

# Data Layout for Interactive Video-on-Demand Storage Systems\*

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

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

## Abstract

*Interactive video-on-demand applications pose a number of interesting problems such as the support of fast forward and rewind at arbitrary speeds. While it has been shown that no single data layout scheme can support these functions at arbitrary speeds without violating load balance conditions, researchers are now exploring methods to allow at least a wide range of speeds to be supported under load balance conditions. In this paper, we introduce Declustered Mirror, a novel scheme that mirrors a set of striped disks to provide improved performance and fault tolerance for a video-on-demand system. The improved performance is obtained by the use of disk stripping to provide high I/O bandwidth for multimedia data and by the use of Distributed Cyclic Layout (DCL) and Staggered Distributed Cyclic Layout (SDCL) to provide support for a wide range of fast forward and rewind speeds under load balance conditions. Fault tolerance is provided through mirroring.*

**Keywords:** load balance, fast forward, rewind, mirroring, stripping.

---

\*This work has been supported in part by grants from NASA and NSF.

# 1 Introduction

The ability to digitize, store, retrieve, process, and transport analog information has changed the dimensions of information handling in the past few years. This is a consequence of various advancements in different parts of the computing technology. A by-product of these advances is the emergence of multimedia information processing which encompasses the integrated generation, representation, processing, storage, and dissemination of independent machine processable information. Moreover, the advent of interactive video-on-demand applications have opened up a number of research issues in data layout and placement. Due to the large sizes of some multimedia objects, the storage allocation strategies for multimedia systems have attracted a lot of attention [10, 9, 18, 16, 17]. Numerous storage strategies for large objects have been proposed and utilized, e.g., [12, 14, 7, 6, 15]. Most of these techniques are extensions to techniques used in traditional (non-multimedia) data processing environments. Moreover, most of the efforts in this direction appear to be focused on the ability to satisfy the huge I/O bandwidth requirements of these large objects. Very little if any effort has been directed to the problem of fault tolerance especially in interactive video-on-demand environments. The primary technique that has been adopted so far to satisfy this huge I/O bandwidth requirement is parallel I/O in the form of disk striping.

Disk mirroring is the replication of one logical disk in  $n$  physical disks. For simplicity we assume that  $n = 2$ . Mirroring has been implemented in many systems[21, 1]. During normal operation, read requests can be serviced by any of the two disks that contain the requested data, potentially doubling throughput; however, writes must be serviced by both disks. Thus mirroring increases I/O bandwidth and fault tolerance is provided at a 100% space overhead since every data item is replicated in a mirror disk.

Disk striping was studied in [20, 12] and found to be an effective technique for satisfying the high bandwidth requirements of certain supercomputing applications. However, the fault tolerance of the striped disks was very poor since the failure of any single disk in the array meant the loss of data. Thus while it was attractive to have data striped across a large disk array, the problem was that increase in the size of the array also meant increase in the probability of data loss.

Redundant array of inexpensive disks (RAID) [19] was introduced to address the fault tolerance requirements of striped disk arrays. In this technique, a parity disk was added to the striped system so that fault tolerance can be maintained for a single disk failure. The RAID technique is very attractive because to provide fault tolerance for a single disk failure, it uses only one parity

disk for every  $G$  disks in a disk array group unlike mirroring which uses a 100% space overhead. While this parity protection appears very efficient with respect to space overhead, it does have some implications for performance especially during updates. A detailed treatment of these various storage techniques is, however, beyond the scope of this paper.

Most of the studies addressing bandwidth issues in multimedia systems have primarily used disk striping [5, 13] without considering the fault tolerance issues associated with disk striping. Although the argument could be made that the addition of a parity disk to a striped array is somewhat straightforward, we argue that the performance consequences of such a scheme especially in an interactive video-on-demand environment could be substantial. In this paper, we make the case for a *Declustered Mirror*, a scheme that mirrors a set of striped disks to provide improved performance and fault tolerance for a video-on-demand system. The advantages of this technique are as follows:

- It supports the high bandwidth requirements of multimedia systems.
- It provides fault tolerance to a single disk failure using mirroring technique.
- During normal mode operation, it provides load balance for a wide range of *fast forward (rewind)* speeds.
- During degraded mode operation (when a single disk has failed), it provides load balance by spreading the load of the failed disk over many disks in the array while still supporting a broad range of fast forward (rewind) speeds.

The primary disadvantage of this scheme is the 100% space overhead of disk mirroring.

The remainder of this paper is organized as follows. In Section 2 we describe disk mirroring and another form of disk mirroring referred to as *chained declustering* [11] which motivated our *Declustered Mirror*. In Section 3 we discuss two disk striping techniques *Distributed Cyclic Layout (DCL)* and the *Staggered Distributed Cyclic Layout (SDCL)* [5] which also partly motivated our ideas. We present the *Declustered Mirror* in Section 4 and conclude the paper in Section 5.

## 2 Mirroring Techniques

In this section we briefly overview traditional mirroring and another form of mirroring referred to as chained declustering. But first we discuss two levels of data replication that are common in multimedia storage systems. These are Intra Node and Inter Node Replication.

