Efficient failure recovery in multi-disk multimedia servers

Harrick M. Vin, **Prashant J. Shenoy**, and **Sriram S. Rao**
Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188

E-mail: {vin,shenoy,sriram}@cs.utexas.edu, Telephone: (512) 471-9732, Fax: (512) 471-8885

ABSTRACT

In this paper, we present 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 impose any significant load on the disk array. Whereas the first approach utilizes the sequential nature of playback of video streams to reduce the overhead of the on-the-fly recovery process, the second exploits the inherent redundancies present in video streams to facilitate efficient failure recovery. For the latter approach, 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. We compare and contrast our methods to the conventional disk failure recovery schemes.

1 INTRODUCTION

Recent advances in computer and communications technologies have made it economically viable to design distributed multimedia information systems that promise to enhance users' ability to access a wealth of audio, video, and textual information. 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,[3] 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. Since video is the most demanding data type (with respect to data rate and real-time performance requirements), in this paper, we present failure recovery techniques tailored for video sequences. However, these techniques can be easily extended to other data types such as images and animation sequences.

Traditionally, the techniques for recovering from disk failure utilize either *mirroring*[2] or *parity encoding*.[4,12] Mirroring techniques, employed at either the block or the disk level, achieve fault-tolerance by duplicating data on separate disks (and thereby incur a 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 is computed over data stored across $(D-1)$ disks and stored on another disk (e.g., the left-symmetric parity assignment shown in Figure 1(a)).[5,8,12] In such architectures, if one of the disks fail, the data on the failed disk is recovered by taking an exclusive-or of the data and parity blocks stored on the surviving disks. Hence, each user access to a block on the failed disk causes one

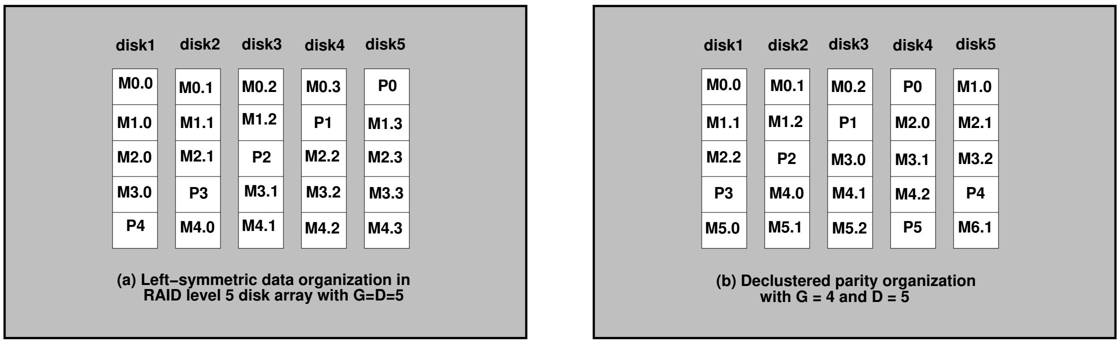

Figure 1: Left-symmetric and declustered parity organizations for RAID architecture. $M_{i.j}$ and P_i denote data and parity blocks, respectively, and $P_i = M_{i.0} \oplus M_{i.1} \cdots \oplus M_{i.(G-2)}$

request to be sent to each of the surviving disks. Thus, if the system is load balanced prior to disk failure, the surviving disks observe at least twice as many requests in the presence of a failure.[7]

The *declustered parity* disk array organization[6,9,11] 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)).

From the perspective of designing large-scale video servers, a fundamental limitation of most of the analytical and simulation models developed for failure recovery analysis is that they presume 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 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.

Recently, several research groups have begun developing failure recovery techniques that are optimized for video servers.[1,10,15,16] The *Streaming RAID* server,[16] for example, employs a RAID level 3 disk array partitions disks into fixed size groups called *disk clusters*. Blocks from a parity group confined to disks within a cluster. Even in the absence of any disk failure, the server reads entire parity groups (including the parity blocks) in response to user requests. This ensures that no additional load is imposed on the disks in the presence of a disk failure, albeit at the expense of higher buffer requirement and load during the fault-free state. Berson et al.[1] have addressed these limitations by using a RAID level 5 disk array with non-clustered parity which: (1) distributes parity blocks from a cluster among disks in other clusters, (2) dynamically modifies the retrieval schedule in the event of a disk failure, and (3) reads entire parity groups at a time only if a block being accessed is on a faulty disk. However, the overhead of such additional block accesses on the surviving disks may violate the real-time performance requirements of some user requests.

