Failure Recovery Algorithms for Multimedia Servers [*]

Prashant J. Shenoy and **Harrick M. Vin**

Distributed Multimedia Computing Laboratory
Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188, USA

E-mail: {shenoy,vin}@cs.utexas.edu, Telephone: (512) 471-9732, Fax: (512) 471-8885
URL: http://www.cs.utexas.edu/users/dmcl

Abstract

In this paper, we present two novel disk failure recovery methods that utilize the inherent characteristics of video streams for efficient recovery. Whereas the first method exploits the inherent redundancy in video streams (rather than error-correcting codes) to approximately reconstruct data stored on failed disks, the second method exploits the sequentiality of video playback to reduce the overhead of online failure recovery in conventional RAID arrays. For the former approach, we present loss-resilient versions of JPEG and MPEG compression algorithms. We present an *Inherently Redundant Array of Disks (IRAD)* architecture that combines these loss-resilient compression algorithms with techniques for efficient placement of video streams on disk arrays to ensure that on-the-fly recovery does not impose any additional load on the array. Together, they enhance the scalability of multimedia servers by: (1) integrating the recovery process with the decompression of video streams, and thereby distributing the reconstruction process across the clients; and (2) supporting graceful degradation in the quality of recovered images with increase in the number of disk failures. We present analytical and experimental results to show that both schemes significantly reduce the failure recovery overhead in a multimedia server.

Keywords: Multimedia storage servers, redundant disk arrays, RAID, fault tolerance, video compression algorithms

[*]This research was supported in part by an IBM Faculty Development Award, Intel, the National Science Foundation (Research Initiation Award CCR-9409666 and CAREER award CCR-9624757), NASA, Mitsubishi Electric Research Laboratories (MERL), and Sun Microsystems Inc.

1 Introduction

Recent advances in computer and communication technologies have made it economically feasible to design sophisticated multimedia information services such as digital libraries, distance learning, online newspapers, etc. The realization of such services, however, requires the development of high performance, scalable multimedia servers that can efficiently store and retrieve multimedia objects. While computer users are accustomed to software failures and operating system crashes, customers of interactive services are not likely to tolerate anything but rare unavailability of these services [22]. Consequently, multimedia servers must employ techniques to guarantee high availability of services. A truly fault-tolerant design will support redundancies in all the key components of the server, including the cpu, memory, I/O, and network subsystems, as well as the system software. In this paper, we confine our focus to fault-tolerant designs for the I/O subsystem, and assume that existing fault-tolerant techniques will be used for other subsystems (for example, see [3, 13, 28]).

Due to the immensity of sizes and the data rate requirements of multimedia objects, multimedia servers are founded on *disk arrays*. Disk arrays connect several disks together, and thereby extend the cost, power, and size advantages of small disks to high capacity configurations [4]. A fundamental tradeoff, though, is that large disk arrays are highly susceptible to disk failures [7]. To illustrate, although the mean time to failure for a single disk is 300,000 hours, an array of 1000 disks will experience a failure every 12 days. Since the storage and retrieval of multimedia objects impose real-time constraints, to provide uninterrupted service, a multimedia server must continue to meet the real-time guarantees provided to clients even in the presence of disk failures. Hence, a fault-tolerant multimedia server must not only provide mechanisms to rapidly recover from a disk failure without losing data, but must also ensure that the recovery process operates without taking the system off-line and has minimal impact on system performance [14]. Since video is the most demanding data type (with respect to its data rate and real-time performance requirements), in this paper, we present failure recovery techniques tailored for video. Such techniques, however, can easily be extended to other data types such as images and animation sequences.

1.1 Relation to Previous Work

Recently several research projects have investigated the design of fault-tolerant storage systems [4, 18, 33]. Most of these approaches are based on the Redundant Array of Independent Disks (RAID) architecture [7, 24], which achieves fault tolerance either by *mirroring* or *parity encoding*. Mirroring (also referred to as RAID level 1) achieves fault tolerance by duplicating data on separate disks [2, 5, 9]. Many different mirroring techniques have been proposed: (1) *mirrored declustering* duplicates each disk onto another disk, (2) *interleaved declustering* uniformly distributes backup blocks of a disk amongst all the remaining disks, and (3) *chained declustering* stores backup blocks of disk i on disk $i + 1$. By maintaining two copies of the information, mirrored arrays achieve fault-tolerance by accessing backup blocks if the primary blocks are unavailable due to a disk failure. Even in the fault-free state, such arrays service read requests from the disk with the lighter load, thereby yielding better performance. A fundamental limitation of mirroring, however, is that it incurs a 100% storage space overhead.

Parity encoding techniques, employed in RAID levels 3, 4, and 5, reduce this overhead by employing error correcting codes [12, 24]. In a disk array consisting of D disks, parity is computed by an exclusive-or operation over data stored across $(D - 1)$ disks, and is stored on another disk [13, 17, 24]. The parity block together with all the data blocks over which parity is computed are referred to as a *parity group*. In RAID level 3 or *bit-interleaved parity*, data is interleaved bit-wise over the data disks and a dedicated disk stores all parity information. In RAID level 4 or *block-interleaved parity*, data is interleaved in fixed-size

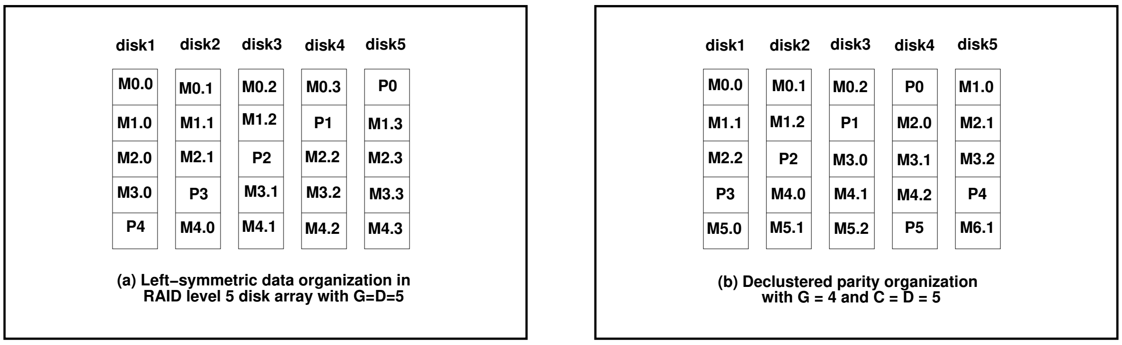

Figure 1 : Left-symmetric and declustered parity organizations in parity-based arrays. $M_{i.j}$ and P_i denote data and parity blocks, respectively, and $P_i = M_{i.0} \oplus M_{i.1} \cdots \oplus M_{i.(G-2)}$

blocks rather than in bits, with a dedicated disk storing parity blocks. Since parity must be updated on every write request, the parity disk can become a bottleneck in RAID levels 3 and 4. Hence, in RAID level 5 or *distributed block-interleaved parity*, parity blocks are uniformly distributed across all the disks (e.g., the left-symmetric parity assignment shown in Figure 1(a)). In parity-based RAID arrays, if one of the disks fail, the data on the failed disk is recovered by an exclusive-or operation over the data and the parity blocks stored on surviving disks. That is, a read access to a block on the failed disk causes one request to be sent to each surviving disk. Thus, if the system load is balanced prior to a disk failure, the surviving disks would observe twice as many read requests in the presence of a failure, causing a 100% increase in the load [15].

Several approaches that address this limitation by trading storage capacity for reduced failure recovery overhead have been proposed. In the *multiple* RAID architecture, an array of D disks is partitioned into clusters of C disks ($C \leq D$) with each cluster independently computing parity information [6, 7]. Whereas a standard RAID array can tolerate a single disk failure, such an architecture can tolerate a failure in each cluster without losing data. Furthermore, in the presence of a failure, only disks within the cluster containing the failed disk see an increased load while disks in other clusters continue to see their normal workload. The *declustered parity* array (also referred to as *clustered RAID*) further reduces the overhead of failure recovery by constraining the number of blocks protected by parity to $(G-1)$ instead of $(C-1)$, $G < C$ [14, 19, 21]. By appropriately distributing the blocks of a parity group across the C disks in a cluster, such a policy ensures that the load on each surviving disk in a cluster increases only by $(G-1)/(C-1)$ instead of $(C-1)/(C-1) = 100\%$ for read requests. This is illustrated in Figure 1(b) where $G = 4$ and $C = D = 5$.

In summary, several techniques for designing fault-tolerant disk arrays have been recently proposed [7]. However, most of the analysis of these failure recovery schemes have presumed a conventional workload consisting of aperiodic reads and writes. In contrast, a multimedia workload is dominated by large reads and infrequent writes, which are periodic and sequential. Furthermore, conventional workloads require good response times but no absolute guarantees, while multimedia workloads, due to their real-time nature, need bounded response times. Recently, several research groups have adapted conventional failure recovery techniques for multimedia servers [1, 8, 20, 28]. An admission control algorithm that reserves contingency disk bandwidth to accommodate load increases in the event of a disk failure in a declustered-parity-based multimedia server was presented in [23]. Similarly, in the *Streaming RAID* server [29], a RAID level 3 array is adapted to exploit the periodic nature of video accesses for efficient data retrieval. By restricting the maximum number of users simultaneously accessing the array, the server ensures that the real time requirements of video streams are not violated even during a disk failure.

A key limitation of these techniques is that they treat video data as an uninterpreted sequence of bits and do not exploit any of its characteristics. However, by exploiting the characteristics of video data, a multimedia server can significantly lower the overhead of online failure recovery. For instance, a multimedia server can exploit the sequential nature of video access to compute and prefetch parity information, and thereby reduce the overhead of failure recovery. Similarly, instead of perfectly recovering video data stored on the failed disk using error-correcting codes, a server can exploit human perceptual tolerances and the inherent redundancies in video streams to *approximately* reconstruct lost image data.

Failure recovery techniques that exploit the characteristics of the stored data to reduce the overhead of failure recovery have not received much attention and constitute the subject matter of this paper.

1.2 Research Contributions of This Paper

In this paper, we present two recovery techniques that utilize the inherent characteristics of video streams to minimize the overhead of on-the-fly disk failure recovery. The first approach exploits human perceptual tolerances and *spatial* and *temporal* redundancies inherent in video streams to approximately reconstruct image data stored on a failed disk. We illustrate our method by presenting loss resilient compression algorithms for JPEG and MPEG, referred to as *Loss-Resilient JPEG (LRJ)* and *Loss-Resilient MPEG (LRM)*, respectively. These algorithms partition each image in the video stream into several sub-images such that a reasonable approximation of an image can be constructed even when one or more of its sub-images are not available. We present an *Inherently Redundant Array of Disks (IRAD)* architecture that combines these loss-resilient compression algorithms with techniques for efficient placement of video streams on disk arrays to ensure that on-the-fly recovery does not impose any additional load on the array. Together, they enhance the scalability of multimedia servers by: (1) separating the tasks of online reconstruction of requested data from rebuild of failed disks onto spare disks, (2) integrating the recovery process with the decompression of video streams, and thereby distributing the reconstruction process across the clients, and (3) supporting graceful degradation in the quality of recovered images with increase in the number of disk failures. The IRAD architecture demonstrates the efficacy of a novel concept— imperfect recovery in disk arrays. Our imperfect recovery technique is equally effective in masking packet losses resulting from network congestion, and hence, is an end-to-end solution for failure recovery.

In the second method, we exploit the sequential nature of video stream accesses to reduce the overhead of on-line recovery in a RAID array. Specifically, by requiring that parity blocks be computed over a sequence of blocks belonging to the same video stream, the method ensures that data blocks retrieved by a server for failure recovery would be requested by the client in the near future. By buffering such blocks and then servicing requests for their access from the buffer, this method minimizes the overhead of on-line failure recovery. Our method reduces the recovery overhead to $1/(C-1)$ for declustered parity arrays, and to $1/(G-1)$ for RAID level 5 arrays. Moreover, the recovery scheme does not make any assumptions about the array architecture or the bit rates of video streams, and hence, is applicable to a variety of architectures as well as for VBR video streams.

To evaluate the efficacy of our algorithms, we have developed prototype codecs for the LRJ and LRM algorithms and have built an event-driven simulator. We evaluate our algorithms through extensive experimentation and simulations using VBR video streams. We present and analyze our results.

