Automatic Recovery from Disk Failure in Continuous-Media Servers

Jack Y.B. Lee, *Member*, *IEEE*, and John C.S. Lui, *Member*, *IEEE*

Abstract—Continuous-media (CM) servers have been around for some years. Apart from server capacity, another important issue in the deployment of CM servers is reliability. This study investigates rebuild algorithms for automatically rebuilding data stored in a failed disk into a spare disk. Specifically, a block-based rebuild algorithm is studied with the rebuild time and buffer requirement modeled. A buffer-sharing scheme is then proposed to eliminate the additional buffers needed by the rebuild process. To further improve rebuild performance, a track-based rebuild algorithm that rebuilds lost data in tracks is proposed and analyzed. Results show that track-based rebuild, while it substantially outperforms block-based rebuild, requires significantly more buffers (17-135 percent more) even with buffer sharing. To tackle this problem, a novel pipelined rebuild algorithm is proposed to take advantage of the sequential property of track retrievals to pipeline the reading and writing processes. This pipelined rebuild algorithm achieves the same rebuild performance as track-based rebuild, but reduces the extra buffer requirement to insignificant levels (0.7-1.9 percent). Numerical results computed using models of five commercial disk drives demonstrate that automatic rebuild of a failed disk can be done in a reasonable amount of time, even at relatively high server utilization (e.g., less than 1.5 hours at 90 percent utilization).

Index Terms—Continuous media, server, disk, rebuild, fault tolerance, reliability, scheduler, block-based, track-based, performance analysis.

1 Introduction

Since the introduction of continuous-media (CM) servers, a large number of researchers have investigated ways to improve server capacity to cope with the bandwidth requirement in delivering high-quality audio-visual contents to a large number of users. Apart from the challenge of capacity, another challenge—reliability—readily comes into the picture when companies deploy paid services.

Specifically, a CM server usually employs multiple disks in the form of a disk array for media data storage and retrieval. Media data are then distributed evenly across all the disks in small units such that data retrieval for a media stream will be spread across all disks to improve load balancing. However, one downside of this disk-array-based storage is reliability. In particular, failure of any one of the disks in the array will render the server inoperable due to data loss. Worse still, the reliability will further decrease when one adds more disks to scale up the system, thereby limiting the system's scalability.

This reliability problem has been investigated by many researchers in the last decade [1], [2], [3], [4], [5], [6], [7], [8] and a number of solutions have been proposed and studied. While the exact method varies, the basic principle is similar, i.e., adding redundant data to the disks so that data lost in a failed disk can be reconstructed in real-time for delivery to the client. A CM server is said to operate under normal mode when there is no disk failure and under degraded mode once a disk fails.

While existing solutions can sustain disk failure without service interruption, operating under degraded mode is still a temporary measure because an additional disk failure will result in total system failure and permanent data loss. Therefore, the server needs to initiate a rebuild mode to reconstruct data lost in the failed disk and store them to a spare disk to restore the server back to normal mode of operation. Once the rebuild process is completed, the server can sustain another disk failure without total system failure or permanent data loss. This gives the system operator more time to repair or replace the failed disk with a new spare disk.

It is worth noting that today's hard disk generally has fairly long mean-time-between-failure (MTBF) ratings, ranging from 300,000 hours to nearly 1,000,000 hours, depending on the particular disk model. Consider a CM server with 16 disks (including one parity disk) plus a spare disk. The MTBF for the disk array computed using a formula derived by Chen et al. [9] is over 42,000 years if the rebuild time is one hour and 4,200 years if the rebuild time is 10 hours. While an MTBF of 4,200 years may appear to be sufficient, Chen et al. [9] also pointed out that the computed MTBF should be taken conservatively because disk failures, in practice, are not necessarily independent. Hence, likelihood of a second disk failure could be much higher after the first disk failure. As the disk array MTBF is inversely proportional to the rebuild time, it is, therefore, important to quickly rebuild the failed disk to prevent total system failure.

To the best of the authors' knowledge, none of the existing work has investigated disk rebuild in a CM server. This paper addresses this problem by presenting efficient

- *J.Y.B. Lee is with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail: jacklee@computer.org.*
- *J.C.S. Lui is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail: cslui@cse.cuhk.edu.hk.*

Manuscript received 6 Oct. 2000; revised 14 May 2001; accepted 17 Oct. 2001.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 112959.

rebuild algorithms to rebuild the failed disk automatically and transparently in a CM server serving constant-bit-rate (CBR) video streams. *Automatic* refers to the fact that the rebuild process does not require human intervention such as locating and loading a backup tape to restore data. *Transparent* refers to the fact that the rebuild process itself can operate without any adverse effect on existing users.