In this paper, we present two failure recovery methods that utilize the inherent characteristics of video streams to minimize the overhead of on-the-fly disk failure recovery. In the first method, we exploit the sequential nature of video stream access to reduce the overhead of on-line recovery in a RAID level 5 disk array. Specifically, by requiring that parity blocks are 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 the requests for their access from the buffer, this method minimizes the overhead of the on-line failure recovery process.

The second method, on the other hand, exploits human perceptual tolerance and utilizes the inherent redundancy in video streams (rather than error correcting codes) for efficient failure recovery. Specifically, since human perception is tolerant to minor distortions in video playback, rather than perfectly recovering images stored on the failed disk, this 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.

The rest of the paper is organized as follows: Our scheme for exploiting the sequentiality of video retrieval to optimize failure recovery are described in Section 2. The SRJ compression algorithm and the SRAD disk array architecture are presented in Section 3. Finally, Section 4 summarizes our results.

2 EXPLOITING SEQUENTIALITY OF VIDEO RETRIEVAL

Consider a video server that interleaves the storage of each video stream (consisting of sequence of images) among disks in an array. Let the unit of data interleaving, referred to as a *media block*, denote the maximum amount of logically contiguous data that is stored on a single disk. Moreover, let successive media blocks of a stream be placed on consecutive disks using a round-robin allocation algorithm.[3] In addition to a sequence of media blocks for each video stream, to recover from disk failures, let the server maintain parity blocks on the array. If the parity group size is denoted by G, then, to exploit the sequentiality of recording and playback operations, let us assume that the server computes parity blocks for each sequence of $(G-1)$ media blocks of a video stream, and that the storage of such parity blocks is interleaved with media blocks on the disk array.

To precisely 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 $\mathcal{R}_{pl}^i,\ \forall i \in [1,n]$, 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 must equal its playback rate and is given by: $f_i = \mathcal{T} * \mathcal{R}_{pl}^i; \quad \forall i \in [1,n]$. Depending on the relationship between image and block sizes, 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. In fact, if a media stream is encoded using a variable bit rate (VBR) compression algorithm, then due to variability in image sizes, the value of k_i may change from one round to another. 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 all of the media and parity blocks within the corresponding parity group to recover the lost information. 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 lost information, transmit the set of requested blocks during the round to the client site, and then discard 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.

Observe that since the playback is sequential, a server can minimize the increase in load on surviving disks yielded by the on-the-fly failure recovery operation by reusing media blocks retrieved during playback for failure recovery and vice-versa. To illustrate, consider a disk array with a parity group size of G. Let the set of disks within a parity group be denoted

Procedure RetrieveInPresenceOfFailures
begin
for each client j do {`* all iterations of the for loop are executed concurrently *`}
(1) Suspend transmission of blocks, and prefetch $(k_j^{\max} + G - 2)$ media blocks
(2) If current playback position 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 $m - 1$.
end if.
(3) Resume transmission of media blocks to the user.
(4) If in a round, k_j blocks are transmitted to the user then
Retrieve 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 2: Conservative algorithm for failure recovery

as $1, 2, ..., G$, and let us assume that, at the beginning of round r, disk $i (i \leq G)$ fails. In such a scenario, for each block accessed from disk i during round r, the server will, in the worst case, access an additional block from each of the surviving $(G - 1)$ disks in the parity group during round r. Due to sequentiality of video playback, however, a subset of the $(G - 1)$ blocks (namely, blocks from disks $(i + 1)$ through G) accessed for recovering a block will be requested within the next few rounds. The server can buffer these 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 that client in the next few rounds. Similarly, the server can reduce the overhead of failure recovery by maintaining the result of an exclusive-or of the blocks within a parity group already transmitted to the client. With this, to recover a block stored on disk i, the server will be required to retrieve additional blocks only from disks $(i + 1)$ through G (rather than the entire parity group). Since this algorithm recovers media blocks stored on the failed disk in the same round in which they are accessed, we refer to it as an *aggressive* failure recovery algorithm.