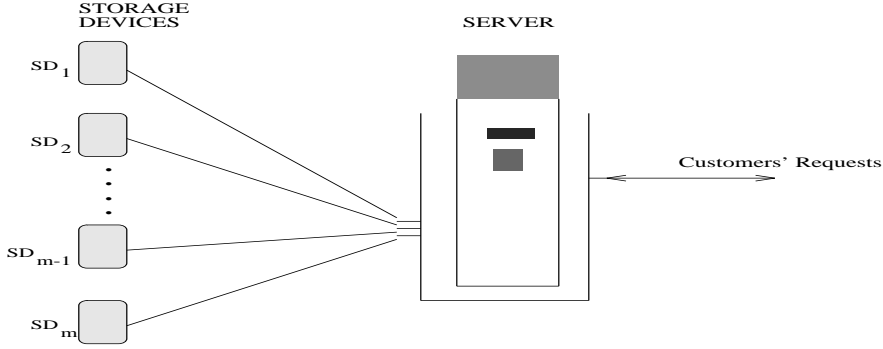
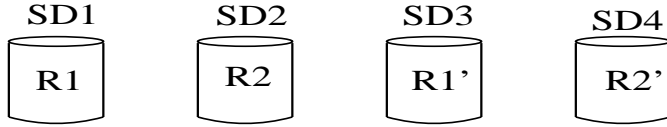


Figure 1: A Single Machine Video Server

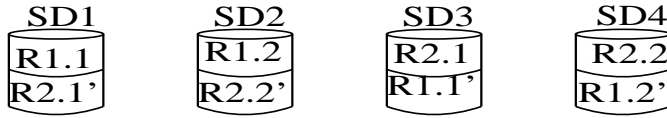
## 2.1 Intra Node and Inter Node Replication

Intra Node Replication (INRra) is replication at the storage device level per node (machine), while Inter Node Replication (INRer) is replication at processing nodes level. Given the architecture in Figure 1, the only possible replication is INRra where different copies of a large object (e.g., video file) or segments of a large object are stored in different storage devices. Given a set of storage devices, we can maintain two forms of replication aggregations, namely, Storage Device Exclusivity (SDE) or Storage Device Inclusivity (SDI). A replication strategy in INRra uses SDE when there is a set of storage devices that contain only replicas of objects. A replication strategy in INRra uses SDI when every storage device can contain both primary and secondary copies of objects. For example, using Figure 1, if  $\{SD_1, SD_2, \dots, SD_{\frac{m}{2}}\}$  are used for the primary copies of video objects and  $\{SD_{\frac{m}{2}+1}, \dots, SD_m\}$  are used for the secondary copies, then we maintain SDE technique. When a given storage device becomes unavailable, then the requests to objects stored in it are directed to the replicas.

Figure 2 uses four storage devices in a node (INRra) to illustrate SDE and SDI. In Figure 2 (a), SD1 and SD2 hold the primary copies of the video data R1 and R2. The secondary copies R1' and R2' are maintained in storage devices SD3 and SD4. This is SDE since certain storage devices hold only secondary copies of video objects. Figure 2 (b) shows the case where each storage device holds both primary and secondary copies. Note that in this example, R1 is partitioned into two such that  $R1 = R1.1 \cup R1.2$ . R2 is also similarly partitioned. While the primary fragments of R1 are held in SD1 and SD2, the secondary fragments are held in SD3 and SD4. Similarly, the primary and



(a) INRra / SDE



(b) INRra / SDI

Figure 2: Intra Node Replication Maintaining SDE and SDI

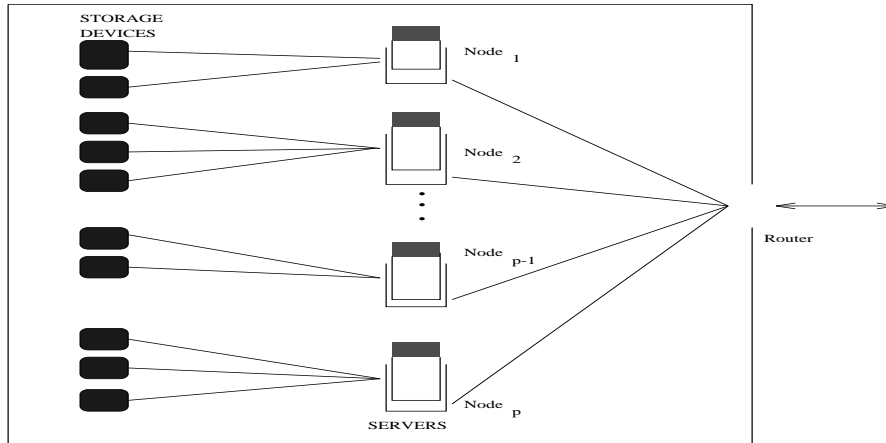


Figure 3: Multiple Video Servers

secondary fragments of R2 are held in SD3, SD4 and SD1, SD2 respectively. Since every device contains both primary and secondary copies of objects, this illustrates SDI.

Similar replication concepts are applied to architectures like the one shown in Figure 3 where objects are replicated for availability among the server nodes. In this case, a Storage Node Exclusivity (SNE) is maintained when an object is replicated on different nodes and a Storage Node Inclusivity (SNI) is maintained when an object is replicated within a single node. If availability and reliability are of paramount concern, then SNI, in this case, is not recommended. A combination of SDE (at the storage device level) and SNE (at the nodes) level provides adequate availability and reliability for storage device or node failures. In the case of node failures, it is imperative that

the system be able to dynamically re-distribute and re-direct its requests. For the remainder of the paper, we focus on Intra Node Replication.

## 2.2 Traditional Mirroring

Traditional mirroring maintains a logical disk image on two physical disks. The two physical disks are exact mirrors of one another. Figure 4 shows an example of traditional mirroring. Four disks are shown; disk 0 and disk 1 contain identical data and hence form a mirror set; similarly disk 2 and disk 3 form another mirror set.

