Efficient Failure Recovery in Multi-Disk Multimedia Servers

Harrick M. Vin, **Prashant J. Shenoy** and **Sriram Rao**

Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188, USA
E-mail: {vin,shenoy,sriram}@cs.utexas.edu, Telephone: (512) 471-9732, Fax: (512) 471-8885

Abstract

In this paper, we present a novel disk failure recovery method that utilizes the inherent redundancy in video streams (rather than error-correcting codes) to ensure that the user-invoked on-the-fly failure recovery process does not impose any additional load on the disk array. We also present a disk array architecture that enhances 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.

1 Introduction

Recent advances in computer and communications technologies promise to lay a foundation for designing sophisticated multimedia information services in a wide range of application domains. The realization of such services, however, requires the development of high performance, scalable multimedia servers which can provide services to clients over high speed networks.

Due to the immensity of the sizes and the data rate requirements of multimedia objects, such multimedia servers will undeniably be founded on large *disk arrays*. Whereas several research groups have recently investigated techniques for ensuring continuous recording and retrieval of digital audio and video streams from a disk array [4], the problem of meeting the real-time requirements imposed by these operations in the presence of disk failure have not received much attention. A scalable 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. Algorithms for efficient failure recovery in multi-disk multimedia servers is the subject matter of this paper.

1.1 Relation to Previous Work

During the past few years, we have seen a dramatic increase in the number of academic and industrial research projects investigating the design of fault-tolerant storage systems [2, 10]. In most of these systems, fault-tolerance is achieved either by *disk mirroring* [1] or *parity encoding* [5, 13]. Disk mirroring achieves fault-tolerance by duplicating data on separate disks (and thereby incurs 100% storage space overhead). Parity encoding, on the other hand, reduces the overhead considerably by employing error correcting codes. For instance, in a RAID level 5 disk array consisting of D disks, parity computed over data stored across $(D-1)$ disks is stored on another disk (e.g., the left-symmetric parity assignment shown in Figure 1(a)) [6, 9, 13]. In such architectures, if one of the disks fail, the data on the failed disk is recovered by taking an exclusive-or operation on the data and parity blocks stored on the surviving disks. That is, each user access to a block on the failed disk causes one request to be sent to each of the surviving disks. Thus, if the system is load balanced prior to disk failure, the surviving disks would observe at least twice as many requests in the presence of a failure [8].

The *declustered parity* disk array organization [7, 11, 12] addresses this problem by trading some of the array's capacity for improved performance in the presence of disk failures. Specifically, it requires that each parity block protect some smaller number of data blocks (say $(G-1)$). By appropriately distributing the parity information across all the D disks in the array, such a policy ensures that each surviving disk would see an on-the-fly reconstruction load increase of $(G-1)/(D-1)$ instead of $(D-1)/(D-1) = 100\%$ (see Figure 1(b)).

In summary, several research groups have recently developed techniques for designing fault-tolerant disk arrays [3]. However, most of the analytical as well as simulation models developed for failure recovery analysis have presumed conventional workload. That is, the load is assumed to predominantly consist of small read and write operations, that are aperiodic, independent of each other, and do not