The rest of this paper is organized as follows: Section 2 reviews some previous works and compares them with this study. Section 3 presents and formulates the system model studied in this paper. Section 4 presents and analyzes a block-based rebuild algorithm. Section 5 presents and analyzes a track-based rebuild algorithm. Section 6 presents a pipelined rebuild algorithm to reduce buffer requirement in track-based rebuild. Section 7 compares the presented algorithms quantitatively using numerical results. And, Section 8 concludes the paper and discusses some future works.

2 Background Related Works

In this section, we review existing work on disk fault tolerance and summarize the contributions of this study.

2.1 Related Works

The problem of supporting degraded mode of operation in CM servers has been investigated by a number of researchers [1], [2], [3], [4], [5], [6], [7], [8]. One approach makes use of data replications such as mirroring to sustain disk failure. The idea is to place two or more replicas in different disks so that at least one copy is available after a disk failure. Examples include the rotational mirrored declustering scheme proposed by Chen et al. [4], the doubly striped mirroring scheme proposed by Mourad [6], and the random duplicated assignment proposed by Korst [8].

Another approach makes use of parity encoding for data redundancy. A parity block, together with a number of data blocks, forms a parity group. The entire parity group can be reconstructed even if one of the blocks in the parity group is lost in a disk failure. Compared to replication, parity encoding generally requires less redundancy overhead, but higher buffer requirement for data reconstruction. This approach has been investigated by Tobagi et al. [1] in their Streaming RAID architecture, by Cohen et al. [3] in their pipelined disk array, by Berson et al. [2] in their non-clustered scheme, and by Özden et al. [5] in their declustered parity scheme and prefetch scheme.

In another work by Cohen and Burkhard [7], a segmented information dispersal (SID) scheme was proposed to allow fine grain tradeoff between the two extremes of mirroring and RAID-5 parity encoding. Reconstruction reads under SID are contiguous, leading to better disk efficiency. The authors showed that the SID schemes match the performance of RAID-5 and schemes based on balanced incomplete block designs under normal mode and outperforms them under degraded mode of operation.

The previous studies all focus on the normal mode and degraded mode of operation. The problem of rebuilding data in a failed disk to a spare disk in a CM server has yet to be studied. While there are many existing studies on disk rebuild, they all focus on data applications, such as online transaction processing (OLTP) servers. Some examples are the work by Menon and Mattson [10], [11], Hou and Patt [12], Hou et al. [13], Thomasian [14], [15], Thomasian and Menon [16], Mogi and Kitsuregawa [17], etc.

Disk rebuild in CM server applications differs from that of OLTP applications in two major ways. First, OLTP applications generally do not have the stringent performance requirement of a CM server. In particular, performance of OLTP applications is commonly measured using response time. While shorter response time is desirable, it is not a condition for correct operation. Therefore, in disk rebuild, the focus is to balance service response time with rebuild time. For example, one can use priority scheduling in OLTP applications to give higher priority to normal requests to minimize their response time and serve rebuild requests with the unused disk time.

By contrast, a CM server has to *guarantee* the retrieval of media data according to a fixed schedule. Even a small delay beyond the schedule will result in service disruption. Consequently, the rebuild process can take place only if normal media data retrievals can still be completed on time. This requires detailed disk modeling and using the use of worst-case analysis to determine exactly how much disk time can be spent on the rebuild process. Unlike rebuild algorithms for OLTP applications, the amount of disk time to spend on rebuild is determined a priori given the disk parameters. Moreover, retrievals for playback data and rebuild data are scheduled to minimize disk-seek time instead of according to priority as in the OLTP case.

Second, OLTP applications commonly employ the RAID-5 striping scheme to maximize I/O concurrency [9]. On the other hand, CM server applications commonly employs the RAID-3 striping scheme for reasons to be discussed in Section 3.1. This fundamental difference in the striping scheme, together with the inherently round-based disk scheduling algorithm employed in CM servers, requires different designs for the rebuild algorithm.

2.2 Contributions

This study incorporates CM server's stringent performance requirement and present rebuild algorithms that can maintain performance guarantee to all existing users while rebuilding the failed disk in the background. We first present a block-based rebuild algorithm and analyze the rebuild time and buffer requirement. Next, we propose a buffer-sharing scheme to eliminate the additional buffers required for rebuild. To further improve rebuild performance, we present and analyze a track-based rebuild algorithm that rebuilds lost data in tracks rather than in blocks. Track-based rebuild takes advantage of the read-on-arrival feature in modern disks to eliminate rotational latency, leading to significantly improved rebuild performance. However, track-based rebuild unfortunately requires significantly more buffers, even with buffer sharing (17-135 percent more). To tackle this problem, we present a novel pipelined rebuild algorithm to take advantage of the sequential property of track retrievals to pipeline the reading and writing processes. This pipelined rebuild algorithm achieves the same rebuild performance as track-

based rebuild but reduces the extra buffer requirement to insignificant levels (0.7-1.9 percent). Our results demonstrate that automatic rebuild of a failed disk can be done in a reasonable amount of time, even at relatively high server utilization (e.g., less than 1.5 hours at 90 percent utilization).