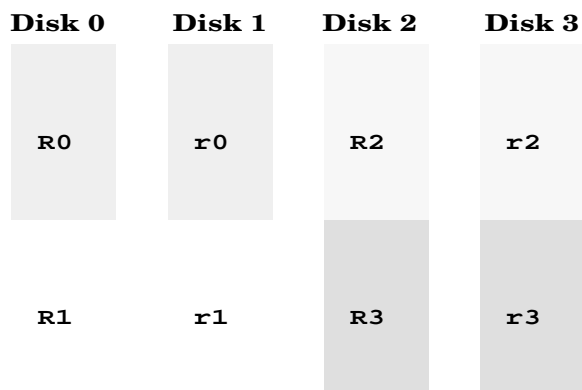


Figure 4: **Traditional Mirroring.** *Disk0 and Disk1 form a mirror set while Disk2 and Disk3 form another set.*

Traditional mirroring improves reliability [1] and performance [3, 4] over a single disk system. Reliability is improved by replicating each component in the I/O subsystem. This allows continuous operation of the I/O subsystem even in the event of a single component failure. During normal operation, read requests can be serviced by any of the two disks that contain the requested data. As a result, the system throughput could be potentially doubled. In addition, a read request is efficiently serviced using a nearest free arm scheduling algorithm. For a disk with  $n$  cylinders, traditional mirroring reduces the average seek distance of a read request to about  $n/5$  cylinders [3]. This is in contrast to the average seek distance of about  $n/3$  cylinders using a single disk. Intuitively, the 2 disk arms divide the total disk band into two regions. Consequently, for multimedia systems, a potentially wide range of scheduling scenarios are admissible to ensure that real time constraints are satisfied.

When one of the disks in a mirror set fails, all requests are serviced by the surviving disk until the failed disk is reconstructed on another drive. For example, if disk 0 fails, disk 1 picks

up all the requests meant for disk 0 until disk 0 is repaired or reconstructed. Disk reconstruction usually involves copying the survivor disk to a replacement disk. Note that regardless of the load on disks 2 and 3 they cannot participate in sharing the load of the failed disk (disk 0). This might place a lot of burden on disk 1 potentially unbalancing the load on the disks during this period. This is the motivation behind Chained Declustering.

### 2.3 Chained Declustering

As in traditional mirroring, with chained declustering [11], two physical copies (a primary and a backup) of each relation are declustered over a set of disks such that the primary and backup copies of a fragment are always placed on different disks. The disks are divided into disjoint groups called *relation clusters* and tuples of each relation are declustered among the disks of the cluster. To simplify our discussion, we assume without loss of generality the existence of one relation cluster as shown in Figure 5.

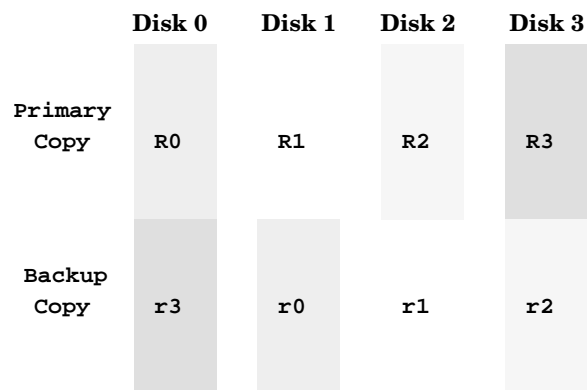


Figure 5: Chained Declustering. Note the chained formation of mirrored sets.

The data placement algorithm for chained declustering operates as follows. Assume there are a total of  $D$  disks numbered  $0, \dots, D - 1$ . We partition the relation fragments in the primary segment as shown in Figure 5. Any partitioning and placement algorithm could be used. In Figure 5 a simple modulo  $D$  algorithm which places fragment  $i$  in disk  $(i \text{ modulo } D)$  has been used. The backup fragments are placed using a modified form of this algorithm. In the backup region, fragment  $i$  is placed in disk  $((i + 1) \text{ modulo } D)$ . Observe the chained formation of mirror sets. Unlike the traditional mirroring technique where two disks are tightly coupled in a mirrored relationship, the chained declustering is more flexible.

The benefit of this scheme becomes obvious when a disk failure occurs. An example is shown

|         | disk 0 | disk 1           | disk 2           | disk 3           |
|---------|--------|------------------|------------------|------------------|
| Primary | –      | $\frac{1}{3}R_1$ | $\frac{2}{3}R_2$ | $R_3$            |
| Backup  | –      | $r_0$            | $\frac{2}{3}r_1$ | $\frac{1}{3}r_2$ |

Table 1: **Fragment Utilization with Chained Declustering After Failure of Disk 0**

in Table 1 where disk 0 is assumed to have failed. Observe how the load is evenly spread over the three remaining disks in the cluster providing better load balance than in the traditional mirror. Disk 1 services all requests for fragment 0 using the backup copy it holds ( $r_0$ ), and one-third of requests for fragment 1 ( $\frac{1}{3}R_1$ ). The requests serviced by disks 2 and 3 can be similarly deduced from Table 1. Note that the load increase is uniform across all operational disks (33%).

### 3 Stream Control in Interactive Video-on-Demand Systems

Users of high performance multimedia servers will soon have access to exciting applications such as multimedia mail, high quality interactive video-on-demand, browsing remote multimedia archives and virtual reality environments. A number of challenges have emerged from these exciting applications. One of them is the need to provide many of the functionalities currently supported in video cassettes. In this section, we take a closer look at some of these functionalities and some data layout schemes designed to support them.