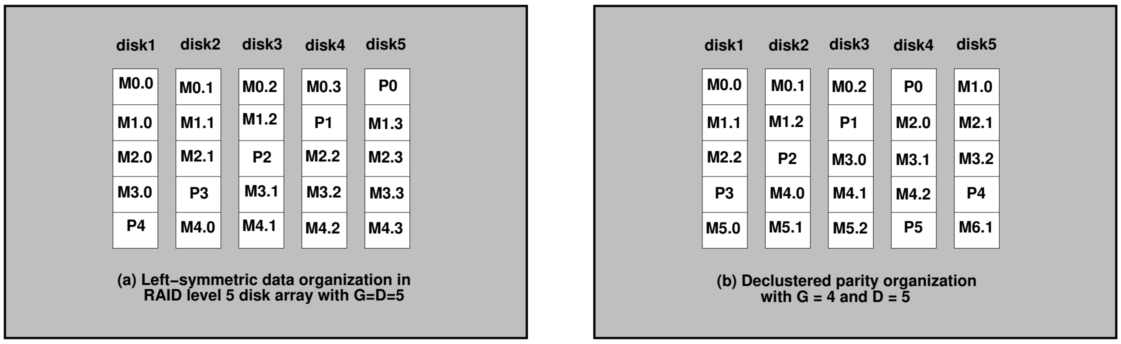

Figure 1 : Left-symmetric and declustered parity organizations for RAID architecture. $M_{i.j}$ and P_i denote data and parity blocks, respectively, and $P_i = M_{i.0} \oplus M_{i.1} \cdots \oplus M_{i.(G-2)}$

have any real-time constraints. On the contrary, retrieval of digital audio and video streams impose real-time constraints, the data accesses are sequential as well as periodic, and the data streams contain inherent redundancy. Hence, direct application of conventional techniques for recovering data from a failed disk in an array may impose higher overhead than necessary, and hence may require the server to operate at lower utilization levels during the fault-free state.

1.2 Research Contributions of This Paper

In this paper, we present a novel failure recovery method that utilizes the inherent redundancy in video streams (rather than error-correcting codes) to recover from disk failures in multimedia servers. Since human perception is tolerant to minor distortions in video playback, rather than perfectly recovering images stored on the failed disk, our method approximates the lost image data by exploiting *spatial* and *temporal* redundancies inherent in video streams. We illustrate our method by presenting: (1) a *Self-Recovering JPEG (SRJ)* compression algorithm which partitions each image into several sub-images such that a reasonable approximation of an image can be reconstructed even when one or more of its sub-images are not available; and (2) a *Self-Recovering Array of Disks (SRAD)* architecture that combines the SRJ compression algorithm 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 disk array. Together, they enhance the scalability of multimedia servers by integrating the recovery process with the decompression of video streams, and thereby distributing the reconstruction process across the clients, and by supporting graceful degradation in the quality of recovered images with increase in the number of disk failures.

We have evaluated the effectiveness of the SRJ compression algorithm as well as the SRAD architecture through extensive experimentation. We present and analyze our results.

The rest of the paper is organized as follows: The fault-free operation of a multi-disk multimedia server as well as simple methods of utilizing parity encoding techniques for failure recovery in multimedia servers are described in Section 2. The SRJ compression algorithm and the SRAD architecture are presented in Section 3. Section 4 describes our experiments, and finally, Section 5 summarizes our results.

2 Multi-Disk Multimedia Servers

Digitization of audio yields a sequence of samples and that of video yields a sequence of images. We refer to a continuously recorded sequence of images or audio samples as a *media stream*. To effectively utilize a disk array, a multimedia server must interleave the storage of each media stream among the disks in the array. The unit of data interleaving, referred to as a *media block*, denotes the maximum amount of logically contiguous data that is stored on a single disk. Successive media blocks of a stream are placed on consecutive disks using a round-robin allocation algorithm. *Parity blocks* are derived by taking an exclusive-or operation of a sequence of media blocks of a stream. The parity block together with all the media blocks over which the parity is computed are referred to as a *parity group*.

To precisely describe the fault-free operation, consider a multimedia server that is servicing n clients, each retrieving a video stream (say $S_1, S_2, ..., S_n$, respectively). Let $\mathcal{R}^i_{pl}$, $\forall i \in [1, n]$, denote the playback rate (expressed in images/sec) of video stream S_i. Due to the periodic nature of media playback, a multimedia server can service 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:

$$f_i = \mathcal{T} * \mathcal{R}_{pl}^i; \quad \forall i \in [1, n]$$

Since the server maintains each video stream in its compressed format, accessing $f_1, f_2, ..., f_n$ images will require the server to retrieve $k_1, k_2, ..., k_n$ media blocks of streams $S_1, S_2, ..., S_n$, respectively. If a media stream is encoded using a variable bit rate (VBR) compression algorithm, then the sizes of images may vary. Hence, the mapping from f_i to k_i may vary from one round to another. For media streams encoded using Constant bit rate (CBR) compression algorithms, on the other end, the mapping from f_i to k_i is fixed. Regardless of the 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, however, recovering images stored on the failed disk will require the server to retrieve additional media and parity blocks from some of the surviving disks. In the simplest case, on accessing a block from the failed disk during a round, a server may retrieve all the blocks within its parity group, recover the lost information, transmit the set of requested blocks during the round to the client site, and then discard the additional blocks. Although relatively straightforward 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.