The rest of this paper is organized as follows: In Section 2, we present loss-resilient schemes for JPEG and MPEG, as well as the IRAD architecture. We present our parity-based failure recovery algorithm in Section 3. We analytically compare these failure recovery schemes to standard recovery schemes in Section

4. The experimental evaluation of our failure recovery schemes is presented in Section 5. Finally, Section 6 summarizes our results.

2 Exploiting Inherent Redundancy of Video Streams

Digitization of video yields a sequence of images.[1] We refer to a continuously recorded sequence of video images as a *media stream*. Due to the immense sizes of media streams, multimedia servers employ disks arrays as their underlying storage medium. Since disk arrays are highly susceptible to disk failures, these server employ failure recovery techniques to guarantee high availability of data. Conventional parity-based recovery techniques use error correcting codes to perfectly recover data stored on a failed disk. However, human perception is tolerant to minor distortions in video playback. Hence, a multimedia server can reduce the overhead of online failure recovery by exploiting the inherent spatial and temporal redundancies within video streams to approximately reconstruct lost images. To illustrate, let I denote an image in the video sequence, where

$$I = \{\ p(x,y)\ \mid\ p(x,y)\ \text{ is the pixel value at } (x,y)\}$$

Let each image I be partitioned into several sub-images $I_1, I_2, \cdots, I_N$ such that $I_i \subset I,\ \ 1 \leq i \leq N$, and $I_1 \cup I_2 \cup \cdots \cup I_N = I$. If these sub-images are stored on different disks, then a single disk failure will result in the loss of a fraction of each image. If the sub-images are created such that none of the immediate neighbors of a pixel in the image belong to the same sub-image, then even in the presence of a single disk failure, all the neighbors of the lost pixels will be available. In this case, the high degree of correlation between neighboring pixels will make it possible to reconstruct a reasonable approximation of the original image. Moreover, no additional information will have to be retrieved from any of the surviving disks for recovery. Since the images are partitioned in the pixel domain (i.e., prior to compression), we refer to the process as *pre-compression partitioning*.

Although conceptually elegant, such pre-compression image partitioning techniques significantly reduce the correlation between the pixels assigned to the same sub-image, and hence adversely affect image compression efficiency [26, 31]. The resultant increase in the bit-rate requirement may impose a higher load on each disk even in the fault-free state, and thereby reduce the number of video streams that can be simultaneously retrieved from the server. Alternatively, a server can employ *post-compression* partitioning techniques which partition each image into several sub-images *after* the redundancies within the video stream have been exploited by the compression algorithm. The key challenge in designing post-compression partitioning schemes is to create sub-images that facilitate effective and efficient recovery of lost image data without significantly affecting the compression efficiency. Most compression algorithms use a transform function (such as the *discrete cosine transform (DCT)* or the *wavelet transform*) that converts an image from the pixel domain to the frequency domain by packing most of the spectral energy into a small number of coefficients, thereby achieving compression. Consequently, post-compression partitioning depends on the characteristics of the transform function used to encode video streams. In what follows, we first describe the characteristics of the DCT, and then present a partitioning scheme that exploits these characteristics to effectively reconstruct lost transform coefficients. The partitioning scheme can be used with any compression algorithm using the discrete cosine transform. We illustrate our method by presenting failure resilient schemes for two popular compression algorithms used for video sequences (namely, motion JPEG and

[1]We shall use the terms image and frame interchangeably in this paper.

MPEG). By integrating these schemes with placement techniques, we derive a new disk array architecture for storing video streams.

2.1 Characteristics of the Discrete Cosine Transform

Compression algorithms based on the discrete cosine transform partition each image into nxn pixel blocks and independently apply the DCT to each block. The DCT uncorrelates each pixel block into an array of n^2 coefficients such that most of the spectral energy is packed into a small number of low frequency coefficients. Whereas the lowest frequency coefficient (referred to as the *DC coefficient*) captures the average brightness and color of the spatial block, the remaining set of $(n^2 - 1)$ coefficients (referred to as the *AC coefficients*) capture the details within the nxn pixel block. The coefficients produced by the DCT have the following characteristics:

- Since the DC coefficients capture the average brightness and color of each 8x8 pixel block and since the average brightness and color of pixels gradually change within a image, the DC coefficients of neighboring nxn pixel blocks are correlated. Consequently, the value of the DC coefficient of a block can be reasonably approximated from the DC coefficients of the neighboring blocks.

  To formally capture this observation, consider an image containing NCOL $\times$ NROW blocks of nxn pixels each. Let us define the *8-neighborhood* of a block at location (x, y) (denoted by $B(x, y)$) as the set:

  $$\mathcal{N}_8(B(x,y)) = \{B(i,j) \ \mid \ |x-i| \leq 1 \text{ AND } |y-j| \leq 1\} \ - \ \{B(x,y)\} \tag{1}$$

  Then, the DC coefficient of $B(x, y)$ can be approximated as:

  $$\mathrm{DC}_{B(x,y)} = \frac{1}{8} * \sum_{B_{(i,j)} \in \mathcal{N}_8(B(x,y))} \mathrm{DC}_{B(i,j)} \tag{2}$$

  where $\mathrm{DC}_{B(i,j)}$ denotes the DC coefficient of block $B(i, j)$.

- Due to the very nature of DCT, the set of AC coefficients yielded for each nxn block are uncorrelated. Moreover, since DCT packs the most amount of spectral energy into a small number low frequency coefficients, quantizing the set of AC coefficients (by using a user-defined normalization array) yields many zeroes, especially at higher frequencies. Consequently, recovering a block by simply substituting a zero for each of the lost AC coefficient is generally sufficient to obtain a reasonable approximation of the original image (at least as long as the number of lost coefficients are small and are scattered throughout the block).

2.2 Image Partitioning Fundamentals

To precisely describe the partitioning process, let $\mathcal{I}$ denote the frequency domain image obtained by applying the DCT to the original image I. Then, the partitioning algorithm exploits the characteristics of the DCT by proceeding in two steps:

1. *Scrambling:* In this step, the partitioning algorithm scrambles image $\mathcal{I}$ by uniformly distributing the AC coefficients of a DCT block across multiple blocks.

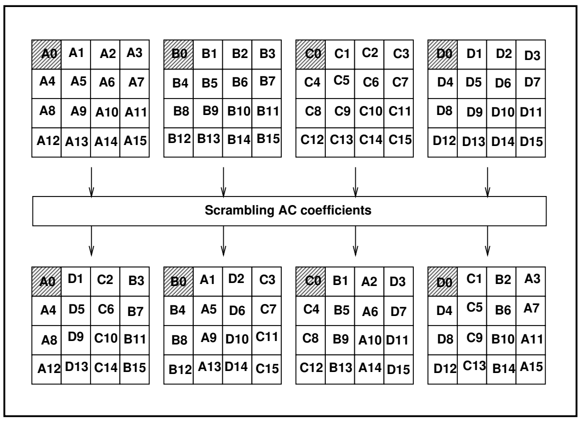

Figure 2 : Scrambling AC coefficients. Here A_0, B_0, C_0 and D_0 denote DC coefficients, and A_i, B_i, C_i and D_i $(1 \leq i \leq 15)$ represent AC coefficients.

2. *Sub-image creation:* The scrambled image $S(\mathcal{I})$ is then partitioned into a set of N sub-images such that none of the DC coefficients in the 8-neighborhood of a block belong to the same sub-image. If $\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_N$ denote the sub-images, then the partitioning algorithm also ensures that: (i) $\mathcal{I}_j \subset S(\mathcal{I}),\ 1 \leq j \leq N$, and (ii) $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \cup \mathcal{I}_N = S(\mathcal{I})$.

Note that this partitioning process is distinct from pre-compression image partitioning since the sub-images are created in the frequency domain as opposed to in the pixel domain. In fact, as we shall demonstrate later, it is this feature of the partitioning algorithm that enables reasonable failure recovery without incurring any significant degradation in compression efficiency. We present our partitioning algorithm by first describing a method for scrambling AC coefficients and then describing the sub-image creation process.

2.2.1 Scrambling AC Coefficients

Although substituting lost AC coefficients by zeroes yields a reasonable approximation of the original image, the degradation in image quality can be minimized by reducing the number of lost coefficients. The number of lost coefficients can be minimized by scattering the AC coefficients of a block amongst multiple sub-images. To achieve this objective, the partitioning algorithm employs a scrambling function f which when given a set of M DCT blocks, creates a new set of M blocks such that the AC coefficients from each of the input blocks are uniformly distributed amongst all of the output blocks. Furthermore, to prevent scrambling from adversely affecting compression efficiency, the scrambling function must ensure that the relative positions of AC coefficients in scrambled blocks is the same as that in the input blocks. Any scrambling function that satisfies these requirements can be used by the partitioning algorithm.

To describe one such scrambling function f, let $\mathcal{B}_0, \mathcal{B}_1, \cdots \mathcal{B}_{M-1}$ denote the DCT blocks of the original image. Then,

$$f(\mathcal{B}_0, \mathcal{B}_1, \cdots, \mathcal{B}_{M-1}) = \{\mathcal{B}'_0, \mathcal{B}'_2, \cdots, \mathcal{B}'_{M-1}\}$$

where $\mathcal{B}'_0, \mathcal{B}'_1, \cdots \mathcal{B}'_{M-1}$ denote the scrambled DCT blocks. Let us assume that the AC coefficients are numbered from left-to-right in a row-major order and that $\mathrm{AC}^k_{\mathcal{B}_i}$ denotes the k^{th} AC coefficient ($k \in [1, n^2 - 1]$) of block $\mathcal{B}_i$. The scrambling function f assigns $\mathrm{AC}^k_{\mathcal{B}_i}$ to be the k^{th} coefficient of block $\mathcal{B}'_j$ where $j = (i + k) \bmod M$. Thus, each resulting block contains $\frac{n^2}{M}$ coefficients of each of the original blocks. Specifically, one of the blocks contain the DC coefficient and $\left(\frac{n^2}{M} - 1\right)$ AC coefficients, and all the remaining $(M-1)$ blocks contain $\left(\frac{n^2}{M}\right)$ AC coefficients. Figure 2 illustrates the scrambling of AC coefficients for four 4x4 blocks.

2.2.2 Creating Sub-images

Having scrambled the image, it must then be partitioned into sub-images such that none of the DC coefficients in the 8-neighborhood of a block belongs to the same sub-image. The minimum value of the degree of image partitioning N (i.e., the number of sub-images) that satisfies this requirement is determined by the following theorem:

Theorem 1 *To ensure that none of the blocks contained in a sub-image are in the 8-neighborhood of each other in the original image, the image must be partitioned into at least 4 sub-images.*

Proof: We demonstrate that partitioning an image into 4 sub-images is both necessary and sufficient.

- *Necessity*: Consider a DCT block $\mathcal{B}(x,y)$ as well as all of the blocks in its 8-neighborhood. Let us partition the set of blocks into four groups G_1, G_2, G_3, and G_4, such that:

$$\begin{aligned} G_1 &= \{\mathcal{B}(x,y)\} \\ G_2 &= \{\mathcal{B}(x-1,y), \mathcal{B}(x+1,y)\} \\ G_3 &= \{\mathcal{B}(x-1,y-1), \mathcal{B}(x-1,y+1), \mathcal{B}(x+1,y-1), \mathcal{B}(x+1,y+1)\} \\ G_4 &= \{\mathcal{B}(x,y-1), \mathcal{B}(x,y+1)\} \end{aligned}$$

  By the definition of 8-neighborhood, it is clear that none of the groups contain any two blocks that are in the 8-neighborhood of each other. Moreover, none of the groups can be merged together without violating this condition. Hence, to ensure that no sub-image contains two blocks that are in the 8-neighborhood of each other, an image must be partitioned into at least 4 sub-images.

- *Sufficiency*: To prove the sufficiency, let us partition an image into 4 sub-images (denoted by $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$, and $\mathcal{I}_4$) as follows:

$$\begin{aligned} \mathcal{I}_1 &= \{\mathcal{B}(i,j) \mid (i \bmod 2) = 0 \text{ AND } (j \bmod 2) = 0\} \\ \mathcal{I}_2 &= \{\mathcal{B}(i,j) \mid (i \bmod 2) = 1 \text{ AND } (j \bmod 2) = 0\} \\ \mathcal{I}_3 &= \{\mathcal{B}(i,j) \mid (i \bmod 2) = 0 \text{ AND } (j \bmod 2) = 1\} \\ \mathcal{I}_4 &= \{\mathcal{B}(i,j) \mid (i \bmod 2) = 1 \text{ AND } (j \bmod 2) = 1\} \end{aligned}$$

  Since these definitions cover blocks in both odd and even numbered rows as well as columns, the 4 sub-images together contain all the blocks within the original image. Moreover, each block within the original image is a member of of exactly one of the sub-images.