3 System Model

In this section, we present the system model used in this study. In particular, we discuss the rationale for adopting the RAID-3 striping scheme instead of the RAID-5 striping scheme; present a storage allocation policy and I/O scheduling algorithm based on the RAID-3 striping scheme; present a detailed disk model; and explain a capacity dimensioning procedure.

3.1 Disk Redundancy

In the pioneering study by Patterson et al. [18], a number of striping schemes are proposed for supporting disk-level fault tolerance. Among them, RAID-5 is the most widely used in general data and OLTP applications. RAID-5 is designed to maximize I/O concurrency by allowing individual disks in a disk array to serve different requests simultaneously. This design choice is particularly suitable for OLTP applications, as request size is generally small and performing I/Os in parallel can reduce response time.

For CM server applications however, RAID-5 is less popular for two reasons. First, CM server generally uses large request size to maximize disk throughput. Moreover, minimizing response time is less important, as most CM disk scheduler operates in fixed-duration cycles. As long as a block can be retrieved within the cycle, the exact response time is irrelevant.

Second, when a RAID-5 disk array operates in degraded mode with a failed disk, significant overheads (up to 100 percent depending on the placement policy) will be incurred in the remaining disks because reconstructing an unavailable block requires reading corresponding blocks from the same parity group from all the remaining disks. While increases in response time due to this overhead is not critical in OLTP applications, the same overhead will destroy the performance guarantee required in a CM server.[1]

For the previous two reasons, it is more common to employ the RAID-3 striping scheme for use in CM server applications. Unlike RAID-5, where each disk can serve a different I/O request, all the disks in RAID-3 participate in serving an I/O request, thereby maximizing disk throughput. More importantly, no additional overhead will be incurred by the remaining disks during degraded mode. This is because each I/O always retrieves the entire parity group from all the remaining disks and, hence, reconstruction can take place immediately. We will focus on this RAID-3 striping scheme in the rest of the paper.

1. This second problem can be solved by performing striping at the application level instead of at the storage level, i.e., all blocks in a parity group store data from the same media stream. However, it is still necessary to retrieve the entire parity group to reconstruct the unavailable block and, by doing so, the RAID-5 disk array is practically operated as a RAID-3 disk array with block interleaving instead of bit interleaving. In this case, the algorithms presented in this paper will also be applicable.

3.2 Storage Allocation and I/O Scheduling

Table 1a and Table 1b summarize the notations used in this paper. Let N_D be the number of disks in the server where $(N_D - 1)$ of them store data while the remaining one stores parity. The storage is divided into blocks of Q bytes, as shown in Fig. 1. Assuming the disks are homogeneous, then a parity group comprises one block at the same location from each one of the N_D disks. The parity block is computed from the $(N_D - 1)$ data blocks using exclusive-or computation. The storage is then allocated in whole parity groups instead of individual blocks to ensure that data blocks within a parity group always store data from the same CM stream. This RAID-3 striping scheme enables the system to mask the failure of any one of the N_D disks to continue operation through erasure correction processing (Fig. 2).

Retrievals and transmissions are organized into rounds, as shown in Fig. 3. We assume that all media streams are encoded with constant-bit-rate encoding at the same bitrate. Short-term bitrate variations (e.g., due to I, P, B frame differences in MPEG) are assumed to be absorbed by client buffers and, hence, the disk can simply retrieve exactly one media block from each of the N_D disks for each active media stream in each round. Note that this can also be extended to support other bitrates which are multiples of a base rate. These higher-rate streams can be treated as multiple base-rate streams and, hence, we will ignore this minor complication in the rest of the paper.

With the previous disk scheduler, a complete parity group, including the $(N_D - 1)$ media blocks and the associated parity block are all retrieved for each stream in a service round. This enables the server to sustain nonstop service, even when one of the disks fails, by computing the unavailable media block using erasure-correction computation over the remaining blocks in the parity group. Let R_v be the media bitrate. Then, the retrieved $(N_D - 1)$ media blocks will be transmitted in the next service round and the service round length is thus given by

$$T_r = \frac{(N_D - 1)Q}{R_v}. \quad (1)$$

Under this scheduling algorithm, the total number of buffers required is given by

$$B_p = KN_DQ + K(N_D - 1)Q, \quad (2)$$

where the first term is the buffer requirement for retrieval, the second term is the buffer requirement for transmission, and K is the maximum number of requests that can be served in a service round (see Section 3.4). Transmission requires fewer buffers because the retrieved parity block is not transmitted and hence the buffer can be reused.