A server may reduce the increase in load on surviving disks by exploiting the sequentiality of video playback. To illustrate, consider a disk array with a parity group size of G. Denote the set of disks within a parity group as $1, 2, ..., G$, and assume that, at the beginning of round r, disk i ($i \leq G$) fails. Let the number of blocks to be accessed from disk i during round r be denoted by n_i^r. In such a scenario, to recover each of the n_i^r blocks, the server will, in the worst case, access an additional block from each of the surviving $(G - 1)$ disks in the array during round r. Notice, however, that due to the sequentiality of video playback, a subset of the $(G - 1)$ blocks accessed for recovering a block will be requested by the client within the next few rounds. Hence, the server can buffer such blocks and transmit them to client sites during appropriate rounds. Consequently, an increase in the workload on the disks during round r due to a client will be followed by a corresponding reduction in the load due to the client during successive rounds. In fact, if the server can maintain cumulative parity information for the blocks being transmitted to client sites, then over a sequence of rounds, the server will access exactly one additional block (namely, the parity block) for each client from the disk array (yielding an average increase in load of $1/(D - 1)$). Since this algorithm recovers media blocks stored on failed disk in the same round in which they are accessed, we refer to it as an *aggressive* failure recovery algorithm. On the other hand, the server may employ a *conservative* failure recovery algorithm in which the playback of video streams is suspended until blocks constituting at least one parity group have been read-ahead and buffered at the server. This will replace a sequence of playback discontinuities incurred by the aggressive algorithm with a one-time pause in video playback.

The fundamental limitation of both of these algorithms is that, due to the large playback rate requirements of video streams, their additional buffer space requirement can be prohibitively large. Moreover, since the process of recovering blocks stored on the failed disk is performed centrally at the disk array controller, the overhead of the recovery process itself increases linearly with the number of clients. Consequently, such algorithms do not scale very well. To address these limitations, we now propose an algorithm that minimizes the overhead of failure recovery by exploiting the inherent redundancy in video streams.

3 Exploiting Inherent Redundancy of Video Streams

In a disk-array-based multimedia server, since each media block contains one or more images and since several blocks of a stream are stored on each disk, a single disk failure will result in the loss of several image sequences. If, on the other hand, each image is partitioned into sub-images and if the sub-images are stored on different disks, then a single disk failure will result in the loss of fractions of several images. In the simplest case, if the sub-images are created in the pixel domain (i.e., prior to compression) 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.

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 [15, 16]. The resultant increase in the bit-rate requirement may impose higher load on each disk in the array even during the fault-free state, thereby reducing the number of video streams that can be simultaneously retrieved from the server. In what follows, we first present a self-recovering JPEG (SRJ) compression algorithm that addresses the lim-

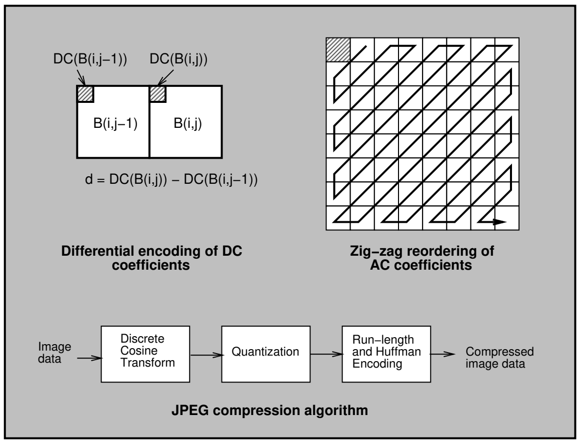

Figure 2 : JPEG compression algorithm

itations of the pre-compression image partitioning technique, and then integrate it with placement techniques for disk arrays to derive a self-recovering disk array architecture for video streams.