  To demonstrate that none of the blocks contained in a sub-image belongs to the 8-neighborhood of each other, without loss of any generality, consider two distinct blocks $\mathcal{B}(i_1,j_1)$ and $\mathcal{B}(i_2,j_2)$ that belong to a sub-image. Then, by the definition of sub-images, either $|i_1 - i_2| \geq 2$ or $|j_1 - j_2| \geq 2$, or both. Hence, by definition (see Equation (1)), blocks $\mathcal{B}(i_1,j_1)$ and $\mathcal{B}(i_2,j_2)$ do not belong to the 8-neighborhood of each other. Thus, the definitions of sub-images $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$, and $\mathcal{I}_4$ denote a scheme for partitioning an image which guarantees that no sub-image contains blocks that are in the 8-neighborhood of each other. ■

2.2.3 Determining Image Partitioning Parameters

Before partitioning an image, the algorithm must first choose the values of parameters M and N (i.e., the number of DCT blocks produced by each invocation of the scrambling function and the degree of image partitioning, respectively). To enable good recovery of DC coefficients, the algorithm must choose $N \geq 4$, where the exact value of N is governed by the required quality of the reconstructed image. To minimize the impact of lost AC coefficients on the visual quality of an image, the partitioning algorithm must assign no more than one block from every set of M scrambled blocks to each sub-image. This requires that M be atmost N (i.e., $M \leq N$). Furthermore, since each scrambled blocks consists of $1/M$th of the AC coefficients from each of M input blocks, the number of lost AC coefficients can be minimized by choosing M as large as possible. Hence, the algorithm can minimize the degradation in image quality due to a failure by choosing $M = N$.

If an image is partitioned into 4 sub-images, then each sub-image will contain 25% of the image data in the frequency domain. Consequently, if the information contained in a sub-image is not available, the image will have to be reconstructed from the remaining 75% of the data. Since the quality of the reconstructed image is dependent on the amount of original image data available for reconstruction, increasing the degree of image partitioning improves the quality of the reconstructed images. However, as we shall point out later, increasing the degree of image partitioning decreases the correlation between the DC coefficients of blocks assigned to the same sub-image, and thereby deteriorates compression efficiency. Hence, the degree of image partitioning must be chosen so as to simultaneously optimize the quality of reconstructed image and the compression efficiency. In what follows, we show how scrambling and sub-image creation can be combined to derive loss-resilient versions of JPEG and MPEG.

2.3 Loss-Resilient JPEG (LRJ) Algorithm

The JPEG compression algorithm groups image data into a sequence of 8x8 pixel blocks. Each pixel block is then subjected to the DCT which yields a DC coefficient and 63 AC coefficients. The DC coefficients of successive blocks are difference encoded using a DPCM scheme independent of the AC coefficients. Within each block, the AC coefficients are quantized to remove high frequency components, scanned in a zig-zag manner to obtain an approximate ordering from lowest to highest frequency, and finally run length and entropy encoded. Figure 3 depicts the main steps involved in the JPEG compression algorithm [25]. The motion-JPEG algorithm applies the JPEG algorithm to a sequence of images yielding a compressed video stream.

The *Loss-Resilient JPEG (LRJ)* algorithm is an enhancement of the JPEG compression standard, and uses the partitioning technique presented in the Section 2.2. Given that each image in a video stream is being partitioned into $N (N \geq 4)$ sub-images, the LRJ compression algorithm is as follows:

1. *Scrambling:* The algorithm selects N consecutive DCT blocks from the same row of the original image and scrambles the AC coefficients within the blocks to create a new set of N blocks. Since each invocation of the scrambling function requires N blocks from the same row, N is chosen such that it divides the total number of blocks in a row. In the event that this is not possible (e.g., when the number of blocks in a row are prime), each row of blocks is padded with additional "zero" blocks such

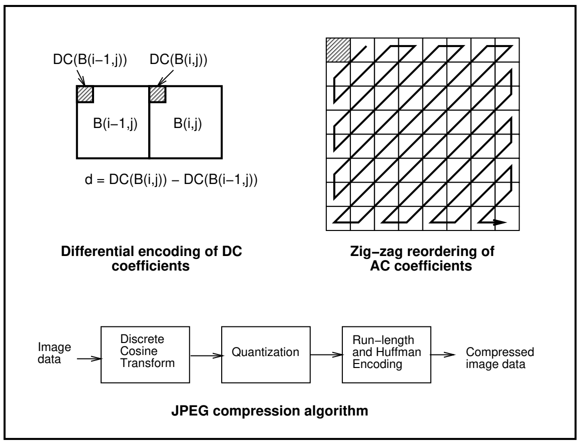
DC(B(i−1,j))
DC(B(i,j))
B(i−1,j)
B(i,j)
d = DC(B(i,j)) − DC(B(i−1,j))
Differential encoding of DC coefficients
Zig−zag reordering of AC coefficients
Image data
Discrete Cosine Transform
Quantization
Run−length and Huffman Encoding
Compressed image data
JPEG compression algorithm

Figure 3 : JPEG compression algorithm

that the number of blocks in a row becomes an integral multiple of N. Since all coefficients in such blocks are zeroes, they can be efficiently run-length and huffman encoded without any significant degradation in compression efficiency.

2. *Sub-image Creation:* Blocks obtained from each invocation of the scrambling function are then assigned to sub-images (one block per sub-image) such that none of the DC coefficients contained in a sub-image belong to blocks that are in the 8-neighborhood of each other. Since $N \geq 4$, the latter objective can be achieved by assigning the scrambled blocks belonging to a row to sub-images in a round-robin manner, and by ensuring that the assignment of the first block from each row is offset by 2 sub-images from the corresponding block in the previous row. That is, if block $\mathcal{B}'(i,j)$ is assigned to sub-image $\mathcal{I}_k$, then block $\mathcal{B}'(i+1,j)$ is assigned to sub-image $\mathcal{I}_{(k+1) mod N}$, $0 \leq i \leq \text{NCOL}$, $0 \leq j \leq \text{NROW}$. Moreover, if block $\mathcal{B}'(i,j)$ is assigned to sub-image $\mathcal{I}_k$, then block $\mathcal{B}'(i,j+1)$ is assigned to sub-image $\mathcal{I}_{(k+2) mod N}$.

Once all the blocks within the image have been processed, each of the N sub-images can be independently encoded. Specifically, the DC coefficients within each sub-image are encoded with a lossless DPCM scheme using the DC coefficient from the previous block assigned to the sub-image as a 1-D predictor. Similarly, the 2-D array of 63 AC coefficients within each block is formatted as a 1-D vector using a zigzag reordering, and then run-length and huffman encoded. Note that the huffman tables utilized for this purpose can either be optimized over each individual sub-image or over the entire image. Whereas the former approach will require a huffman table to be stored with each sub-image, the latter requires a single huffman table to be stored for an entire image. However, in such a scenario, the huffman table must be replicated across multiple sub-images to guarantee its availability table even when one or more of the sub-images are not available.

At the time of decompression, once each sub-image has been run-length and huffman decoded, the LRJ algorithm repeatedly selects a block from each of the N sub-images (referred to as the *merging* step) and uses an unscrambling function to obtain blocks of the original image. In the event that the information contained in a sub-image is not available, the unscrambling function also performs a predictive reconstruction of the lost DC coefficients from the DC coefficients of the neighboring 8x8 blocks. Lost AC coefficients, on the other hand, are replaced by zeroes. Since the scrambling function employed by the encoder ensures that each scrambled block contains coefficients from several blocks of the original image, the artifacts yielded by such

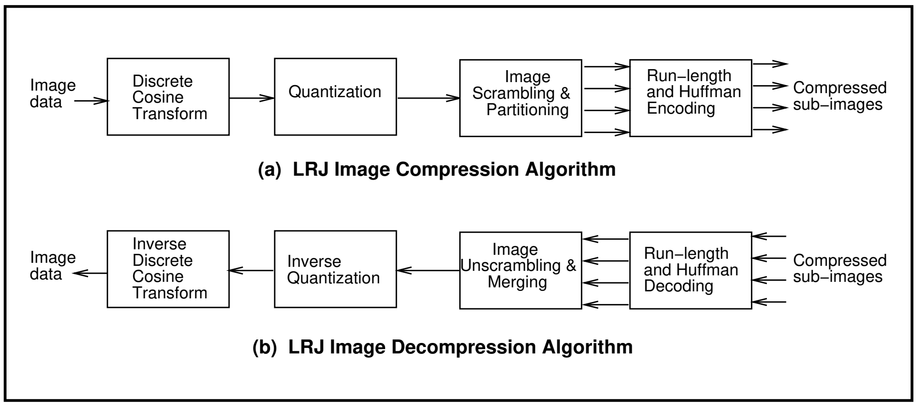
Image data
Discrete Cosine Transform
Quantization
Image Scrambling & Partitioning
Run−length and Huffman Encoding
Compressed sub−images
(a) LRJ Image Compression Algorithm
Image data
Inverse Discrete Cosine Transform
Inverse Quantization
Image Unscrambling & Merging
Run−length and Huffman Decoding
Compressed sub−images
(b) LRJ Image Decompression Algorithm

Figure 4 : LRJ compression and decompression algorithms

a recovery mechanism are dispersed over the entire reconstructed image, thereby significantly improving the visual quality of the image. Figure 4 depicts the various modules involved in the LRJ compression and decompression algorithm, and Figure 5 describes both of these algorithms in detail.

As a final note, observe that since successive blocks within a sub-image do not belong to the 8-neighborhood of each other, the correlation between their DC coefficients is smaller than the neighboring DC coefficients in the original image. The reduced correlation diminishes the efficiency of DPCM encoding of DC coefficients, and hence increases the total size of the compressed image (as compared to the corresponding JPEG image). Scrambling AC coefficients of a block across several sub-images, on the other hand, does not have any significant impact on the compressed image size. This is because, due to the very nature of DCT, AC coefficients are uncorrelated. Moreover, quantization yields a large number of zero coefficients. Since the scrambling algorithm does not alter the relative position of an AC coefficient within the zig-zag ordering, the effect of scrambling on the efficiency of run-length and huffman encoding is minimal. Thus, the increase in compressed image size yielded by the LRJ algorithm can be mostly attributed to the need for replicating huffman tables and the reduced correlation of successive DC coefficients.