#### 3.1 Interactive Video-on-Demand Functionalities

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 [5]:

- **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 [8].

- **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  $rw$  [5]. 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 [5], 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  $\{0, 1, 2, 3, 5, 6, \dots\}$ , whereas for the  $ff$  the same sequence is  $\{0, 2, 4, 6, 8, 10, \dots\}$ . If data layout is simple round robin (modulo  $D$ ) algorithm, then the set of nodes visited during normal playout is  $\{0, 1, 2, 3, 4, 5, \dots\}$ , whereas in  $ff$  mode the nodes visited are  $\{0, 2, 4, 0, 2, 4, \dots\}$ . 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.

### 3.2 Data Layout for Interactive Video-on-Demand Systems

A number of data layout schemes have been introduced to address the load balancing problem identified above. In [13] the prime round robin (PRR) layout policy was introduced. The PRR uses arbitrary number of disks ( $N$ ) with uniform load balancing 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

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

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  $N = 10$  disks uses only  $N_p = 7$  of

| Object # | Object Segments |        |        |        |        |        |
|----------|-----------------|--------|--------|--------|--------|--------|
|          | disk 0          | disk 1 | disk 2 | disk 3 | disk 4 | disk 5 |
| 0        | X0.0            | –      | X0.1   | X0.2   | X0.3   | X0.4   |
|          | –               | X0.5   | X0.6   | X0.7   | X0.8   | X0.9   |
|          | X0.10           | –      | X0.11  | X0.12  | X0.13  | X0.14  |
|          | –               | X0.15  | X0.16  | X0.17  | X0.18  | X0.19  |
|          | X0.20           | –      | X0.21  | X0.22  | X0.23  | X0.24  |
|          | –               | X0.25  | X0.26  | X0.27  | X0.28  | X0.29  |
| 1        | X1.3            | X1.4   | X1.0   | –      | X1.1   | X1.2   |
|          | X1.8            | X1.9   | –      | X1.5   | X1.6   | X1.7   |
|          | X1.13           | X1.14  | X1.10  | –      | X1.11  | X1.12  |
|          | X1.18           | X1.1   | –      | X1.15  | X1.16  | X1.17  |
|          | X1.23           | X1.24  | X1.20  | –      | X1.21  | X1.22  |
|          | X1.28           | X1.29  | –      | X1.25  | X1.26  | X1.27  |
| 2        | X2.1            | X2.3   | X2.3   | X2.4   | X2.0   | –      |
|          | X2.6            | X2.7   | X2.8   | X2.9   | –      | X2.5   |
|          | X2.11           | X2.12  | X2.13  | X2.14  | X2.10  | –      |
|          | X2.16           | X2.17  | X2.18  | X2.19  | –      | X2.15  |
|          | X2.21           | X2.22  | X2.23  | X2.24  | X2.20  | –      |
|          | X2.26           | X2.27  | X2.28  | X2.29  | –      | X2.25  |
| 3        | X3.0            | –      | X3.1   | X3.2   | X3.3   | X3.4   |
|          | –               | X3.5   | X3.6   | X3.7   | X3.8   | X3.9   |
|          | X3.10           | –      | X3.11  | X3.12  | X3.13  | X3.14  |
|          | –               | X3.15  | X3.16  | X3.17  | X3.18  | X3.19  |
|          | X3.20           | –      | X3.21  | X3.22  | X3.23  | X3.24  |
|          | –               | X3.25  | X3.26  | X3.27  | X3.28  | X3.29  |

Table 2: Placement of Segments of Three Objects Across 6 Disks Using PRR Algorithm.

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, if parity is added for fault tolerance [19], PRR would complicate the parity placement algorithm and in some cases cause the parity disk(s) to become a performance bottleneck. As an example, consider a PRR placement scheme with  $N = 6$  and  $N_p = 5$ . Using the PRR placement algorithm given by Equation 1, we show in Table 2 the placement of the segments of three multimedia objects in the disk array. Observe that if the parity for each stripe is placed in the unused disk in each stripe (a logical thing to do), the parity would not be evenly distributed over the disks. In fact, if we considered the three small objects shown in Table 2 as a single large object, all the parity would map to disks 0 and 1.

A number of multimedia data layout schemes were defined in [5]. Two of these are the *Dis-*

| Type of Layout | Object Segments |              |              |              |              |              |
|----------------|-----------------|--------------|--------------|--------------|--------------|--------------|
|                | disk 0          | disk 1       | disk 2       | disk 3       | disk 4       | disk 5       |
| DCL            | <b>X0.0</b>     | X0.1         | X0.2         | X0.3         | X0.4         | X0.5         |
|                | <b>X0.6</b>     | X0.7         | X0.8         | X0.9         | X0.10        | X0.11        |
|                | <b>X0.12</b>    | X0.13        | X0.14        | X0.15        | X0.16        | X0.17        |
|                | <b>X0.18</b>    | X0.19        | X0.20        | X0.21        | X0.22        | X0.23        |
|                | <b>X0.24</b>    | X0.25        | X0.26        | X0.27        | X0.28        | X0.29        |
|                | <b>X0.30</b>    | X0.31        | X0.32        | X0.33        | X0.34        | X0.35        |
|                | <b>X0.36</b>    | X0.37        | X0.38        | X0.39        | X0.40        | X0.41        |
|                | <b>X0.42</b>    | X0.43        | X0.44        | X0.45        | X0.46        | X0.47        |
| SDCL           | <b>X0.0</b>     | X0.1         | X0.2         | X0.3         | X0.4         | X0.5         |
|                | X0.11           | <b>X0.6</b>  | X0.7         | X0.8         | X0.9         | X0.10        |
|                | X0.16           | X0.17        | <b>X0.12</b> | X0.13        | X0.14        | X0.15        |
|                | X0.21           | X0.22        | X0.23        | <b>X0.18</b> | X0.19        | X0.20        |
|                | X0.26           | X0.27        | X0.28        | X0.29        | <b>X0.24</b> | X0.25        |
|                | X0.31           | X0.32        | X0.33        | X0.34        | X0.35        | <b>X0.30</b> |
|                | <b>X0.36</b>    | X0.37        | X0.38        | X0.39        | X0.40        | X0.41        |
|                | X0.47           | <b>X0.42</b> | X0.43        | X0.44        | X0.45        | X0.46        |