3.1 Self-Recovering JPEG (SRJ) Algorithm

Since human perception is less sensitive to high frequency components of a video signal, most compression algorithms transform video signals into the frequency domain so as to separate low and high frequency components. For instance, the JPEG compression standard fragments image data into a sequence of 8x8 pixel blocks which are separately transformed into the frequency domain using discrete cosine transform (DCT). DCT uncorrelates each pixel block into an 8x8 array of coefficients such that most of the spectral energy is packed in the fewest number of low frequency coefficients. Whereas the lowest frequency coefficient (referred to as the *DC coefficient*) captures the average brightness of the spatial block, the remaining set of 63 coefficients (referred to as the *AC coefficients*) capture the details within the 8x8 pixel block. The DC coefficients of successive blocks are difference encoded 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 2 depicts the main steps involved in the JPEG compression algorithm [14].

The self-recovering JPEG (SRJ) algorithm, that we present in this section, is an enhancement of the JPEG compression standard, and is motivated by the following two observations:

- Since the DC coefficients capture the average brightness of each 8x8 pixel block and since the average brightness of pixels gradually changes across most images, the DC coefficients of neighboring 8x8 pixel blocks are correlated. Consequently, the value of DC coefficient of a block can be reasonably approximated by extrapolating from the DC coefficients of the neighboring blocks.

  To formally capture this observation, consider an image containing $\text{NROW} \times \text{NCOL}$ of 8x8 pixel blocks. Let us define the *8-neighborhood* of a block at location (x, y) (denoted by $B(x, y)$) as the set:

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

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

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

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

- Due to the very nature of DCT, the set of AC coefficients yielded for each 8x8 block are uncorrelated. Moreover, since DCT packs the most amount of spectral energy into a few low frequency coefficients, quantizing the 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).

Thus, even when parts of a compressed image have been lost, a reasonable recovery is possible if: (1) the image in the frequency domain is partitioned into a set of sub-images such that none of the DC coefficients in the 8-neighborhood of a block belong to the same sub-image, and (2) the AC coefficients of a block are scattered amongst multiple sub-images. Note that this is distinct from the pre-compression image partitioning technique 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 SRJ compression algorithm that enables reasonable failure recovery without incurring any significant degradation in compression efficiency. To clearly define the SRJ compression algorithm, we will first determine the degree of image partitioning (i.e., the number of sub-images that must be created so as to satisfy the above requirement), and then define a method for scattering the AC coefficients.

3.1.1 Determining the Degree of Image Partitioning

The minimum value of the degree of image partitioning can be determined using 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 4 sub-images.*

Proof: The necessity and the sufficiency of this condition can be derived by construction [17]. ■

Notice that when an image is partitioned into 4 sub-images, each sub-image contains 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 directly 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 the blocks assigned to the same sub-image, and thereby deteriorates the 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.

3.1.2 Scrambling AC Coefficients

To enable better recovery, the AC coefficients of a block should be scattered amongst multiple sub-images. To achieve this objective, the SRJ compression algorithm employs a scrambling technique which when given a set of N blocks of AC coefficients, creates a new set of N blocks such that the AC coefficients from each of the input blocks are equally distributed amongst all of the output blocks.

To precisely describe the scrambling technique, let us denote the set of N blocks of the original image as O_i, $i \in [0, N-1]$, and the new set of N blocks created as $\widehat{O_i}$. Assuming that the AC coefficients are numbered from left-to-right in a row-major order and that $\mathrm{AC}^k_{O_i}$ denotes the k^{th} AC coefficient ($k \in [1, 63]$) of block O_i, then the scrambling operation assigns $\mathrm{AC}^k_{O_i}$ to be the k^{th} coefficient of block $\widehat{O_j}$ where $j = (i + k) \bmod N$. Thus, each resulting block contains exactly $\frac{64}{N}$ coefficients of each of the original blocks. Specifically, one of the blocks contain the DC coefficient and $\left(\frac{64}{N} - 1\right)$ AC coefficients, and all the remaining $(N-1)$ blocks contain $\left(\frac{64}{N}\right)$ AC coefficients. Figure 3 illustrates the operation of scrambling AC coefficients of four 4x4 blocks.