2.4 Loss-Resilient MPEG (LRM) Algorithm

The MPEG compression standard exploits the large temporal and spatial redundancies present within an video stream to achieve a high degree of compression [11, 16]. MPEG supports three kinds of frames : (1) I frames (intra-coded without any reference to other frames), (2) P frames (predictively coded using an I or P frame), (3) B frames (bidirectionally interpolated from both the previous and the following I and/or P frame). To derive these types of frames, MPEG groups image data into 16x16 pixel areas called macro blocks. Macro blocks belonging to I frames are independently encoded. Macro blocks belonging to B and P frames, on the other hand, are temporally interpolated from corresponding reference frame(s), and the error macro block is computed as the difference between the actual and interpolated blocks. Macro blocks for which the encoder is unable to find a good reference block are intra-coded. The interpolation process also produces up to two motion vectors for each macro block, which denote the positions of the interpolated macro blocks in the reference frames. Regardless of the frame type, each macro block is then partitioned into six 8x8 pixel blocks — four luminance and two chrominance blocks. Each 8x8 pixel block is transformed into the frequency domain using the DCT. The DC coefficients of successive blocks are difference encoded.

```
Procedure LRJ-Compress
begin
    Input: Image consisting of NROW $\times$ NCOL
            8x8 pixel blocks and the value of $N$;
    Perform DCT on each 8x8 pixel block;
    Quantize the coefficients using a user-defined matrix;
    NCOL$'$ $=$ NCOL rounded to the next higher
                multiple of $N$;
    Pad each row of the transformed image with
        (NCOL$'$ $-$ NCOL) zero blocks;
    for ($i = 0$ to NROW $- 1$) do
        OFFSET $= (2 * i) \bmod N$;
        for ($j = 0$ to NCOL$' - 1$) do
            $k = (j +$ OFFSET$) \bmod N$;
            Assign $\mathrm{DC}_{B(i,j)}$ to sub-image $k$;
            for ($m = 1$ to 63) do
                Assign $\mathrm{AC}^{m}_{B(i,j)}$ to sub-image
                        $(k + m) \bmod N$;
            end for
        end for
    end for
    for each sub-image do
        DPCM encoding of the DC coefficients;
        Run-length encode the AC coefficients;
        Huffman encode the resultant stream;
    end for
end.
```

```
Procedure LRJ-Decompress
begin
    Input: $N$ sub-images;
    for each sub-image do
        Huffman decode the bit stream;
        Run-length decode AC coefficients;
        Inverse DPCM for DC coefficients;
    endfor
    for ($j = 0$ to NROW $- 1$) do
        OFFSET $= (2 * j) \bmod N$;
        for ($i = 0$ to NCOL$' - 1$) do
            $k = (i +$ OFFSET$) \bmod N$;
            if sub-image $k$ is available
                $\mathrm{DC}_{B(i,j)} \leftarrow$ next DC coefficient
                        from sub-image $k$;
            else
                Derive $\mathrm{DC}_{B(i,j)}$ from Equation 2;
            fi
            for ($m = 1$ to 63) do
                if sub-image $(k + m) \bmod N$ is available
                    $\mathrm{AC}^{m}_{B(i,j)} \leftarrow m^{th}$ AC coefficient from
                            sub-image $(k + m) \bmod N$;
                else
                            $\mathrm{AC}^{m}_{B(i,j)} \leftarrow 0$;
                fi
            end for
        end for
    end for
    Perform inverse quantization of each 8x8 block;
    Perform inverse DCT on each 8x8 block;
end.
```

Figure 5 : LRJ compression and decompression algorithms

The AC coefficients within a block are quantized, scanned in a zig-zag manner, and finally, run-length and entropy encoded. The motion vectors in P and B frames are also difference and entropy encoded.

Thus, while JPEG exploits only the spatial redundancies present within images, MPEG exploits both temporal and spatial redundancies present in image sequences. The key difference between the loss-resilient MPEG (LRM) algorithm and the LRJ algorithm is that LRM must also recover lost motion vectors of a macro-block in addition to lost DCT blocks. In order to recover lost DCT blocks, the LRM algorithm uses a partitioning algorithm similar to that in the LRJ algorithm. Since the least unit of encoding in MPEG is a macro block, the partitioning algorithm operates on macro blocks rather than DCT blocks. To precisely describe the LRM algorithm, let $\mathcal{B}^i(x,y)$ denote the i^{th} DCT block, $0 \leq i \leq 5$, in the macro block located at (x,y), and let the first four DCT blocks with each macro blocks denote luminance blocks. The algorithm proceeds in two steps:

1. *Scrambling:* Given N macro blocks from the same row of the image, the scrambling function takes the i^{th} DCT block of each macro block and scrambles them such that the AC coefficients of each block are uniformly distributed among the output blocks. That is,

$$f(\mathcal{B}^i(x,y), \mathcal{B}^i(x+1,y), \cdots \mathcal{B}^i(x+N-1,y)) = \mathcal{B'}^i(x,y), \mathcal{B'}^i(x+1,y), \cdots \mathcal{B'}^i(x+N-1,y)$$

2. *Sub-image Creation:* The partitioning algorithm assigns DCT blocks to sub-images such that none of the DC coefficients in the 8-neighborhood of a block belong to the same sub-image. Since each macro block contains blocks from the luminance as well as both chrominance components, the above property must hold for all three components of an image. To ensure this property, the partitioning algorithm partitions each image as follows : (1) block $\mathcal{B'}^0(x,y)$ is assigned to sub-image $\mathcal{I}_{(2x+4y)\ mod\ N}$, (2) if block $\mathcal{B'}^0(x,y)$ is assigned to sub-image $\mathcal{I}_k$, then block $\mathcal{B'}^i(x,y)$ is assigned to sub-image $\mathcal{I}_{(k+i)\ mod\ N}$, $0 \leq i \leq 3$, and (3) blocks $\mathcal{B'}^4(x,y)$ and $\mathcal{B'}^5(x,y)$ are assigned to sub-images $\mathcal{I}_{(x+2y)\ mod\ N}$ and $\mathcal{I}_{(x+2y+1)\ mod\ N}$, respectively. Figure 6 illustrates the sub-image creation process.

While a lost DC coefficient can be extrapolated from its neighboring DCT blocks, extrapolating a lost motion vector from neighboring macro blocks yields poor results due to the small correlation between motion vectors of adjacent macro-block within an image. Furthermore, if the MPEG encoder is unable to temporally interpolate a macro block from its reference frames, then the block is intra-coded. Thus, if the neighboring macro blocks of a macro block are intra-coded, there won't be any motion vectors to extrapolate from. Hence, to enable recovery even in such scenarios, the loss-resilient MPEG (LRM) algorithm must replicate motion vectors of macro blocks. In our algorithm, the motion vectors of a macro block stored in sub-image $\mathcal{I}_k$ are replicated in sub-image $\mathcal{I}_{k+1}$. Since MPEG allows applications to store any application-specific data in the user-defined section of the MPEG stream, such replication can be achieved without violating the syntax of MPEG. The primary motion vectors, on the other hand, are stored with their respective macro blocks, as dictated by the syntax of MPEG. Besides replicating motion vectors, the LRM algorithm also replicates the header information of each macro block in the user-defined section of a sub-stream. The macro block header contains information such as whether the macro block is intra-coded, predictively coded, or bidirectionally interpolated, which is needed by the decoder to decode the macro block. Such header information can be efficiently replicated since only a few bits per macro-block are needed to encode the information.

Given such encoded streams, if one of the sub-streams is unavailable, then the recovery process operates as follows: (1) DC coefficients of lost DCT blocks are extrapolated from neighboring DCT blocks, (2) lost AC coefficients are substituted by zeroes, and (3) lost motion vectors are recovered from the replicas stored

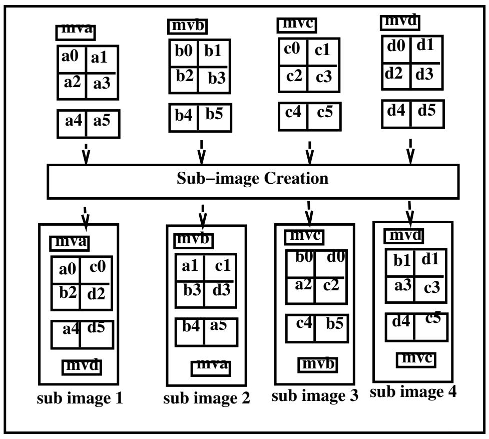

Figure 6 : Sub-image creation in LRM: $a0$ through $a5$ represent the six DCT blocks of macro block a and mva represents its motion vector. Successive DCT blocks within the same row of the luminance component (e.g., $a0, a1, b0, b1, ...$) are assigned to successive sub-images in a round-robin manner. Successive blocks within the same column (e.g., $a0$ and $a2$) are assigned to sub-images that are offset by 2 from each other. Successive blocks of each chrominance component (e.g., $a4, b4, c4...$) are mapped to sub-images in a round-robin manner.

in the user-defined sections of the surviving sub-streams. Observe that, since neighboring macro blocks can be encoded differently, a DC coefficient must be extrapolated from only those neighboring DCT blocks with the same encoding type. For instance, a DC coefficient contained in an intra-coded macro block must be extrapolated from only those neighboring DCT blocks belonging to intra-coded macro blocks. Thus, just as in the LRJ algorithm, a reasonable reconstruction of the image can be obtained by exploiting the inherent redundancies within the video stream. However, due to the dependencies between frames in MPEG, errors due to imperfect recovery in a frame can get propagated to other frames. That is, imperfect recovery of an I frame macro block can cause artifacts to appear in the blocks within the B and P frames which have been interpolated from it. The impact of such error propagation on the visual quality can be reduced by choosing a larger value of N, thereby reducing the fraction of the data lost per image. As we shall see in Section 5, the loss in compression efficiency caused by choosing larger values of N (as compared to LRJ) is not high.

2.5 Inherently Redundant Array of Disks (IRAD)

The LRJ or the LRM compression algorithms, when applied to a sequence of images constituting a video stream yield N sequences of sub-images. We refer to each such sequence as a *sub-stream*. For effective recovery, the server must ensure that the disks over which the sub-streams are striped do not overlap (i.e., even in the presence of a single disk failure, at least $(N-1)$ sub-streams are available). This can be achieved by striping each sub-stream over a mutually exclusive subset of disks in the array. We refer to a disk array architecture that employs such a placement strategy as an *Inherently Redundant Array of Disks (IRAD)*. In the event of a disk failure, clients use the LRJ or the LRM algorithm to approximately reconstruct lost image data. A careful analysis of this process of recovering from disk failures illustrates the following salient characteristics of the IRAD architecture:

- Since each image in the video stream is reconstructed by extrapolating information retrieved from the surviving disks, the failure recovery process does not impose any additional load on the disk array.

Consequently, the number of clients that can be serviced simultaneously by a multimedia server will be constrained solely by the playback rate requirements of each of the video streams during the fault-free state. Moreover, operating the server at very high levels of utilization during the fault-free state will not present any risk of saturation in the presence of failure.

- Since the recovery of lost image data is integrated with the decompression algorithm, the reconstruction process is carried out at client sites. This is an important departure from the conventional RAID architectures — distributing the functionality of failure recovery to client sites will significantly enhance the scalability of multi-disk multimedia servers.
- Since the recovery process only exploits the inherent redundancy in imagery, client sites will be able to reconstruct a video stream even in the presence of multiple disk failures. The quality of the reconstructed image, albeit, will degrade with increase in the number of simultaneously failed disks (i.e., IRAD supports graceful degradation in the image quality with increase in the number of failed disks).[2]
- Since the cause of the data loss is irrelevant to the recovery algorithm, the unscrambling algorithms in LRJ and LRM can be adapted to mask packet losses due to network congestion as well.[3]

Thus, the IRAD architecture provides an integrated, scalable, end-to-end solution for failure recovery.

In case of a disk failure, a redundant array must: (1) perform online reconstruction and thereby provide uninterrupted service to user requests, and (2) rebuild the failed disk onto a spare disk so that the array can revert back to the normal operating mode. Whereas the LRJ and LRM algorithms can be used to achieve the former objective in IRAD, rebuild can be accomplished either by restoring the failed disks from tertiary storage (i.e., tape backups), or by employing parity-based techniques. To rebuild failed disks from tertiary storage, the controller must allow a user to backup and restore each disk individually. Since rebuild of a failed disk from tertiary storage can operate independently of other disks in the array, the rebuild operation does not impose any additional load on the surviving disks. Alternatively, if parity information is used to rebuild the contents of the failed disk, then on-line rebuild onto a spare disk can proceed simply by issuing low-priority read requests to access media blocks from each of the surviving disks [15]. Note that, the array controller still depends on the LRJ/LRM algorithms for *online reconstruction* of user requested images while the parity information is used for *online rebuild* of the failed disk. The choice of a particular rebuild technique is environment dependent: whereas rebuild from tertiary storage will suffice for predominantly read-only environments, parity-based rebuild will be required for environments with frequent writes.

3 Exploiting Sequentiality of Video Retrieval

In the previous section, we presented failure techniques that approximately reconstruct data stored on failed disks. While approximate reconstruction is adequate for most continuous media applications, certain demanding continuous media applications require perfect reconstruction of data (e.g., a video clip showing a cat scan of a brain tumor). For such applications, we present a failure recovery scheme that exploits

[2]Observe that, to tolerate multiple disk failures, multiple copies of huffman tables and motion vectors would have to be maintained.

[3]Several techniques have been proposed which scramble media streams prior to network transmission to enable approximate reconstruction in case of packet losses. [10, 26]. The efficacy of these techniques validates our claim.

the sequential nature of continuous media accesses to perfectly recover data stored on failed disks without imposing a large overhead on the server.

3.1 Parity-based Reconstruction

Consider a multimedia server that employs a disk array for storing continuous media streams. To effectively utilize the array bandwidth, the server interleaves (i.e., stripes) each media stream among disks in the array. The unit of data interleaving, referred to as a *media block* or a *stripe unit*, denotes the maximum amount of logically contiguous data that is stored on a single disk. In addition to a sequence of media blocks for each video stream, to recover from disk failures, the server maintains parity blocks on the array. Parity blocks are computed by an exclusive-or operation over all media blocks within the parity group.

