripe (since 7 is the greatest prime number less than or equal to 10). This is a 30% underutilization of available storage and also a 30% decrease in parallel I/O transfers. Second, it complicates the parity placement algorithm and in some cases causes the parity disk(s) to become a performance bottleneck. As an example, consider a PRR placement scheme with $N = 6$ and $N_p = 5$. Although this choice minimizes space overhead (it does not incur additional space overhead besides the parity disk associated with the RAID scheme), it has some implication for parity placement. Using the PRR placement algorithm, we show in Table 3 the placement of the segments of three multimedia objects in the disk array. Table 4 shows the placement of the segments of one large multimedia object in the disk array. The parity for each stripe is indicated by **P** in the tables.

Obj#	Object Segments					
	d0	d1	d2	d3	d4	d5
0	0	**P**	1	2	3	4
	P	5	6	7	8	9
	10	**P**	11	12	13	14
	P	15	16	17	18	19
	20	**P**	21	22	23	24
	P	25	26	27	28	29
1	3	4	0	**P**	1	2
	8	9	**P**	5	6	7
	13	14	10	**P**	11	12
	18	1	**P**	15	16	17
	23	24	20	**P**	21	22
	28	29	**P**	25	26	27
2	1	3	3	4	0	**P**
	6	7	8	9	**P**	5
	11	12	13	14	10	**P**
	16	17	18	19	**P**	15
	21	22	23	24	20	. **P**
	26	27	28	29	**P**	25

Table 3: **Placement of Segments of Three Objects Across 6 Disks Using PRR Algorithm.** Each row is a stripe, and **P** is parity for a stripe.

A look at the tables will reveal the problems with PRR. Observe from Table 3 that for each object, the parity is not evenly distributed across the disk array. Note that the parity of all segments of a single object map to only two disks. For example the parity for all the segments of Object 0 map to only disks 0 and 1, the parity for Object 1 to only disks 2 and 3. The pattern starts to repeat when we get to Object 3 (not shown). In fact, in the case of a single large object, all the parity blocks map to only two disks (see Table 4). This could present serious problems if updates are allowed since the parity disks would become a bottleneck.

4 Load Balancing, Fault Tolerance and Data Layout Algorithms

A number of multimedia data layout schemes were defined in [2]. Two of these are the *Distributed Cyclic Layout (DCL)* and the *Staggered Distributed Cyclic Layout (SDCL)*. Table 5 shows an example. These are, essentially different forms of disk striping [13], and while they increase the I/O throughput of multimedia systems, they provide no fault tolerance. We note that DCL is basically analogous to the *modulo D* algorithm (D is number of disks in array), while SDCL is a slightly modified version of DCL where data placement for the next stripe starts at disk $(k+s)$ *modulo D* given that the current stripe starts at disk k. s is the stagger distance. In Table 5, $s = 1$. SDCL is also similar to *staggered striping* introduced in [1]. The starting block in each stripe is shown in bold in Table 5.

	Object Segments					
Obj#	d0	d1	d2	d3	d4	d5
	0	**P**	1	2	3	4
	P	5	6	7	8	9
	10	**P**	11	12	13	14
	P	15	16	17	18	19
	20	**P**	21	22	23	24
	P	25	26	27	28	29
	30	**P**	31	32	33	34
	P	35	36	37	38	39
0	40	**P**	41	42	43	44
	P	45	46	47	48	49
	50	**P**	51	52	53	54
	P	55	56	57	58	59
	60	**P**	61	62	63	64
	P	65	66	67	68	69
	70	**P**	71	72	73	74
	P	75	76	77	78	79
	80	**P**	81	82	83	84
	P	85	86	87	88	89

Table 4: **Placement of Segments of a Single Large Object Across 6 Disks Using PRR Algorithm.** Each row is a stripe, and **P** is parity for a stripe.

A set of D frames is said to be load-balanced if the set of nodes from which these frames are retrieved contains each of the D nodes only once [2]. Using this definition, a number of important load balancing theorems in distributed data layout were proved in [2]. We summarize the key features of the theorems as follows:

1. If the number of storage nodes is finite, no distributed (multimedia) data layout scheme will support fast forward (rewind) of arbitrary skipping distance without violating the load balance condition.

2. For a DCL over D storage nodes, if the fast forward (rewind) distance d_f is relatively prime to D, then the set of nodes S_n from which consecutive D frames in fast forward frame set S_f are retrieved is load-balanced. For example, if D = 16, DCL will produce load-balanced node sets for d_f = 3,5,7,11,13,15. These numbers are prime to 16. A corollary to this is that if D is prime and the ff (rw) distance is not a multiple of D, then the set of nodes from which D frames are retrieved is load-balanced. For example, for D = 11, DCL will produce load-balanced node sets for d_f = 2,3,4,5,6,7,8,9,10 for consecutive D frames retrieved.

3. For a SDCL over D storage nodes with stagger distance $s = 1$, load-balancing will be achieved under certain conditions. For example, if the *ff* starts at a node corresponding to a pivot frame – shown in bold in Table 5 (or 2D - 1 nodes from pivot for *rw*), and if the *ff (rw)* distance d_f is a factor of D, then the set of nodes from which D frames are retrieved is load-balanced. For example, if D = 16, SDCL will produce load-balance node sets for d_f = 2,4,8,16.