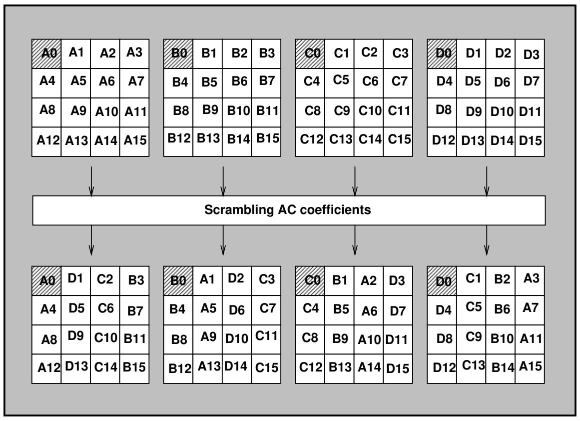

Figure 3 : Scrambling AC coefficients. Here A_0, B_0, C_0 and D_0 denote DC coefficients, and $\forall i \in [1, 15]$: A_i, B_i, C_i and D_i represent AC coefficients.

3.1.3 Combining Image Partitioning and Scrambling Techniques

Given that each image in a video stream is being partitioned into $N(N \geq 4)$ sub-images, the SRJ compression algorithm involves two steps: (1) select a set of N blocks from the original image, and scramble the the set of AC coefficients within the blocks to create a new set of N blocks; and (2) assign the resulting blocks 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 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.

Notice that since each invokation of the scrambling technique requires N blocks from the same row of the image, the number of blocks within each row of the original image must be an integral multiple of N. In the event that this condition is not met for the original image, each row of blocks may required to be padded with additional "zero" blocks (i.e., blocks with the DC as well as all of the AC coefficients set to zero). Since a sequence of zeroes can be efficiently run-length and huffman encoded, the addition of such zero blocks does not yield any noticeable increase in the size of the compressed image.

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 loseless DPCM scheme using the DC coefficient from the previous block 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, to guarantee the availability of the huffman table even when one or more of the sub-images are not available, it must be replicated across multiple sub-images.

At the time of decompression, once each sub-image has been run-length and huffman decoded, the SRJ algorithm employs an unscrambler to recover blocks of the original image from the corresponding blocks of the sub-images. In the event that the information contained in a sub-image is not available, the unscrambling module 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 scrambler module employed by the encoder ensures that each block within a sub-image contains coefficients from several blocks of the original image, the artifacts yielded by such a recovery mechanism are dispersed over the entire reconstructed image, thereby significantly improving the visual quality of the image.

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). The effect of scattering 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, a large fraction of the quantized AC coefficients are zeroes. Since the scattering algorithm does not alter the relative position of an AC coefficient within the zig-zag ordering, the effect of scattering on the efficiency of run-length and huffman encoding is minimal. Thus, the increase in compressed image size yielded by the SRJ algorithm can be mostly attributed to the need for replicating huffman tables and the uncorrelation of successive DC coefficients.

3.2 Self-Recovering Array of Disks (SRAD)

The SRJ compression algorithm, when applied to a sequence of images constituting a video stream yields N sequences of sub-images. We refer to each such sequence as *sub-stream*. To ensure effective recovery, the server must organize each of the sub-streams on the array such 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). We refer to a disk array architecture that employs such placement methodologies as a *Self-Recovering Array of Disks (SRAD)*. A careful analysis of this process of recovering from disk failure illustrates the following salient characteristics of the SRAD 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.
- 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 technology — 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., SRAD supports graceful degradation in the image quality with increase in the number of failed disks).