Similarly, the server may also employ a *conservative* failure recovery algorithm in which the transmission of video streams to users is suspended until the server prefetches sufficient media blocks to satisfy even the worst case user request. Specifically, if $k_j^{\max}$ and k_j^{avg} denote the maximum and the average number of media blocks requested in a round by client j, and if the i^{th} disk $(i \leq G)$ of the parity group fails at the beginning of 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 $\left\lceil \frac{k_j^{\max} + G - 2}{k_j^{avg}} \right\rceil$ rounds. On resuming transmission (and hence, playback), the server must retrieve sufficient number of blocks in each round so as to ensure that at least $(k_j^{\max} + G - 2)$ blocks of each client are always in buffer. This guarantees that the server will always be able to reconstruct a block requested from a failed disk from those available in the buffer (i.e., without retrieving any additional blocks). The precise failure recovery algorithm as well as the flow of media blocks prior to as well as in the presence of disk failure are presented in Figures 2 and 3, respectively.

Notice that, as a result of exploiting sequentiality of video playback, over a sequence of rounds, the only additional load imposed by both of these algorithms on the server is due to the retrieval of parity blocks. Consequently, if the server employs a clustered parity placement policy, then the average increase in the load on the disks constituting the parity group

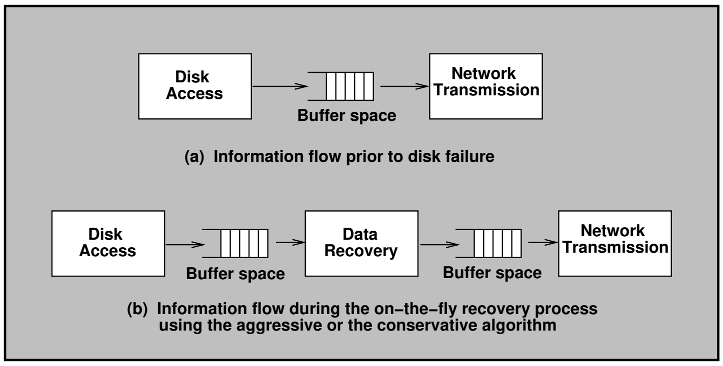

Figure 3: Information flow in a multimedia server prior to as well as in the presence of a disk failure

is $1/(G-1)$. If, on the other hand, the server employs a flat parity placement scheme, in which all data blocks are clustered and parity blocks from a cluster are uniformly distributed among disks in other clusters, then the average increase in the load on surviving disks is $1/(D-(G-1))$. The fundamental difference between the aggressive and the conservative algorithm is that the latter replaces a sequence of playback discontinuities possible in the former by a one-time pause. In fact, it may be possible for a video server to dynamically decide between the two approaches on a disk failure. Specifically, if the array utilization is low and can sustain a large increase in the load without saturating, then the server may choose to employ the aggressive algorithm. On the other hand, if the utilization of the array is high when a failure occurs, the server may employ the conservative algorithm to prefetch and buffer data.

As a final caveat, observe that both the aggressive and the conservative failure recovery algorithms assume that all media blocks contained within a parity blocks belong to the same video stream. However, most of the conventional hardware RAID controllers do not provide any control over the membership of media blocks within a parity group. Consequently, to implement such algorithms, conventional hardware RAID controllers must be suitably extended. In the absence of such hardware controllers, such flexibility can be obtained by implementing a software RAID architecture. The software implementation of failure recovery operations (i.e., exclusive-or operation), however, may impose significant overhead on the CPU, and hence, limit the scalability of such an architecture. In the next section, we present a failure recovery method that addresses these limitations by exploiting human perceptual tolerances to approximate lost image data (rather than perfectly recovering data) using the inherent redundancies present 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, 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 in the presence of a single disk failure, all the neighbors of the lost pixels will be available. In this case, due to the high degree of correlation between neighboring pixels, lost pixels can be reasonably approximated from their neighboring pixels. Notice that, 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.[14,17] The resultant increase in the bit-rate may impose a 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 limitations of the pre-compression image partitioning

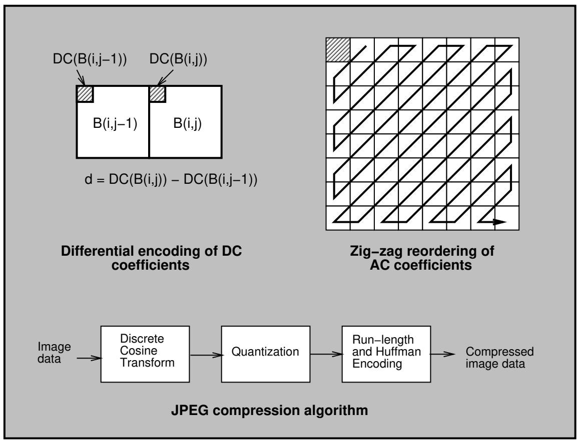

Figure 4: JPEG compression algorithm

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 4 depicts the main steps involved in the JPEG compression algorithm.[13]