Type of	Object Segments					
Layout	d0	d1	d2	d3	d4	d5
	0	1	2	3	4	5
	6	7	8	9	10	11
	12	13	14	15	16	17
DCL	**18**	19	20	21	22	23
	24	25	26	27	28	29
	30	31	32	33	34	35
	36	37	38	39	40	41
	42	43	44	45	46	47
	0	1	2	3	4	5
	11	**6**	7	8	9	10
	16	17	**12**	13	14	15
SDCL	21	22	23	**18**	19	20
	26	27	28	29	**24**	25
	31	32	33	34	35	**30**
	36	37	38	39	40	41
	47	**42**	43	44	45	46

Table 5: **DCL and SDCL Placement Schemes**

Type of	Object Segments						
Layout	d0	d1	d2	d3	d4	d5	d6
	P	0	1	2	3	4	5
	11	**P**	**6**	7	8	9	10
	16	17	**P**	**12**	13	14	15
RAID5 +	21	22	23	**P**	**18**	19	20
SDCL	26	27	28	29	**P**	**24**	25
	31	32	33	34	35	**P**	**30**
	36	37	38	39	40	41	**P**
	P	**42**	43	44	45	46	47

Table 6: **Staggered RAID**

The problem with the DCL and SDCL schemes is that they provide no fault tolerance even to a single disk failure. In the prime round robin (PRR) algorithm it was possible to store the parity of a stripe in one of the unused disks in the group – even though as already noted, the parity data so stored was not evenly spread over all the disks as suggested in the RAID5 architecture. However, in the DCL and SDCL schemes there are no unused disks, hence a parity disk must be added to the schemes.

For the DCL algorithm, since every stripe starts from the same disk – disk 0 in Table 5, a disk dedicated entirely for parity must be added. Such a scheme is analogous to what was described as RAID level 4 in [12] and presents performance problems since the parity disk becomes a bottleneck during updates.

However, it is rather straightforward to add a parity disk to the SDCL scheme. Table 6 is an example. We refer to this scheme combining SDCL and RAID5 as

Staggered RAID. The example in Table 6 has a stagger distance $s = 1$. Note that the parity in a current stripe is placed in disk $(k+1)$ *modulo* D where the parity in the preceding stripe is in disk k and there are D disks in the array. The location of the first (pivot) block (frame) in each stripe is similarly determined.

The Staggered RAID architecture provides fault tolerance for a single disk failure. Moreover, unlike the PRR scheme where the parity blocks are likely to be unevenly placed in all the disks in the array, the parity blocks (frames) in the Staggered RAID scheme are evenly spread over the entire disks in the array. With respect to load balance, the Staggered RAID provides load balance like the underlying SDCL algorithm. However, to compute the fast forward (rewind) distances over which load balance would be maintained, the number of disks D must be taken as the total number of disks minus one (the parity disk). For example, from Table 6 if we recognize that $D = 6$, it is straightforward to verify that load-balanced node sets for retrieving D frames will be produced for $d_f = 2,3$ (the factors of 6). Assume we wish to use the SVS scheme to perform fast forward starting with frame 0. For $d_f = 2$ the first D frames retrieved would be {0,2,4,6,8,10}, and these would require access to nodes {1,3,5,2,4,6}. Similarly for $d_f = 3$, the first D frames would be {0,3,6,9,12,15} requiring access to nodes {1,4,2,5,3,6}.

5 Conclusions

In this paper, we have discussed and proposed efficient strategies and techniques for the allocation of multimedia objects in heterogeneous and homogeneous storage device environments. An optimal allocation of objects is important for maintaining multimedia system 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 objects, it has become necessary to account for the many properties of these objects. We outlined criteria for decomposing objects for distributed and parallel retrievability. We also discussed performance consequences of some data layout schemes. It was noted that a number of data layout schemes addressed the issues of performance and fault tolerance in isolation. We introduced *Staggered RAID* – a novel architecture that provides fault tolerance like the RAID5 scheme and at the same time provides load balancing features supported by the *Staggered Distributed Cyclic Layout (SDCL)* technique.

References

[1] S. Berson and S. Ghandeharizadeh. Dynamic File Allocation in Disk Arrays. In *Proceedings of the International Conference of the ACM SIGMOD*, Minneapolis, Minnesota, May 1994.

[2] M. Buddhikot and G. Parulkar. Distributed Data Layout, Scheduling and Playout Control in a Large Scale Multimedia Storage Server. Technical Report WUCS-94-33, Dept. of Computer Science, Washington University, St. Louis, MO 63130, 1994.

[3] 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.

[4] J. Dey-Sircar, J. Salehi, J. Kurose, and D. Towsley. Providing VCR Capabilities in Large-Scale Video Servers. In *Proceedings of ACM Multimedia International Conference*, pages 25–32, San Franscisco, CA, October 1994.

[5] B. Furht, D. Kaira, F. Kitson, A. Rodriguez, and W. Wall. Design Issues for Interactive Television Systems. *IEEE Computer*, pages 25–38, May 1995.

[6] D. Gemmell, H. Vin, D. Kandlur, P. Rangan, and L. Rowe. Multimedia Storage Servers: A Tutorial. *IEEE Computer*, pages 40–49, May 1995.

[7] M. Kim. Synchronized Disk Interleaving. In *IEEE Transactions on Computers, Vol. C-35, No. 11*, November 1986.

[8] T. Kwon and S. Lee. Data Placement for Continuous Media in Multimedia DBMS. In *Proceedings 1995 International Workshop on Multi-Media Database Management Systems*, 1995.

[9] K. Nwosu. *Data Storage Modeling and Management for Multimedia Information Systems.* PhD thesis, Syracuse University, Syracuse, New York, December 1993.

[10] 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).

[11] 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).

[12] 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.

[13] K. Salem and H. Garcia-Molina. Disk Striping. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 336–345, Los Angeles, California, February 1986.

Information Processing Society of Japan (IPSJ)