Observe also that although the quality of the recovered image in the presence of a single disk failure is acceptable for most applications, to prevent any accumulation of errors across multiple disk failures, the server must also maintain parity information to perfectly recover the contents of the failed disk onto a spare disk. In such a scenario, on-line reconstruction onto a spare disk can proceed simply by issuing low-priority read requests to access media blocks from each of the surviving disks [8]. By assigning low priority to each read request issued by the on-line reconstruction process, the server can ensure that the performance guarantees provided to all the clients are met even in the presence of disk failures.

4 Experimental Evaluation

To evaluate the effectiveness of the SRJ compression algorithm and the SRAD architecture, we have carried out extensive trace-driven simulations in an environment consisting of an asynchronous disk array with 16 disks. Each media stream is assumed to be interleaved across the entire array. The characteristics of each disk are shown in Table

Disk capacity	2 GBytes
Number of disks in the array	16
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 1 : Disk Parameters of Seagate-Elite3 disk used in the paper

1. Clients are assumed to arrive at the beginning of the simulation and remain in the system till the end of the simulation. The video stream accessed by clients are assumed to be independent and encoded using a Variable Bit Rate (VBR) compression technique. The conventional SCAN disk scheduling algorithm is employed for retrieving media blocks from a disk during each round. The playback rate of each video stream is assumed to be 30 images/sec with an average data rate requirement of 8 Mb/sec.

4.1 Parity-Based Failure Recovery

As we had pointed out in Section 2, the two of the limitations of parity-based failure recovery algorithms include: (1) the increase in the load on the surviving disks induced by the user-invoked on-the-fly recovery process, and (2) the higher buffer space requirement. Figure 4(a) depicts the increase in load on the surviving disks yielded by the naive RAID implementation (in which the set of blocks accessed from the parity group for reconstruction are discarded immediately after recovering the desired block) and the aggressive failure recovery algorithm. It demonstrates that both of these algorithms induce a large increase in load (nearly 100%) during rounds immediately following the disk failure. Whereas the naive implementation sustains the increased load for all the succeeding rounds, the average increase in the load on surviving disks yielded by the aggressive algorithm is about 6%.

The buffer space requirement of the aggressive failure recovery algorithm, on the other hand, is dependent on the relative values of the parity group size and the average number of disks accessed for each client during a round. Figure 4(b) depicts the variation in the buffer space requirement with increase in parity group size, assuming that, prior to failure, each client accessed two disks on an average during each round.

4.2 SRJ algorithm and SRAD Architecture

Figure 5 describes the SRJ compression and the decompression algorithm in detail. As per the 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 6 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:

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

where

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

Figure 7(a) depict the variation in the PSNR value of the recovered image with increase in N. Figure 7(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%.

Figure 8 illustrates that the quality of the reconstructed image gradually deteriorates with increase in number of failed disks. It also demonstrates that the simple methods employed by the SRJ algorithm to extrapolate DC and AC coefficient values significantly improve the quality of the reconstructed image.

5 Concluding Remarks

In this paper, we have demonstrated the limitations of the conventional parity-based failure recovery algorithms, and have presented a novel failure recovery method that utilizes