Table 3: **DCL** and **SDCL** Placement Schemes

*tributed Cyclic Layout (DCL)* and the *Staggered Distributed Cyclic Layout (SDCL)*. Table 3 shows an example. These are essentially forms of disk striping [20], 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  $(k + s) \text{ modulo } D$  given that the current stripe starts at disk  $k$ .  $s$  is the stagger distance. In Table 3,  $s = 1$ . SDCL is also similar to *staggered striping* introduced in [2] and chained declustering [11].

The periodic nature of multimedia data makes it a good candidate for striping. If we consider a frame as a logical unit of repetition, we can assume that each logical unit is physical distributed on different storage devices and accessed in parallel. Reconsider the DCL placement scheme illustrated in Table 3, we note that each object segment of object **X0** can be a frame. The first frame in each group of frames that can be accessed in parallel is referred to as **anchor** frame [5] or **pivot** frame. Note that the location of anchor node is fixed in the DCL scheme. From Table 3, this is disk 0. The anchor frames are shown in bold in the table.

As already indicated in Section 3.1, there are some problems with the DCL scheme. Consider the video frames stored in  $D = 6$  nodes using the DCL algorithm as shown in Table 3. In normal

playout, the frame sequence is X0.0, X0.1, X0.2, X0.3, X0.5, X0.6, . . . , whereas for *ff* the same sequence is X0.0, X0.2, X0.4, X0.6, X0.8, X0.10, . . . . Since data layout is with DCL algorithm, the set of nodes visited during normal playout is 0,1,2,3,4,5, . . . , whereas in *ff* mode the nodes visited are 0,2,4,0,2,4, . . . . The problem with this is the stream control alters the sequence of node-visits from the normal linear (modulo D) sequence. Moreover, it creates “hot-spots” and in turn requires bandwidth to be reserved at each node to deal with the overloads.

Note that if the SDCL scheme suggested in Table 3 is used, there would not be the hot-spots problems observed with the DCL scheme. For example, reconsider the *ff* sequence X0.0, X0.2, X0.4, X0.6, X0.8, X0.10, . . . . From Table 3 we observe that with SDCL the nodes visited are 0,2,4,1,3,5.

### 3.3 Load Balancing

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 [5]. Using this definition, a number of important load balancing theorems in distributed data layout were proved in [5]. 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 balancing 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-balance 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-balance node sets for  $d_f = 2,3,4,5,6,7,8,9,10$  for consecutive  $D$  frames retrieved. (See also PRR in Section 3.2).
3. For a SDCL over  $D$  storage nodes with stagger distance  $s = 1$ , load-balance will be achieved under certain conditions. For example, if the *ff* starts at a node corresponding to an anchor frame – shown in bold in Table 3 (or  $2D - 1$  nodes from anchor node 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-balanced node sets for  $d_f = 2, 4, 8, 16$ .

## 4 Declustered Mirror

Two issues of primary interest in this paper are:

1. The support of a wide range of fast forward and rewind speeds under a balanced load condition.
2. The support of high I/O bandwidth under fault tolerant condition.

Some ideas have been proffered for each of these problems. However, none of the solutions presents an integrated framework for addressing the two issues jointly. *Declustered mirror* is a novel scheme that efficiently addresses these problems within an integrated data layout strategy. Declustered mirror uses:

- *Disk stripping* to provide high I/O bandwidth for multimedia data.
- *Distributed Cyclic Layout (DCL)* and *Staggered Distributed Cyclic Layout (SDCL)* to provide a data layout scheme which supports a wide range of fast forward and rewind speeds under load balance condition for interactive video-on-demand systems.
- *Mirroring* to provide fault tolerance against a single disk failure.
- *Chained Declustering* to provide load balance during failure mode operation.

Table 4 shows an example of data layout in a declustered mirror scheme. There are  $N = 12$  disks, numbered  $0, 1, \dots, 11$ . These disks are divided into two clusters as suggested in chained declustering [11]. Cluster 0 consists of disks  $0, 1, \dots, 5$  while cluster 1 consists of disks  $6, 7, \dots, 11$ . The clusters form a mirror pair in a manner almost analogous to chained declustering. All disks in a cluster form one logical disk, with the logical disk of cluster 0 holding the primary copy of data and the backup copy held in the logical disk that constitutes cluster 1. To simply exposition, we assume one large object  $\mathbf{X}$  with sixty fragments  $X.0, X.1, \dots, X.59$  placed in the disks as shown in Table 4. Data in the primary logical disk is allocated using the DCL algorithm, while SDCL is used in the backup logical disk. Note that mirroring protection refers to a logical disk. Fore example all the disks in cluster 0 form logical disk 0 while all the disks in cluster 1 form logical disk 1.