To describe the fault-free operation, consider a scenario where the server is servicing n clients, each retrieving a video stream (say $S_1, S_2, ..., S_n$, respectively). Let $\mathcal{R}_i$ denote the playback rate (expressed in images/sec) of video stream $S_i, \quad 1 \leq i \leq n$. Due to the periodic nature of media playback, a multimedia server services these clients by proceeding in *rounds*. During each round, the server retrieves a fixed number of images for each client. Thus, if $\mathcal{T}$ denotes the duration of a round, then to ensure continuous playback, the number of images of streams $S_1, S_2, ..., S_n$ retrieved during each round is given by: $f_i = \mathcal{T} * \mathcal{R}_i; \quad \forall i \in [1, n]$. To access $f_1, f_2, ..., f_n$ images, the server will be required to retrieve $k_1, k_2, ..., k_n$ media blocks, respectively, of streams $S_1, S_2, ..., S_n$,. If a media stream is encoded using a variable bit rate (VBR) compression algorithm, then the sizes of images will vary. Hence, the mapping from f_i to k_i may vary from one round to another. For media streams encoded using a constant bit rate (CBR) compression algorithm, on the other hand, the mapping from f_i to k_i is fixed. Regardless of the actual compression algorithm, during the fault-free state, a server only retrieves media blocks and skips over all the parity blocks.

In the presence of a disk failure, when a client requests the retrieval of a block stored on the failed disk, the server must access the parity and data blocks stored on the surviving disks to recover the lost information. In the simplest case, the server retrieves blocks of the parity group to recover lost information, transmits the set of requested blocks to the client site, and then discards the additional blocks. Although relatively simple to implement, the transient increase in load on disks induced by such a scheme may yield service times (i.e., the total time spent in retrieving images during a round) that exceed $\mathcal{T}$, resulting in playback discontinuities at client sites.

Alternatively, consider a multimedia server that always computes parity over a sequence of $(G - 1)$ media blocks from the *same* video stream [32]. That is, all $(G - 1)$ media blocks within a parity group are successive blocks of the same video stream. In such a scenario, the server can exploit sequentiality of media playback to minimize the increase in load due to a failure by using media blocks retrieved during playback for failure recovery and vice-versa. Observe that the server can recover blocks stored on the failed disk either in the round in which they are accessed, or at least one round prior to their access. These recovery policies are referred to as *lazy* and *eager* failure recovery, respectively. In what follows, we present these policies in detail and discuss their tradeoffs.

To describe the lazy failure recovery algorithm, consider a disk array with a parity group size of G. Let the set of disks within a parity group be denoted as $1, 2, ..., G$, and let us assume that, disk $i (i \leq G)$ fails in round r. In such a scenario, for each block accessed from the failed disk during round r, the server will access an additional block from each of the surviving $(G - 1)$ disks in the parity group. Then, due to sequentiality of video playback, a subset of the $(G-1)$ blocks (namely, blocks from disks $(i+1)$ through G) accessed for recovering the lost block will be requested by the client within the next few rounds. The server

```
Procedure OnlineParity
begin
    Let $R_j$ denote the set of $k_j$ blocks to be retrieved for client $j$.
    for every block b in set $R_j$ do
        if $b$ belongs to the same parity group as the blocks over which online parity $p_j$ has been computed then
            $p_j = p_j \oplus b$
        else
            $p_j = b$ {* new parity group starts with $b$ *}
        fi
    end for
end.
```

Figure 7 : Procedure for computing online parity

can buffer these blocks and service requests for their access from the buffer. Consequently, an increase in load on disks $(i+1)$ through G during round r due to a client will be followed by a corresponding reduction in load due to that client in the next few rounds. Observe that, while blocks from disks $(i+1)$ through G are reused for servicing retrieval requests, blocks from disks 1 through $(i-1)$ are not. Hence, these blocks must be retrieved for reconstructing lost blocks and subsequently discarded. However, due to the sequential nature of playback, the client would have accessed blocks on disk 1 through $(i-1)$ in the past few rounds for normal playback. Hence, the server can further reduce the overhead of failure recovery by maintaining an exclusive-or of these blocks when the client accesses them for playback. The server can then use the result of this exclusive-or computation for reconstructing the block on disk i, instead of retrieving these blocks again from the array. We refer to the result of this exclusive-or computation as *online parity*. The server can maintain online parity only in the event of a disk failure or even in the fault-free state. Since a disk failure can not be anticipated in advance, maintaining online parity even in the fault free state minimizes recovery overhead. On the other hand, since parity computations can be expensive, maintaining parity only in the event of a disk failure and only for the parity group containing the failed disk significantly reduces parity computation overheads. However, this approach yields a transient increase in the load on disks 1 through $(i-1)$ immediately after a disk failure, since additional blocks must be accessed from these disks to reconstruct lost blocks. The exact algorithms for maintaining online parity and lazy failure recovery are described in Figures 7 and 8, respectively. The lazy recovery algorithm presented in Figure 8 assumes that online parity is maintained even in the fault-free state, and can be easily modified if online parity is maintained only in presence of disk failures.

In the eager failure recovery algorithm, on the other hand, instead of recovering blocks stored on a failed disk only when they are accessed, the multimedia server exploits the sequentiality of video playback to prefetch data blocks so that blocks on the failed disk are recovered sufficiently prior to their access. To precisely describe the recovery algorithm, let $k_j^{\max}$ and $k_j^{\min}$, respectively, denote the maximum and the minimum number of media blocks requested by client j in a round. Then the following theorem determines the number of blocks that must be prefetched to reconstruct blocks on the failed disk prior to their access.

Theorem 2 *By maintaining $(k_j^{\max} + G - 2)$ blocks per client in the buffer, the server can ensure the reconstruction of lost media blocks at least one round prior to their access.*

```
Procedure LazyRecovery
begin
  Let $R_j$ denote the set of $k_j$ blocks to be retrieved for client $j$ and $p_j$ denote its online parity block.
  for each client $j$ do
    if $R_j$ consists of a block on the failed disk $i$ then
      Let $b_l$ denote the lost block
      Let $Q_j$ consist of data blocks from disks $(i+1)$ through $G$ for client $j$
      Retrieve blocks $R_j - \{b_l\}$.
      Retrieve blocks $Q_j - R_j$. {* Retrieve any remaining blocks of the parity group *}
      Retrieve the parity block $p$
      $b_l = p_j \oplus p \oplus$ blocks of the parity group in $Q_j \cup R_j$
      Buffer blocks in $Q_j - R_j$ for future rounds.
    else
      Retrieve blocks in $R_j$ (do not retrieve blocks present in the buffer).
    fi
    Compute online parity $p_j$ over $R_j$
    Schedule blocks of $R_j$ for transmission to client $j$.
  end for
end.
```

Figure 8 : Lazy Failure Recovery Algorithm

Proof: To reconstruct a media block stored on the failed disk, all the blocks in its parity group must be accessed. Thus, in the worst case, if $k_j^{\max}$ blocks are requested by client j from the server during a round, and if the last of these blocks is stored on the failed disk and none of the remaining $(k_j^{\max} - 1)$ requested blocks belong to its parity group, then the server will require $(G-1)$ additional blocks to recover the lost block. Hence, if the server ensures that $\left((k_j^{\max} - 1) + (G-1)\right) = (k_j^{\max} + G - 2)$ blocks have been prefetched into its buffer, then it can reconstruct any lost block at least one round prior to its access. ■

In the simplest case, regardless of the presence or absence of failures, a multimedia server can prefetch $(k_j^{\max} + G - 2)$ blocks per client prior to initiating playback. In such a scenario, if a client requests the retrieval of k_j blocks in a round, then the server transmits these blocks from its buffer and retrieves the next k_j blocks from the array so that $(k_j^{\max} + G - 2)$ blocks are always in its buffer. By maintaining $(k_j^{\max} + G - 2)$ blocks per client, the server ensures that an entire parity group is retrieved and buffered at least one round before any of its blocks are accessed. Hence, media blocks on the failed disk can always be reconstructed prior to their access. The disadvantage of this approach is the increase in initiation latency experienced by clients due to the prefetch operation. Alternatively, the server can maintain $(k_j^{\max} + G - 2)$ blocks per client only in the event of a disk failure. In this case, assuming that the i^{th} disk $(i \leq G)$ of the parity group fails in round r, the server suspends transmission of media blocks to clients until $(k_j^{\max} + G - 2)$ blocks are prefetched for each client. Observe that, if the server is lightly loaded, then the duration for which transmission (and hence, the playback) is required to be suspended is relatively short. In the worst case, however, the prefetch operation may take up to $\left\lceil \frac{k_j^{\max} + G - 2}{k_j^{\min}} \right\rceil$ rounds. In addition to these blocks, if the first of the requested blocks for a client in round r is stored on disk m of the parity group, and if $m \leq i$, then the server must retrieve additional blocks from disks 1 to $(m-1)$ to reconstruct the lost block.

```
Procedure EagerRecovery
begin
for each client $j$ do
    (1) Suspend transmission of blocks, and prefetch $(k_j^{\max} + G - 2)$ media blocks
    (2) If the first requested block in $R_j$ is on the $m^{th}$ disk of the parity group of the failed disk and $m \leq i$ then
            Retrieve additional media blocks from disks $1$ though $m - 1$.
            Retrieve the parity block and reconstruct block on disk $i$.
            Discard blocks retrieved from disks 1 through $m - 1$.
        end if.
    (3) Resume transmission of media blocks to the user.
    (4) If in a round, $k_j$ prefetched blocks are transmitted to the user then
            Retrieve the next $k_j$ blocks from the disk array so that $(k_j^{\max} + G - 2)$ blocks are always in the buffer.
            If a requested block belongs to a failed disk then
                Retrieve the corresponding parity block instead.
            end if.
        end if.
    (5) Reconstruct lost blocks as soon as all remaining blocks within its parity group have been read into
        the buffer.
end for.
end.
```

Figure 9 : Eager failure recovery algorithm

These additional blocks are discarded by the server after reconstructing the block on disk i. On resuming transmission (and hence, playback), the server retrieves sufficient number of blocks in each round so as to ensure that the $(k_j^{\max} + G - 2)$ blocks per client are always in buffer. Observe that this approach shifts the latency from playback initiation time to the time when the server experiences a disk failure. Since disk failures are infrequent events, most clients will not experience this latency in the common case. However, a disadvantage of the approach is the temporary pause in playback that clients experience immediately after a disk failure. The precise eager failure recovery algorithm is described in Figure 9. The algorithm presented in Figure 9 assumes that media blocks are prefetched only in the event of a disk failure, and can be easily modified if blocks are prefetched prior to initiating playback.

3.2 Failure Recovery Overheads

Whereas in standard RAID each media block within the parity group would be accessed twice in the worst case, once for playback and once for reconstructing the lost block, in the lazy and eager schemes each media block is accessed precisely once. Thus, the only additional load on the disks is due to the retrieval of parity blocks, thereby considerably reducing the overhead of failure recovery. Recall that, the lazy scheme causes a sequence of load fluctuations since an increase in load due a client (caused by accessing a block on the failed disk) is followed by a reduction in load due to that client in the following rounds. Hence, the fundamental difference between the lazy and the eager recovery algorithms is that the latter trades buffer space to replace a sequence of load fluctuations possible in the former by a constant increase in the load.

Observe that, both the lazy and the eager recovery schemes make no assumptions about the array architecture, and hence can be used with many different architectures. In what follows, we analyze the

Table 1 : Overhead of the lazy/eager algorithms

	Standard Array	Array with lazy/ eager recovery
RAID level 4	Data Disks: 100 % increase Parity Disk: same load as the failed disk prior to fault	Data Disks: No increase Parity Disk: same load as the failed disk prior to the fault
RAID level 5	$\frac{G-1}{G-1} = 100$ % increase	$\frac{1}{G-1}$
Declustered parity	$\frac{G-1}{C-1}$	$\frac{1}{C-1}$
Flat parity	Data Disks: 100% increase Parity Disk: $\frac{1}{C-(G-1)}$	Data Disks: No increase Parity Disk: $\frac{1}{C-(G-1)}$

overhead of these schemes for various architectures including RAID level 4, RAID level 5, declustered parity, etc., and compare it with standard recovery techniques.