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 **nrow** $\times$ **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:

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

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

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

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

- *Necessity*: Consider an 8x8 block $B(x, y)$ 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:

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

  By the definition of 8-neighborhood, it is clear that none of the groups contain any two blocks that are 8-neighbors 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:

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

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 $B(i_1, j_1)$ and $B(i_2, j_2)$ 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 $B(i_1, j_1)$ and $B(i_2, j_2)$ 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 two blocks that are in the 8-neighborhood of each other. ■

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. Our experiments indicate that, for most images, this objective is achieved when the image is partitioned into 8 sub-images[18] (See Figure 5).

3.1.2 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 (See figure 6); 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.

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 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. Furthermore, 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. Complete descriptions of the SRJ compression and decompression algorithms can be found in Vin et. al.[18]

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.

Figure 5: Reconstructed image for $N = 4, 8, 12, 16$ with a single disk failure

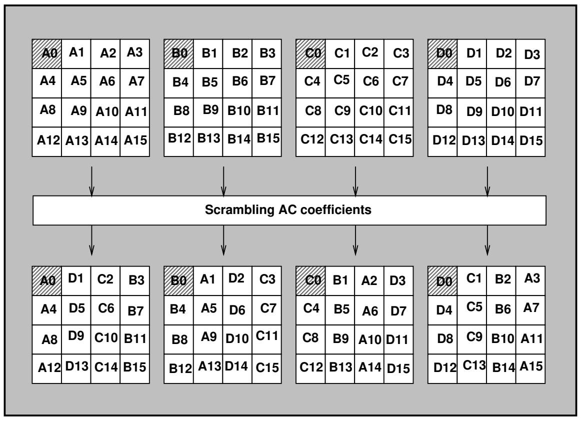

Figure 6: Scrambling AC coefficients. Here A_0, B_0, C_0 and D_0 denote DC coefficients, and $\forall i \in [1, 15] : A_i, B_i, C_i$ and D_i represent AC coefficients.

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 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, a large fraction of the quantized AC coefficients are zeroes. 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 SRJ algorithm can be mostly attributed to the need for replicating huffman tables and the reduced correlation between 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).

Notice that, with a degree of image partitioning of 8, the SRJ algorithm imposes 6% higher storage space requirement as compared to the JPEG compression algorithm.[18] To ensure a similar storage space overhead in RAID architectures, parity group size must be chosen to be at least 16. Assuming the same striping unit size in both cases, since the RAID architecture is required to buffer blocks constituting at least one parity group during failure recovery, they impose 100% larger buffer space requirement at the server as compared to the SRAD architecture. On the other hand, if the buffer space at the server is held fixed for both of the architectures, then a smaller parity group size must be chosen in RAID architectures, which significantly increases the storage space overhead as compared to the SRJ algorithm.

Observe 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 be able to perfectly reconstruct the contents of the failed disk onto a spare disk. To do so, the server can either reconstruct lost information from tertiary storage, or maintain parity information. In the latter 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.[7] 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.

Finally, the SRJ algorithm can provide network-level fault tolerance in addition to disk-array based fault tolerance. Since client sites can not distinguish between data loss due to a disk failure from that due to network congestion, they can utilize the recovery process to reconstruct information lost during network transmission as well. Thus, the SRJ algorithm provides an integrated, scalable, end-to-end solution for failure recovery.

4 CONCLUDING REMARKS

In this paper, we have demonstrated the limitations of the conventional parity-based failure recovery algorithms, 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 impose any significant load on the disk array. Whereas the first approach exploits the sequential nature of playback of video streams to reduce the overhead of the on-the-fly recovery process, the second utilizes the inherent redundancy in video streams (rather than error-correcting codes) to recover from disk failures in multimedia servers. We have demonstrated that the latter approach 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.

5 REFERENCES

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

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

[3] J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L.A. Rowe. Multimedia Storage Servers: A Tutorial and Survey. *IEEE Computer*, 28(5):40–49, 1995.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

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

[14] 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*, pages 243–252, February 1995.

[15] R. Tewari, D M. Dias, R Mukherjee, and H M. Vin. High Availability in Clustered Multimedia Servers. Technical report, IBM, 1995.

[16] 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.

[17] 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.

[18] 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.