| Cluster | Disk | Object Segments |            |             |             |             |             |             |             |             |             |
|---------|------|-----------------|------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 0       | 0    | <b>X.0</b>      | <b>X.6</b> | <b>X.12</b> | <b>X.18</b> | <b>X.24</b> | <b>X.30</b> | <b>X.36</b> | <b>X.42</b> | <b>X.48</b> | <b>X.54</b> |
|         | 1    | X.1             | X.7        | X.13        | X.19        | X.25        | X.31        | X.37        | X.43        | X.49        | X.55        |
|         | 2    | X.2             | X.8        | X.14        | X.20        | X.26        | X.32        | X.38        | X.44        | X.50        | X.56        |
|         | 3    | X.3             | X.9        | X.15        | X.21        | X.27        | X.33        | X.39        | X.45        | X.51        | X.57        |
|         | 4    | X.4             | X.10       | X.16        | X.22        | X.28        | X.34        | X.40        | X.46        | X.52        | X.58        |
|         | 5    | X.5             | X.11       | X.17        | X.23        | X.29        | X.35        | X.41        | X.47        | X.53        | X.59        |
| 1       | 6    | <b>X.0</b>      | X.11       | X.16        | X.21        | X.26        | X.31        | <b>X.36</b> | X.47        | X.52        | X.57        |
|         | 7    | X.1             | <b>X.6</b> | X.17        | X.22        | X.27        | X.32        | X.37        | <b>X.42</b> | X.53        | X.58        |
|         | 8    | X.2             | X.7        | <b>X.12</b> | X.23        | X.28        | X.33        | X.38        | X.43        | <b>X.48</b> | X.59        |
|         | 9    | X.3             | X.8        | X.13        | <b>X.18</b> | X.29        | X.34        | X.39        | X.44        | X.49        | <b>X.54</b> |
|         | 10   | X.4             | X.9        | X.14        | X.19        | <b>X.24</b> | X.35        | X.40        | X.45        | X.50        | X.55        |
|         | 11   | X.5             | X.10       | X.15        | X.20        | X.25        | <b>X.30</b> | X.41        | X.46        | X.51        | X.56        |

Table 4: **Declustered Mirror with Cluster Size = 6**. Cluster 0 or logical disk 0 holds the **Primary** copy allocated using DCL algorithm. Cluster 1 or logical disk 1 holds the **Backup** copy allocated using SDCL algorithm.

#### 4.1 Address Mapping

We identify two address schemes in Table 4.

1. **The address scheme of a logical disk (primary or backup)**. For our example in Table 4, in cluster 0 (the primary copy cluster, or logical disk 0), the segment or fragment number is identical to the logical disk address. In other words, the logical addresses are assigned through all the disks in the cluster in a round robin manner. While the same round robin assignment of addresses is used in the SDCL scheme (logical disk 1), it is important to note that it does not necessarily correspond to the segment or fragment number placed at the address. For example, note that in cluster 1, fragment 11 (X.11) is placed in logical address 6 of logical disk 1 (backup cluster).
2. **The physical addresses in each disk of a logical disk**. In Table 4, each row is equivalent to a disk, and ten physical addresses are shown for each disk. For example, Table 5 shows the mapping between the object fragments in disk 0 and the physical disk addresses in which they are placed.
  - A **cluster** is a group of disk drives that are accessed concurrently to retrieve a subobject X.
  - A **fragment (segment)** is the unit of data transferred from a single disk drive.

- A **stripe** consists of data in the same physical location in the disks in a cluster. For example, in Table 4, multimedia data X.0,X.1,X.2,X.3,X.4,X.5 in disks 0,1,2,3,4,5 respectively, form a stripe and are expected to be accessed in parallel.
- The anchor frame for each stripe is shown in bold in Table 4.

|                  |     |     |      |      |      |      |      |      |      |      |
|------------------|-----|-----|------|------|------|------|------|------|------|------|
| Object Segment   | X.0 | X.6 | X.12 | X.18 | X.24 | X.30 | X.36 | X.42 | X.48 | X.54 |
| Physical Address | 0   | 1   | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    |

Table 5: **Physical Mapping of Segments of Disk 0**

During normal operation, playback requests can be serviced either from the primary or backup copy. Since application programs make requests using the primary address scheme, the primary addresses must be mapped to backup addresses if the backup copy of data must be used. The mapping scheme will vary from one installation to another depending on the logical partitioning of the disk between primary and backup segments. For illustration, we present a scheme which assumes the logical partitioning suggested in Table 4.

The mapping problem can be generalized as follows: Given a DCL allocation scheme, determine a scheme that maps a given logical DCL address to corresponding logical and physical SDCL addresses, assuming that the SDCL scheme uses a stagger distance  $s = 1$ .

The algorithm is straightforward.

- Input:
  - Logical DCL address,  $DC L_{logical}(A)$ , and,
  - Number of disks  $D$  in the cluster.
- Output:
  - Logical SDCL address,  $SDCL_{logical}(A)$ ,
  - Disk number  $d$  in SDCL scheme that holds the data item, and
  - Corresponding physical SDCL address,  $SDCL_{physical}(A)$  in disk  $d$ .

The mapping is given by:

$$SDCL_{logical}(A) = ((P + T + F) \text{ mod } D) + P$$

$$\begin{aligned}
\text{Disk number } d &= ((T + F) \bmod D) + D \\
SDCL_{\text{physical}}(A) &= \frac{DCL_{\text{logical}}(A)}{D}
\end{aligned} \tag{2}$$

where

- $P = SDCL_{\text{pivot}} = \left(\frac{DCL_{\text{logical}}(A)}{D}\right) \times D$
- $T = SDCL_{\text{pivot offset}} = \left(\frac{DCL_{\text{logical}}(A)}{D}\right) \bmod D$
- $F = \text{Fragment offset from } SDCL_{\text{pivot}} = DCL_{\text{logical}}(A) - P$

We note that the disk number and physical address for the DCL scheme are also straightforward.