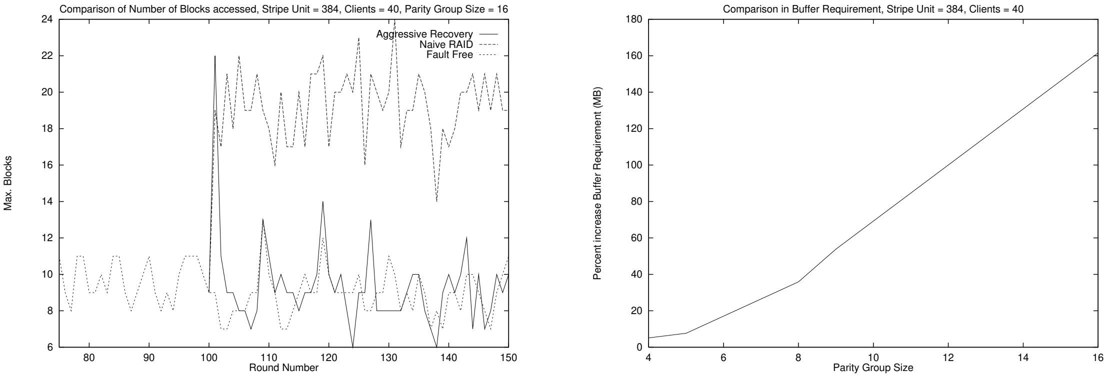

Figure 4 : (a) Increase in load on disks yielded by the parity-based failure recovery algorithm (a disk is assumed to fail at the beginning of round 100); (b) Increase in buffer space requirement of the aggressive failure recovery algorithm

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

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

Figure 5 : SRJ compression and decompression algorithms

Figure 6 : Reconstructed image for $N = 4, 8, 12, 16$ with a single disk failure

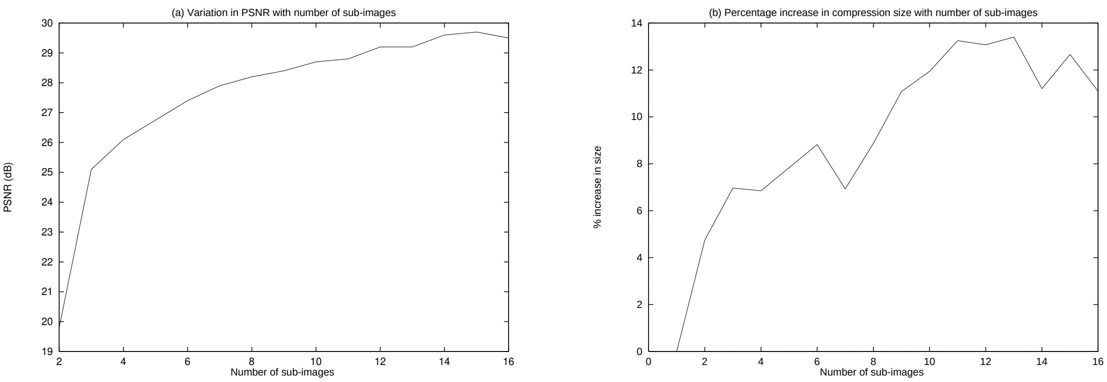

Figure 7 : Variation of PSNR and the degradation in compression with number of sub-images

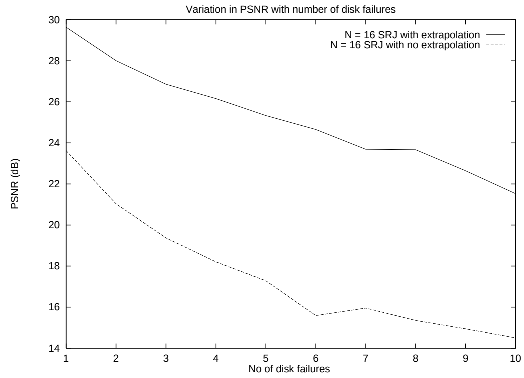

Figure 8 : Variation of quality of reconstructed image for multiple disk failures

the inherent redundancy in video streams (rather than error-correcting codes) to recover from disk failures in multimedia servers. We have demonstrated that our architecture is inherently distributed, scalable, and supports graceful degradation in the quality of the reconstructed images with increase in the number of disk failures. Although presented within the framework of the JPEG compression standard, the concepts presented in this paper can be easily extended to other compression techniques.

REFERENCES

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

[2] 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.

[3] 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*, 1994.

[4] J. Gemmell, H. M. Vin, D. D. Kandlur, and P. V. Rangan. Multimedia Storage Servers: A Tutorial and Survey. *IEEE Computer*, 1995.

[5] G. Gibson and D. Patterson. Designing Disk Arrays for High Data Reliability. *Journal of Parallel and Distributed Computing*, January 1993.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] H. M. Vin, P. J. Shenoy, and S. Rao. Efficient Failure Recovery in Multi-Disk Multimedia Servers. Technical Report TR-95-15, Department of Computer Sciences, Univ. of Texas at Austin, 1995.