Assuming that the load on each disk is balanced prior to a failure, the recovery overhead for different architectures is shown in Table 1. Since the only additional blocks that are retrieved by the lazy and eager schemes are parity blocks, the data disks in a RAID level 4 array experience no increase in the load. The parity disk, however, experiences a load equal to that of the failed disk prior to failure (since every access to the failed disk causes an access on the parity disk). For RAID level 5 arrays, since parity blocks are uniformly distributed among all disks in the parity group (see Fig 1(a)), the recovery overhead is $1/(G-1)$. This is a significant reduction over the 100% load increase seen by each disk in standard RAID level 5 arrays. In declustered parity arrays, since parity blocks are uniformly distributed across the $(C-1)$ surviving disks within the cluster, the recovery overhead is $1/(C-1)$. Lastly, consider a uniform flat parity placement scheme in which: (1) each cluster is partitioned into groups of $(G-1)$ disks (i.e., $C = n \cdot (G-1),\ n = 1, 2, ..$), and (2) each group of $(G-1)$ disk stores the data blocks of a parity group with the parity block uniformly distributed among the remaining $C-(G-1)$ disks within the cluster. In the presence of a failure, while the $(G-1)$ disks storing data blocks see no increase in the load, the remaining $(C-(G-1))$ disks see a load increase due to retrieval of parity blocks. Thus, the overhead of failure recovery on these disks is $1/(C-(G-1))$.

3.3 Discussion

Recently a scheme similar to the lazy recovery algorithm was proposed in [1]. In this scheme, on experiencing a disk failure, the cluster with a failed disk switches to the degraded mode with requests reading and buffering entire parity groups at a time. Thus, the entire cluster acts like a single logical disk. In the lazy recovery algorithm, accessing a block on the failed disk i causes blocks from disk $(i+1)$ through G to be retrieved. Thus, disks $(i+1)$ through G act as a single logical disk, and only blocks of the parity group retrieved from these disks need to be buffered. Whereas both schemes have identical worst case buffer requirements, the lazy scheme has a lower average case buffer requirement.

Recall that, both the lazy and the eager failure recovery algorithms require that all media blocks contained within a parity group belong to the same video stream. However, most of the existing array controllers provide the abstraction of a single large disk addressable by logical blocks numbers to the operating system software. Thus, details such as the logical to physical block mapping, membership of a parity group, etc., are implemented by the controller logic and are hidden from the operating system. Without

Table 2 : Comparative evaluation of fault-tolerant schemes

	multiple RAID-5	Declustered Parity (Standard)	Declustered Parity (Lazy/Eager)	IRAD
Storage Overhead	$1/C$	$1/G$	$1/G$	depends on N
Load Overhead	$(G-1)/(G-1)$	$(G-1)/(C-1)$	$1/(C-1)$	0
Buffer (fault-free)	$nbk^{\max}$	$nbk^{\max}$	eager: $nbk^{\max}$ lazy: $nb(k^{\max}+1)$	$nb'Nk'^{max}$
Buffer (fault)	$nb[I(k^{\max}+G-1)$ $+(D-I)k^{\max}]/D$	$nb[I(k^{\max}+G-1)$ $+(D-I)k^{\max}]/D$	eager: $nb(k^{\max}+G-1)$ lazy: depends on $k^{\min}$	$nb'Nk'^{\max}$
MTTDL	$\frac{MTTF^2}{D(C-1)MTTR}$	$\frac{MTTF^2}{D(C-1)MTTR}$	$\frac{MTTF^2}{D(C-1)MTTR}$	depends on the rebuild algorithm

these details, a multimedia server can not determine the block numbers of data blocks constituting a parity group, and hence can not control the membership of media blocks within a parity group. Consequently, to implement our failure recovery algorithms, conventional array controllers must be suitably extended. Specifically, if the controller can implement a function that takes a logical block number as its input and returns a list of logical block numbers of all blocks which belong to its parity group, then a multimedia server can exercise precise control over membership of blocks within a parity group.

Since a multimedia workload is dominated by read requests, the preceding discussion focussed on the overhead of read requests and ignored write requests. We assume that standard techniques such as full-stripe writes in which entire parity groups are written at a time will be used for the large sequential writes seen in a multimedia workload. Regardless of the presence or absence of failures, in full-stripe writes, the server computes the new parity information and writes it to the appropriate disk along with the data blocks in the parity group. In case of a disk failure, the server either discards the block to be stored on the failed disk, or writes it to a replacement disk. Since the array operation is unchanged in the presence of failures, these full-stripe writes impose no extra overhead on the server as compared to the fault-free state.

4 Comparative Evaluation

A comparison of the schemes presented in this paper with standard RAID level 5 and declustered parity arrays is shown in Table 2. The array architectures are compared with respect to their storage space overhead, overhead imposed by to failure recovery, buffer space requirements, and mean time to data loss (MTTDL).

Storage Space Overhead

To compute the storage space required to maintain parity information, consider a disk array with a cluster size of C. Then, the storage space overhead of a RAID level 5 array is $1/C$, while that of a declustered parity array is $1/G$. Since $G < C$, declustered parity arrays have a higher overhead as compared to RAID level 5 arrays. For the IRAD architecture, the storage space overhead is primarily due to the loss in compression efficiency during image partitioning in the LRM and LRJ algorithms. The replication of motion vectors adds to this overhead in the LRM algorithm. The overhead increases initially with increase in the degree of image partitioning, and then becomes independent of the degree of image partitioning for large values of N. Finally, if the IRAD architecture rebuilds failed disks using parity information rather than from tertiary storage, then it incurs an additional overhead of $1/C$.

Failure Recovery Overhead

Recall from Section 1 that the recovery overhead due to read requests for RAID level 5 and declustered parity arrays is $(G-1)/(G-1)$ and $(G-1)/(C-1)$, respectively. As shown in Section 3, if the lazy and eager recovery algorithms are used, this overhead reduces to $1/(G-1)$ and $1/(C-1)$ for RAID level 5 and declustered parity arrays, respectively. For the IRAD architecture, on the other hand, since the array operation is unchanged even in the presence of a failure, there is no overhead due to failure recovery. However, this is at the expense of a slightly higher load in the fault free state caused by the degradation in compression efficiency due to image partitioning in LRJ and LRM.

Buffer Requirements

To compute the buffer overhead, let us assume that the load on each disk in the array is balanced in the fault-free state. Let n denote the total number of clients accessing the disk array, and let b denote the media block size. Since in the worst case, each client accesses $k^{\max}$ blocks in the fault-free state, the total buffer requirement for RAID level 5 and declustered parity arrays is $nbk^{\max}$. Assuming that the eager recovery scheme prefetches media blocks only in the event of a failure, its fault-free buffer requirement is $nbk^{\max}$. Similarly, assuming that the lazy scheme maintains online parity even in the fault free state, it incurs a buffer requirement of $nb(k^{\max}+1)$. In the IRAD architecture, assuming that each client accesses $k'^{\max}$ blocks of size b' from each of the N sub-streams during a round, the total buffer required is $nb'Nk'^{\max}$ ($k'^{\max}$ and b' can be distinct from those for parity-based arrays). While choosing $b'=b$ yields an array utilization that is comparable to parity-based arrays, it increases the buffer space requirement. On the other hand, choosing $b'<b$ lowers the buffer required for IRAD architectures at the expense of a lower array utilization. Hence, the block size must be chosen to balance these tradeoffs.

To compute the buffer requirement in the presence of disk failures, let us assume that I clusters in the array have experienced a single disk failure ($I \leq D/C$). Further, assume for simplicity, that a client accesses at most one failed disk in each round. For RAID level 5 and declustered parity arrays, in the worst case, each client accessing a block on the failed disk would access $(G-1)$ additional blocks. Since each disk is accessed by n/D clients and the array contains I failed disks, the total buffer required is $nb(k^{\max}+G-1)I/D+nbk^{\max}(D-I)/D$. In the eager recovery scheme, the server buffers $(k^{\max}+G-2)$ blocks per client and requires an additional block per client to store the reconstructed block. Hence, the total buffer required is $nb(k^{\max}+G-1)$. The buffer requirement for lazy recovery is dictated by the following lemma:

Lemma 1 *The worst case buffer requirement for the lazy failure recovery algorithm is*

$$\begin{array}{ll} nb(k^{\max}+G-1)I/D+nbk^{\max}(D-I)/D & \textit{if } (G-2) \leq k^{\min} \\ ((G-2)(j+1)+k^{\max}-k^{\min}j(j-1)/2)Inb/D+(n-(j+1)In/D)bk^{\max} & \textit{otherwise} \end{array}$$

where $k^{\min}$ denotes the minimum number of blocks accessed by a client in a round, $k^{\min} \geq 1$, and $j = \lfloor \frac{G-2}{k^{\min}} \rfloor$.

Proof: In the lazy scheme, when a client requests a block on the failed disk i, the server accesses and buffers blocks stored on disks $(i+1)$ through G of the parity group. Then in the worst case, the server will buffer $(G-2)$ data blocks per client and these blocks will be used to service requests in the following $(\frac{G-2}{k^{\min}})$ rounds. If $(G-2) \leq k^{\min}$, then all the $(G-2)$ buffered blocks would be accessed by the client in the following round. In such a scenario, the total buffer required would be the same as that in a RAID 5 array. That is, total buffer required is $nb(k^{\max} + G - 1)I/D + nbk^{\max}(D - I)/D$.

On the other hand, if $(G - 2) > k^{\min}$, then in the worst case, the server uses the buffered blocks to service client requests for up to $j = \lfloor \frac{G-2}{k^{\min}} \rfloor$ rounds after the client has accessed a block on the failed disk. To compute the buffer requirement in this scenario, consider the set of clients who have accessed a failed disk in the current round or the any of previous j rounds. The buffer required for these clients is

$$[(k^{\max} + G - 2) + (G - 2) + (G - 2 - k^{\min}) + (G - 2 - 2k^{\min}) \cdots + (G - 2 - (j - 1)k^{\min})]Inb/D$$

Simplifying, this yields

$$[k^{\max} + (G - 2)(j + 1) - k^{\min}(1 + 2 + \cdots + (j - 1))]Inb/D$$

or

$$[k^{\max} + (G - 2)(j + 1) - k^{\min}j(j - 1)/2]Inb/D$$

The buffer required for the remaining clients is $[n - (j + 1)In/D]bk^{\max}$. Thus, when $k^{\min} < (G - 2)$, the total buffer required is

$$[k^{\max} + (G - 2)(j + 1) - k^{\min}j(j - 1)/2]Inb/D + [n - (j + 1)In/D]bk^{\max} \tag{3}$$

$\blacksquare$

Lastly, for the IRAD architecture, since no additional blocks are accessed by a client in the presence of failures, there is no increase in the buffer requirement.

Mean Time To Data Loss

RAID level 5, declustered parity, and the lazy/eager recovery based disk arrays experience data loss if a disk within a cluster fails while another disk within that cluster is being rebuilt. The mean time to data loss (MTTDL) for these architectures is given as

$$MTTDL = \frac{MTTF^2}{D(C - 1)MTTR} \tag{4}$$

where $MTTF$ is the mean time to failure for an individual disk, and $MTTR$ is the mean time to rebuild a failed disk [7]. To illustrate, consider an array of 32 disks and a cluster size of 8. If the $MTTR$ of a disk is 2 hours and its $MTTF$ is 300,000 hours, then the mean time to data loss for the array is about 23,000 years. Since the $MTTR$ of a disk is proportional to the load on the array during the online rebuild process, and since the lazy/eager recovery schemes impose a lower recovery overhead on the array, they have a higher $MTTDL$ than standard RAID level 5 or declustered parity arrays.