For the DCL scheme

$$\begin{aligned}
\text{Disk number } d &= DCL_{\text{logical}}(A) \bmod D \\
DCL_{\text{physical}}(A) &= \frac{DCL_{\text{logical}}(A)}{D}
\end{aligned} \tag{3}$$

As an example to illustrate the mapping, consider object fragment number 27 (X.27) in the primary logical disk 0 (Table 4). Since this object is also in logical address 27 in the primary logical disk, and given that  $D = 6$  in our example, using Equation 3, the object is in disk number  $27 \bmod 6$  which is 3. The physical address of this fragment in disk 3 is  $27/6$  which is 4. These numbers can be easily verified from cluster 0 in Table 4.

To map this to the backup cluster, from Equation 2

$$SDCL_{\text{logical}}(27) = ((P + T + F) \bmod 6) + P$$

where

- $P = SDCL_{\text{pivot}} = \left(\frac{DCL_{\text{logical}}(A)}{D}\right) \times D = (27/6) \times 6 = 24.$
- $T = SDCL_{\text{pivot offset}} = \left(\frac{DCL_{\text{logical}}(A)}{D}\right) \bmod D = (27/6) \bmod 6 = 4.$
- $F = \text{Fragment offset from } SDCL_{\text{pivot}} = DCL_{\text{logical}}(A) - P = 27 - 24 = 3.$

Thus  $SDCL_{\text{logical}}(27) = ((24 + 4 + 3) \bmod 6) + 24 = 25$ . This means that in the backup cluster (cluster 1 in our example), the object is stored in logical address 25 of the logical disk 1. To complete the example

- Disk number  $d = ((T + F) \bmod D) + D = ((4 + 3) \bmod 6) + 6 = \text{disk number } 7$ .
- $SDCL_{physical}(A) = \frac{SDCL_{logical}(25)}{6} = (25/6) = 4$ .

It is straightforward to verify from Table 4 that the fragment is indeed backed up at physical address 4 of disk number 7.

Note then that given any logical request, the backup address can be speedily identified. Thus during normal operation, the system can efficiently determine whether the request is best serviced from the primary or backup location. Moreover, it is clear that since every data item is mirrored, fault tolerance is provided for a single disk failure. Note also that like chained declustering, there is no tight coupling between any two physical disks, so that should a disk fail, the data in that disk is spread over multiple disks in the other cluster. This provides a natural load balancing mechanism in the event of a failure.

Since the data is stored using both DCL and SDCL techniques, the advantages of both can be utilized in providing a wide range of fast forward (rewind) speeds. For our specific example with  $D = 6$ , we note that DCL will provide load balance for fast forward speeds that are relatively prime to 6, i.e. for  $d_f = 5$ . However, the SDCL scheme will provide load balance for  $d_f = 2,3,6$  – factors of 6. Thus fast forward (rewind) speeds  $d_f = 2,3,5,6$  can be supported. For a carefully chosen number of disks, a wide range of speeds can be supported. Assume there are 32 disks, so that  $D = 16$ , then DCL can support  $d_f = 3,5,7,9,11,13,15$ , while SDCL can support  $d_f = 2,4,8,16$  for a total of eleven fast forward and rewind speeds.

## 4.2 Effect of Cluster Size on Performance

The cluster size has a major impact on the performance of declustered mirror. Our initial observations suggest that this scheme may not be appropriate for installations with a very large number of disks. There are two reasons for this. The first is the large cost associated with duplicating a large multimedia storage system. The second is intrinsic in the cluster size. Beyond a certain number of disks, there is practically no benefit in mirroring the data for performance. Recall that for our purposes here, performance is defined in terms of the number of variable speeds supported under balanced-load condition.

In Section 3.3, it was noted that the DCL algorithm provides load-balance for speeds that are prime to the number of disks (nodes)  $D$ , while SDCL provides load-balance for speeds that are factors of  $D$ . What is observed is that as  $D$  increases, the numbers that are prime to  $D$  increase

faster than the numbers that are factors of  $D$ . In fact, the factors of  $D$  fluctuate a lot and in the particular cases when  $D$  is prime, no factors are obtained with the result that the SDCL scheme does not contribute to the number of variable speeds supported except at speed =  $D$ , the cluster size.

| No. of Disks | Fraction of All Speeds Supported |      | No. of Disks | Fraction of All Speeds Supported |      |
|--------------|----------------------------------|------|--------------|----------------------------------|------|
|              | DCL                              | SDCL |              | DCL                              | SDCL |
| 2            | 0.00                             | 0.50 | 30           | 0.73                             | 0.23 |
| 3            | 0.33                             | 0.33 | 40           | 0.80                             | 0.17 |
| 4            | 0.25                             | 0.50 | 50           | 0.88                             | 0.10 |
| 5            | 0.60                             | 0.20 | 60           | 0.80                             | 0.18 |
| 6            | 0.33                             | 0.50 | 70           | 0.89                             | 0.10 |
| 7            | 0.71                             | 0.14 | 80           | 0.88                             | 0.11 |
| 8            | 0.50                             | 0.38 | 90           | 0.87                             | 0.12 |
| 9            | 0.67                             | 0.22 | 100          | 0.91                             | 0.08 |
| 10           | 0.60                             | 0.30 | 110          | 0.93                             | 0.06 |
| 11           | 0.82                             | 0.09 | 120          | 0.87                             | 0.12 |
| 12           | 0.50                             | 0.42 | 130          | 0.94                             | 0.05 |
| 13           | 0.85                             | 0.08 | 140          | 0.91                             | 0.08 |
| 14           | 0.71                             | 0.21 | 150          | 0.92                             | 0.07 |
| 15           | 0.73                             | 0.20 | 160          | 0.93                             | 0.07 |
| 16           | 0.69                             | 0.25 | 170          | 0.95                             | 0.04 |
| 17           | 0.88                             | 0.06 | 180          | 0.90                             | 0.09 |
| 18           | 0.67                             | 0.28 | 190          | 0.96                             | 0.04 |
| 19           | 0.89                             | 0.05 | 194          | 0.98                             | 0.02 |
| 20           | 0.70                             | 0.25 | 195          | 0.96                             | 0.04 |
| 21           | 0.81                             | 0.14 | 196          | 0.95                             | 0.04 |
| 22           | 0.82                             | 0.14 | 197          | 0.99                             | 0.01 |
| 23           | 0.91                             | 0.04 | 198          | 0.94                             | 0.06 |
| 24           | 0.67                             | 0.29 | 200          | 0.94                             | 0.05 |

Table 6: Fraction of All Possible Fast Forward and Rewind Speeds Supported by DCL and SDCL Algorithms for Various Cluster Sizes.

Table 6 shows the fraction of all possible speeds supported by each of the schemes for a given cluster size. For example, for ten disks (cluster size = 10), DCL will support fast forward (rewind) speeds,  $d_f = 3,4,6,7,8,9$ , that is a total of six speeds, while SDCL will support three speeds,  $d_f = 2,5,10$ . Hence the corresponding values 0.60 and 0.30 in the table for ten disks.

Another way to look at the data in Table 6 is to consider the additional speeds supported by SDCL as the benefit of mirroring data. For example, we assume that all multimedia data can be stored using DCL organization, and the only reason to buy additional disks would be to mirror

data to support additional speeds. It would then depend on individual organizations to determine whether the additional speeds supported by SDCL justifies the cost of mirroring the data. We can only note that in general, this choice tends to become less attractive as the cluster size increases.

## 5 Conclusion

The ability to digitize, store, retrieve, process, and transport analog information has changed the dimensions of information handling in the past few years. Moreover, the advent of interactive video-on-demand applications have opened up a number of research issues in data layout and placement. Most of the efforts in addressing these problems appear to be focused exclusively on the ability to satisfy the huge I/O bandwidth requirements of these large objects. Very little if any effort has been directed to the problem of fault tolerance. Even when these issues are addressed they appear to be addressed in isolation and not in association. In this paper we have viewed fault tolerance and performance as associated issues and presented an integrated framework for addressing both in an interactive video-on-demand environment.

In this paper, we introduced *Declustered Mirror*, a novel scheme that mirrors a set of striped disks to provide improved performance and fault tolerance for a video-on-demand system. Declustered mirror uses:

- *Disk stripping* to provide high I/O bandwidth for multimedia data.
- *Distributed Cyclic Layout (DCL)* and *Staggered Distributed Cyclic Layout (SDCL)* to provide a data layout scheme which supports a wide range of fast forward and rewind speeds under load balance conditions for interactive video-on-demand systems.
- *Mirroring* to provide fault tolerance for a single disk failure.
- *Chained Declustering* to provide load balance during failure mode operation.

The primary disadvantage of this scheme is the 100% space overhead of disk mirroring.

## References

- [1] K. Bates and M. TeGrotenhuis. Shadowing Boosts System Reliability. *Computer Design*, April 1985.

- [2] 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.
- [3] D. Bitton. Arm Scheduling in Shadowed Disks. In *Proceedings of the IEEE Computer Society International Conference (COMPCON)*, pages 132–136, San Francisco, California, February 1989.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] P. Chen and D. Patterson. Maximizing Performance in a Striped Disk Array. In *Proceedings of the 17th International Symposium on Computer Architecture*, volume 18, No. 2, pages 322–331, June 1990.
- [8] 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 Francisco, CA, October 1994.
- [9] B. Furht, D. Kaira, F. Kitson, A. Rodriguez, and W. Wall. Design Issues for Interactive Television Systems. *IEEE Computer*, pages 25–38, May 1995.
- [10] D. Gemmell, H. Vin, D. Kandlur, P. Rangan, and L. Rowe. Multimedia Storage Servers: A Tutorial. *IEEE Computer*, pages 40–49, May 1995.
- [11] H. Hsiao and D. DeWitt. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 456–465, Los Angeles, California, February 1990.
- [12] M. Kim. Synchronized Disk Interleaving. In *IEEE Transactions on Computers*, Vol. C-35, No. 11, November 1986.
- [13] 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.
- [14] M. Livny, S. Khoshafian, and H. Boral. Multi-Disk Management Algorithms. In *Proceedings of the 1967 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 69–77, Alberta, Canada, 1987.
- [15] K. Nwosu. *Data Storage Modeling and Management for Multimedia Information Systems*. PhD thesis, Syracuse University, Syracuse, New York, December 1993.
- [16] 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).

- [17] C. Orji, P. Bobbie, and K. Nwosu. Design and Configuration Rationales for Digital Video Storage and Delivery. Submitted to Journal of Multimedia Tools and Applications (JMTA).
- [18] 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).
- [19] 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.
- [20] 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.
- [21] Tandem. Configuring Disks. *Tandem Systems Review*, December 1986.