The $MTTDL$ for the IRAD architecture, on the other hand, depends on the rebuild algorithm used by the array. If parity information is used to rebuild failed disks, then the MTTDL is given by Equation 4. Since a disk failure imposes no additional load on the surviving disks, the IRAD architecture has a lower $MTTR$ for a failed disk, and hence, a higher $MTTDL$ as compared to RAID level 5 or declustered parity architectures. For IRAD arrays that rebuild failed disks from tertiary storage, it is not meaningful to compute $MTTDL$ since lost data can always be recovered from backup tapes. Hence, we define a new metric to compute the resilience of the array to failures. The mean time to shutdown ($MTTS$) for the IRAD architecture is defined as the average time before the array is taken offline for repairs. Since the architecture

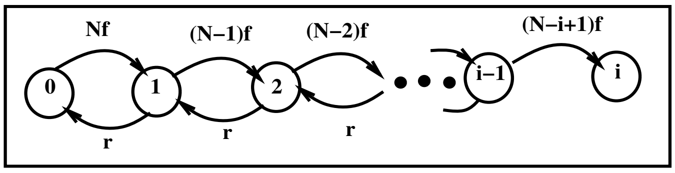

Figure 10 : Markov model for determining MTTS. State j represents j failed disks in a cluster. With j failed disks in a cluster, the failure rate is $(N - j)f$, and the repair rate is r. The system is taken offline when it reaches state i.

supports graceful degradation in the image quality in the presence of multiple failures, the array must be taken offline when i disks fail within any group of N disks and the resulting image quality is too poor to be acceptable. Let f denote the failure rate of a single disk (i.e., $f = 1/MTTF$), and let r denote the rate of repair of a disk from tertiary storage (i.e., $r = 1/MTTR$). Then the IRAD architecture that tolerates $i - 1$ disk failures per cluster can be modeled as an $i + 1$ state Markov chain as shown in Figure 10. The mean time to shutdown can be either computed analytically using the theory of Markov chains [30], or computed numerically using tools such as SHARPE [27]. In the simplest case, where the array can tolerate two failures per cluster (i.e., $i = 3$), $MTTS = MTTF^3/[D(C-1)(C-2)MTTR^2]$. Thus, when $D = 32$, $N = C = 8$, and $MTTR = 3$ hours, the $MTTS$ is over 250 million years.

To summarize, the lazy/eager recovery based arrays and the IRAD architecture have a storage space overhead and MTTDL that is comparable to conventional arrays. However, they have a lower failure recovery overhead and a higher buffer requirement as compared to conventional arrays. Thus, our approaches trade buffer space for lower recovery overhead. Whereas lazy/eager recovery scheme may be chosen for perfect recovery of images, the IRAD architecture may be chosen for its advantages such as tolerance to multiple disk failures, resilience to network losses, etc. The choice of a particular recovery scheme depends on the application requirements.

5 Experimental Evaluation

5.1 LRJ/LRM algorithms and the IRAD Architecture

To evaluate the efficacy of our loss-resilient algorithms, as well as the IRAD architecture, we have developed prototype codecs for LRJ and LRM. We carried out several experiments using these prototype codecs. In the LRJ algorithm, when the information contained in a sub-image is not available, the quality of the reconstructed image is directly dependent on the amount of original image data available for reconstruction. Hence, increasing the value of the degree of image partitioning, N, improves the quality of the reconstructed images. However, with increase in N, the efficiency of the compression algorithm deteriorates. Figure 11 illustrates the visual quality of the reconstructed image for various values of N.

To quantitatively capture the improvement in the quality of the reconstructed images with increase in N, we have also computed the *Peak Signal to Noise Ratio (PSNR)* for all the images. For an $M \times N$ image of resolution r bits/pixel, if $p(x, y)$ and $p'(x, y)$ denote the pixel values at location (x, y) in the original and the reconstructed images, respectively, then the PSNR value can be defined as:

$$PSNR = 10 * \log \left(\frac{(2^r - 1)^2}{\sigma^2} \right) \text{ dB}$$

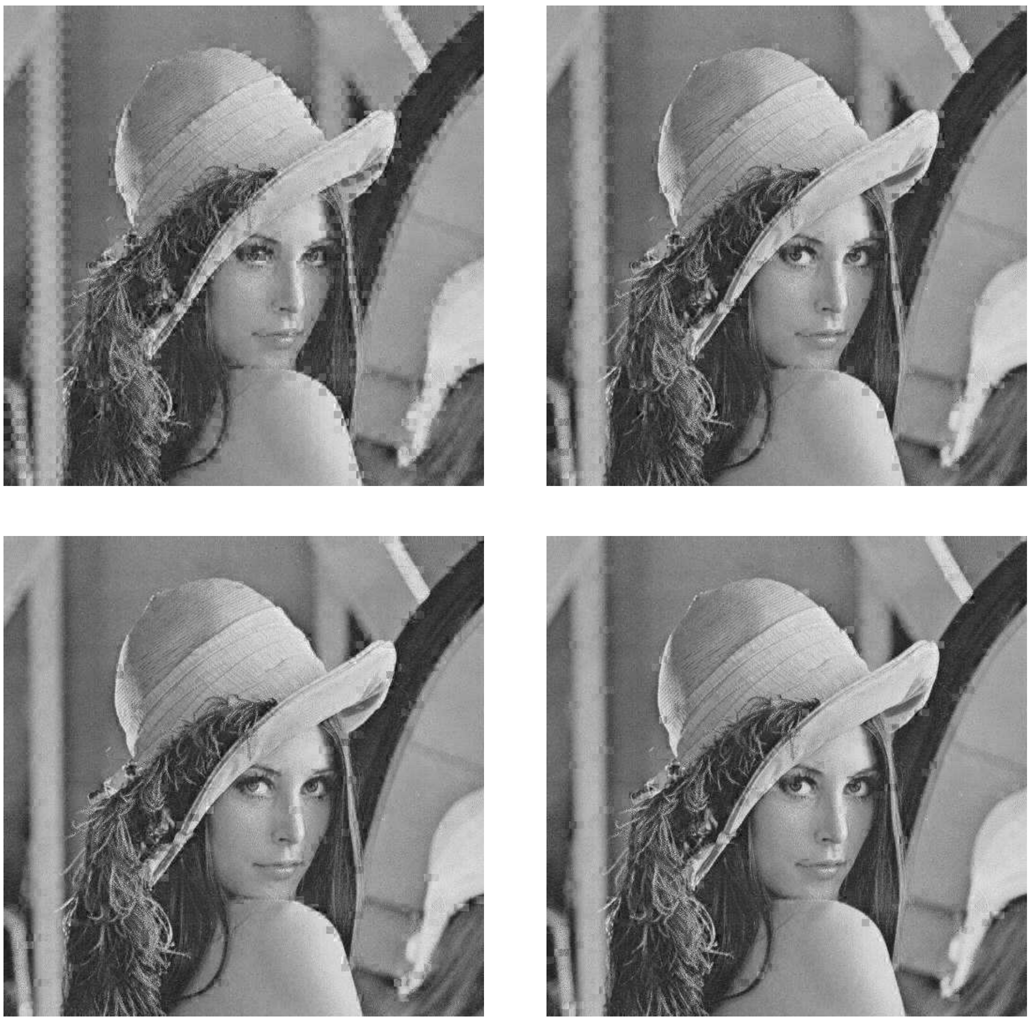

Figure 11 : Reconstructed image for $N = 4, 8, 12, 16$ with a single disk failure

where

$$\sigma = \sum_{x=1}^{M} \sum_{y=1}^{N} (p(x,y) - p'(x,y))^2$$

Figure 12(a) depict the variation in the PSNR value of the recovered image with increase in N for the LRJ algorithm. Figure 12(b), on the other hand, illustrates the degradation in compression efficiency (measured in terms of percentage increase in compressed image size) with increase in N. In practice, a server can choose an appropriate value of N depending upon the desired quality of the reconstructed image and the maximum tolerable degradation in compression efficiency. Our experiments indicate that $N = 8$ yields acceptable image quality, and results in an increase in compressed image size by about 6%.

Next, we conducted experiments to determine the efficacy of the LRM algorithm. The characteristics of the MPEG streams used in our experiments are shown in Table 4. Figure 13(a) depicts the picture quality (i.e., PSNR) for LRM streams obtained by varying the degree of image partitioning N, while Figure 13(b) shows the loss in compression efficiency due to image partitioning for these streams. Table 3 tabulates

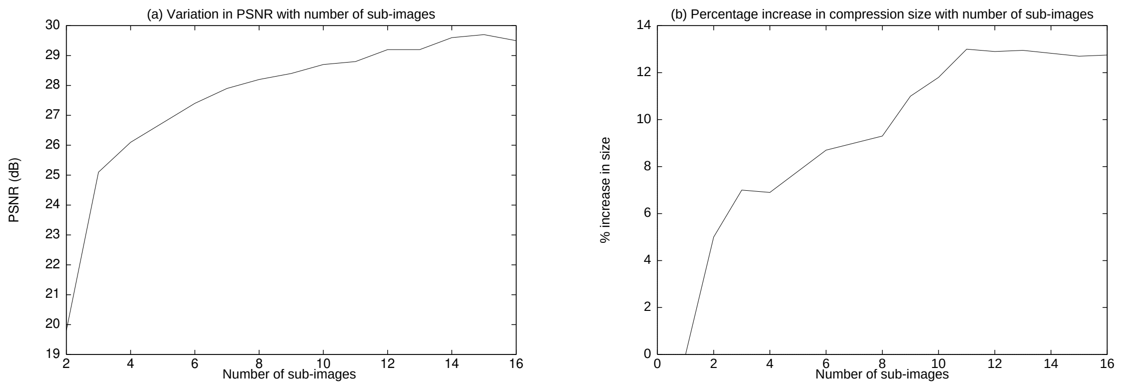

Figure 12 : Variation of PSNR and compression efficiency with number of sub-images in LRJ

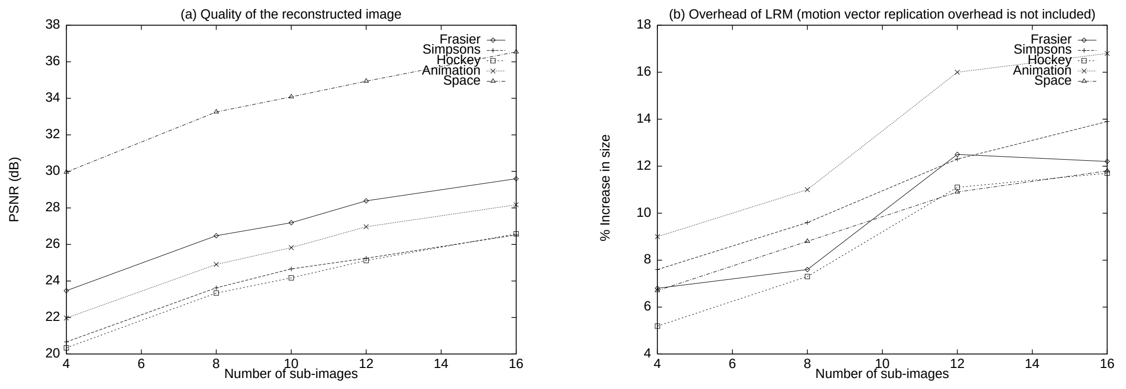

Figure 13 : Variation of PSNR and compression efficiency with number of sub-images for LRM

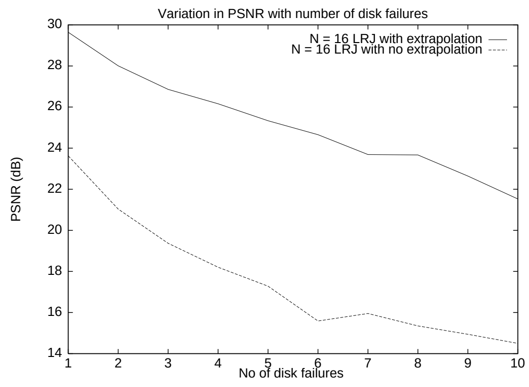

Figure 14 : Variation of quality of reconstructed image for multiple disk failures

Frasier	5.3%
Simpsons	3.35 %
Hockey	4.8%
Animation	8.08%
Space	2.39%

Table 3 : Overhead of Motion Vector Replication. The table shows the percentage increase in size of the original MPEG streams due to motion vector replication.

Table 4 : Characteristics of MPEG traces

File	Encoding	Pattern	Length (frames)	Average bit rate (Mb/s)	Motion
Frasier	MPEG	$I(BBP)^3BB$	6000	1.498	Moderate
Ice Hockey	MPEG	$I(BBP)^4BB$	750	1.53	High
Simpsons	MPEG	$I(BBP)^2BB$	720	0.8	High
Animation	MPEG	$I(B^9P)^3B^9$	1200	0.7	Moderate
Space	MPEG	$IBBPBB$	550	0.61	Low

the overhead of maintaining an additional copy of the motion vectors. The overhead of replicating motion vectors varied from 2% to 8%. The overhead of 8% was obtained for the animation sequence which had an abnormally large number of B frames as compared to I and P frames (see Table 4), and hence a larger number of motion vectors. However, for all other streams, the overhead was much lower with an average overhead of 4%. We observed reasonable recovery for $8 \leq N \leq 10$, with an 8% loss in compression efficiency. Thus, the total storage space overhead was around 12%.

Finally, to demonstrate that the IRAD architecture can tolerate multiple disk failures, we carried out several experiments. Figure 14 illustrates that the quality of the reconstructed image gradually deteriorates with increase in number of failed disks for the LRJ algorithm. It also demonstrates that the simple methods employed by the LRJ algorithm to extrapolate DC and AC coefficient values significantly improve the quality of the reconstructed image.

5.2 Parity-Based Failure Recovery

To evaluate the effectiveness of the lazy and the eager recovery schemes, we have developed an event-driven disk array simulator called *diskSim*. We carried out extensive trace-driven simulations to evaluate these failure recovery schemes. The simulation environment consisted of a disk array with 32 disks. The characteristics of each disk is shown in Table 5. The conventional SCAN disk scheduling algorithm is employed for retrieving media blocks from a disk during each round. Each VBR video stream stored on the array is assumed to be encoded using the MPEG compression algorithm. We used the MPEG streams shown in Table 4 for our experiments and simulations. Media blocks of a stream are assumed to be 64kB in size and are interleaved across the disks in array. The placement strategy ensures that all data blocks in

Disk capacity	2 GBytes
Number of disks in the array	32
Bytes per sector	512 KB
Sector per track	99
Tracks per cylinder	21
Cylinders per disk	2627
Minimum seek time	1.7 ms
Maximum seek time	22.5 ms
Maximum rotational latency	11.1 ms

Table 5 : Disk Parameters of Seagate-Elite3 disk

a parity group belong to the same media stream. The playback rate of each stream is assumed to be 30 frames/sec.

We compared the lazy and eager recovery schemes to the standard recovery scheme in a RAID level 5 array. Figure 15(a) depicts the total number of blocks retrieved by the entire array during each round normalized by the number of disks in the array. Recall from Section 3, that the lazy and the eager schemes impose a recovery overhead of $1/(G-1)$ on a RAID level 5 array. As illustrated by the figure, for $G = 2$ or mirroring, all schemes show a 100% increase in load, which is consistent with the analytical result. For larger parity group sizes, the recovery overhead decreases with increase in G for the lazy and eager recovery schemes. On the other hand, for standard RAID level 5, the increase in load in smaller than 100% for small values of G, $(G > 2)$. This is because, the number of blocks accessed by each client in a round approximately equals the parity group size. Since data blocks of the parity group requested for playback need not be accessed again for failure recovery, the number of additional blocks that must be accessed to reconstruct the lost block is smaller than the worst case value of $G - 1$. However, as the parity group size increases, the number of additional blocks that must be accessed to reconstruct the lost block increases and hence, the recovery overhead approaches 100%.

Figure 15(b) shows the total buffer requirements of different recovery schemes. The eager recovery scheme has the highest buffer requirement with the buffer increasing linearly with the parity group size. For small values of the parity group size (i.e., when $G - 2 \leq k^{\min}$), the lazy recovery approach has the same buffer requirements as the RAID level 5 array, consistent with the analytical result. However, as the parity group size increases, $(G - 2)$ becomes greater than $k^{\min}$, and hence, the buffer requirements of the lazy scheme become larger than that for RAID level 5. Thus, our simulation results validate the analytical results derived in Section 4. They also demonstrate that the lazy/eager recovery schemes trade buffer space for lower recovery overhead and hence, higher array utilization.

6 Concluding Remarks

In this paper, we have demonstrated the limitations of the conventional parity-based failure recovery algorithms for multimedia servers, and have presented two disk failure recovery methods that utilize the inherent characteristics of video streams to ensure that the user-invoked on-the-fly failure recovery process does not

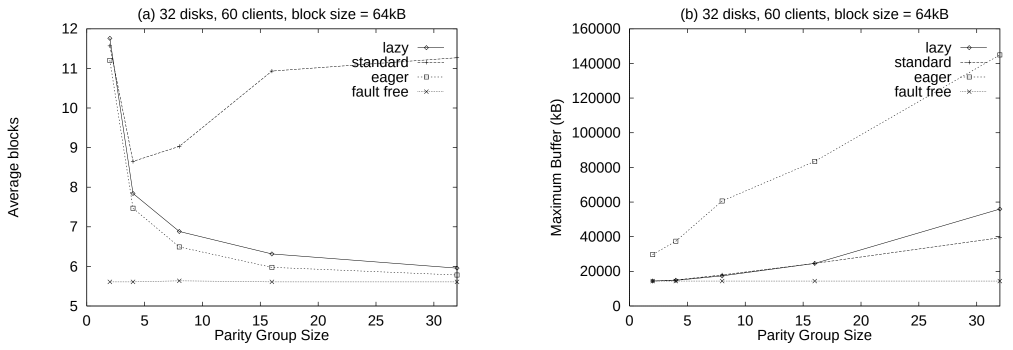

Figure 15 : Disk recovery and buffer space overhead for lazy, eager and standard RAID-5

impose any significant load on the disk array. Whereas the first approach utilizes the inherent redundancy in video streams (rather than error-correcting codes) to recover from disk failures in multimedia servers, the second exploits the sequential nature of playback of video streams to reduce the overhead of the recovery process,. We have demonstrated the efficacy of the former technique in the context of JPEG and MPEG compression algorithms. We have also shown that the latter technique significantly reduces the failure recovery overhead as compared to standard RAID arrays. The IRAD architecture that we have presented is an inherently distributed, scalable, end-to-end solution to failure recovery and supports supports graceful degradation in the quality of the reconstructed images with increase in the number of disk failures. These failure recovery algorithms are being incorporated into an integrated multimedia file server being built at the Distributed Multimedia Computing Laboratory, UT Austin.

REFERENCES

[1] S Berson, L Golubchik, and R R. Muntz. Fault Tolerant Design of Multimedia Servers. In *Proceedings of SIGMOD Conference*, pages 364–375, 1995.

[2] D. Bitton and J. Gray. Disk Shadowing. In *Proceedings of the 14th Conference on Very Large Databases*, pages 331–338, 1988.

[3] T. C. Bressoud and F. B. Schneider. Hypervisor-based Fault Tolerance. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles, Copper Mountain Resort, Colorado*, pages 1–11, Decmember 1995.

[4] P. Cao, S. B. Lim, S. Venkatraman, and J. Wilkes. The TickerTAIP parallel RAID architecture. In *Proceedings of the 1993 International Symposium on Computer Architecture*, pages 52–63, May 1993.

[5] M. S. Chen, H. I. Hsiao, C. S. Li, and P. S. Yu. Using Rotational Mirrored Declustering for Replica Placement in a Disk-array-based Video Server. In *Proceedings of the Third ACM Conference on Multimedia, San Francisco, California*, pages 121–130, November 1995.

[6] P. M. Chen and E. K. Lee. Striping in a RAID Level 5 Disk Array. In *Proceedings of the 1995 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1995.

[7] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys*, pages 145–185, June 1994.

[8] A. Cohen and W. A. Burkhard. Segmented Information Dispersal. Technical Report CS95-444, Department of Computer Science, University of California, San Diego, 1995.

[9] G. Copeland and T. Keller. A Comparison of High-Availability Media Recovery Techniques. In *Proceedings of the ACM Conference on Management of Data*, pages 98–109, 1989.

[10] J. M. Danskin, G. M. Davies, and X. Song. Fast Lossy Internet Image Transmission. In *Proceedings of the Third ACM Conference on Multimedia, San Francisco, California*, pages 321–332, November 1995.

[11] D. Le Gall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):47–58, April 1991.

[12] G Gibson and D Patterson. Designing Disk Arrays For High Data Reliability. *Journal of Parallel and Distributed Computing*, pages 4–27, January 1993.

[13] J. Gray, B. Horst, and M. Walker. Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput. In *Proceedings of the 16th Very Large Data Bases Conference*, pages 148–160, 1990.

[14] M. Holland and G. Gibson. Parity Declustering for Continuous Operation in Redundant Disk Arrays. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 23–35, October 1992.

[15] M. Holland, G. Gibson, and D. Siewiorek. Fast, On-line Recovery in Redundant Disk Arrays. In *Proceedings of the 23rd International Symposium on Fault Tolerant Computing*, pages 422–431, 1993.

[16] International Organization for Standardization (ISO). *Information technology – Coding of moving pictures and associated audio for digital storage media upto 1,5 Mbits/s, International Standard IS 11172 (MPEG)*, 1992.

[17] E.K. Lee and R. Katz. Performance Consequences of Parity Placement in Disk Arrays. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–199, 1991.

[18] J. Menon and J. Cortney. The Architecture of a Fault-Tolerant Cached RAID Controller. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 76–86, May 1993.

[19] A. Merchant and P. S. Yu. Design and Modeling of Clustered RAID. In *Proceedings of the International Symposium on Fault Tolerant Computing*, pages 140–149, 1992.

[20] A. Mourad. Reliable Disk Striping in Video-On-Demand Servers. Technical report, AT&T Bell Labs, 1995.

[21] R. R. Muntz and J. C.S. Lui. Performance Analysis of Disk Arrays Under Failure. In *Proceedings of the 16th Conference on Very Large Databases*, pages 162–173, 1990.

[22] M. N. Nelson, M. Linton, and S. Owicki. A Highly Available, Scalable ITV System. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles, Copper Mountain Resort, Colorado*, pages 54–67, Decmember 1995.

[23] B. Ozden, R. Rastogi, P. J. Shenoy, and A. Silberschatz. Fault-tolerant Architectures for Continuous Media Servers. In *Proceedings of the ACM SIGMOD, Montreal, Canada*, June 1996.

[24] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Array of Inexpensive Disks (RAID). *ACM SIGMOD'88*, pages 109–116, June 1988.

[25] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[26] E. J. Posnak, S. P. Gallindo, A. P. Stephens, and H. M. Vin. Techniques for Resilient Transmission of JPEG Video Streams. In *Proceedings of Multimedia Computing and Networking, San Jose, CA*, February 1995.

[27] R. Sahner and K. S. Trivedi. SHARPE: Symbolic Hierarchical Automated Reliability/Performance Evaluator, Introduction and Guide for Users. Technical report, Department of Computer Science, Duke University, September 1986.

[28] R. Tewari, D. Dias, R. Mukherjee, and H. M. Vin. High Availability for Clustered Multimedia Servers. In *Proceedings of International Conference on Data Engineering,New Orleans*, February 1996.

[29] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: A Disk Storage System for Video and Audio Files. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 393–400, August 1993.

[30] K. S. Trivedi. *Probability & Statistics With Reliability, Queuing, And Computer Science Applications*. Prentice-Hall, Inc., 1982.

[31] C.J. Turner and L.L. Peterson. Image Transfer: An End to End Design. In *Proceedings of ACM SIGCOMM'92, Baltimore*, pages 258–268, August 1992.

[32] H. M. Vin, P. J. Shenoy, and S. Rao. Efficient Failure Recovery in Multi-Disk Multimedia Servers. In *Proceedings of the 25th International Symposium on Fault Tolerant Computing Systems, Pasadena, CA*, pages 12–21, June 1995.

[33] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID Hierarchical Storage System. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles, Copper Mountain Resort, Colorado*, pages 96–108, Decmember 1995.

Prashant Shenoy is a doctoral candidate in the Department of Computer Sciences at the University Texas at Austin. His research topics include multimedia file systems and operating systems. He is involved in the design and implementation of an integrated file system being built at Distributed Multimedia Computing Laboratory, UT Austin.

Harrick M. Vin received his Ph.D. in Computer Science from the University of California at San Diego in 1993. He is currently an Assistant Professor of Computer Sciences, and the Director of the Distributed Multimedia Computing Laboratory at the University of Texas at Austin. His research interests are in the areas of multimedia systems, high-speed networking, mobile computing, and large-scale distributed systems. Over the past 5 years, he has co-authored more than 55 papers in leading journals and conferences in the area of multimedia systems. He has been a recipient of several awards including the National Science Foundation CAREER award, IBM Faculty Development Award, AT&T Foundation Award, IBM Doctoral Fellowship, NCR Innovation Award, and the San Diego Supercomputer Center Creative Computing Award.