For a server with a large number of disks, the single parity disk may not provide sufficient redundancy to maintain acceptable reliability. This problem can be solved by dividing the disks into clusters where each cluster has its own parity disk (e.g., Streaming RAID [1]). Multiple disk failures can be sustained as long as no more than one disk

TABLE 1a
Summary of Notations

Description	Symbol	Notes
Average media bit-rate.	R_v	Parameter in bytes per second (e.g. 150,000 Bps).
Media block size.	Q	Parameter in bytes (e.g. 65336 bytes).
Length of a transmission cycle.	T_r	Computed from Q/R, in seconds.
Number of disks.	N_D	Parameter.
Disk storage capacity.	G	Parameter in bytes.
Number of recording surfaces in a disk.	N_{suf}	Parameter in numbers.
Number of tracks per recording surface in a disk.	N_{trk}	Parameter in numbers.
Size of a disk sector.	S	Parameter in bytes.
Number of zones in a disk.	N_{zone}	Parameter in numbers.
Number of sectors per track in zone i.	Y_i	Parameter in numbers.
Minimum number of sectors per track.	Y_{min}	Parameter in numbers.
Maximum number of sectors per track.	Y_{max}	Parameter in numbers.
Transfer rate in zone i.	X_i	Computed, in bytes per second.
Minimum transfer rate.	X_{min}	Computed, in bytes per second.
Disk platter rotation rate.	W	Parameter in cycles per second.
Fixed overhead in disk read.	α	Parameter in seconds.
Seek time in disk read.	t_{seek}	Random variable in seconds.
Rotational latency in disk read.	t_{rot}	Random variable in seconds.
Transfer time in disk read.	t_{xfr}	Random variable in seconds.
Disk seek function.	$f_{seek}(n)$	Seek time for a seek distance of n tracks.
Maximum number of requests that can be served in a service round.	K	Computed, in number of requests.
Head-switching time in disk.	t_{hsw}	Parameter in seconds.
Total track-to-track seek time in serving a request.	t_{track}	Random variable, in seconds.
Earliest completion time for retrieving track i.	e_i	Random variable, in seconds.
Latest completion time for retrieving track i.	l_i	Random variable, in seconds.
Minimum time for retrieving one track.	Δ	Parameter in seconds.
Deviation bound for track-retrieval completion times.	D_{asyn}	Parameter in seconds.
Server load/utilization.	u	Variable, in number of streams.
Number of active streams for a server.	U	Variable, in number of streams.
Server utilization.	ρ	Variable, real number ranging from 0 to 1.
Number of rebuild blocks retrieved by a working disk in a round.	n_b	Computed, in number of media blocks.

fails in a cluster. Results in this study can be directly extended to these clustered schemes and, hence, we will focus on single-cluster disk arrays in the rest of the paper.

3.3 Disk Performance Model

We consider a general model for a multizone disk. Let N_{trk} be the number of tracks per recording surface (or number of cylinders), N_{suf} be the number of recording surfaces, W be the disk rotation speed in rounds per second, S be the sector size in bytes, and N_{zone} be the number of zones, $Y_i (i = 1, 2, \ldots, N_{zone})$ be the number of sectors per track in zone i. Note the disk transfer rate, denoted by $X_i (i = 1, 2, \ldots, N_{zone})$, is also zone-dependent and is given by

$$X_i = SY_iW. \tag{3}$$

TABLE 1b
Summary of Notations (continued)

Description	Symbol	Notes
Rate at which rebuild data are retrieved from the working disks.	R_{rb}	Computed, in bytes per second.
Rate at which lost data are rebuilt.	$R_{rebuild}$	Computed, in bytes per second.
Buffer requirement for playback process.	B_p	Computed, in bytes.
Buffer requirement for rebuild process.	B_r	Computed, in bytes.
Total buffer requirement without buffer sharing.	B_{sum}	Computed, in bytes.
Total buffer requirement with buffer sharing.	B_{share}	Computed, in bytes.

To simplify notations in later sections, we define

$$X_{\min} = \min\{X_i | i = 1, 2, \ldots, N_{zone}\} \quad (4)$$

$$Y_{\min} = \min\{Y_i | i = 1, 2, \ldots, N_{zone}\} \quad (5)$$

$$Y_{\max} = \max\{Y_i | i = 1, 2, \ldots, N_{zone}\} \quad (6)$$

and we shall leave out the subscript i in X_i and Y_i when representing random variables (i.e., X, Y) instead of system parameters.

To model disk performance, we consider the time it takes to serve a request. Specifically, disk time for retrieving a single request can be broken down into four components, namely, fixed overhead (e.g., head-switching time, settling time, etc.) denoted by α, seek time, denoted by t_{seek}, rotational latency, denoted by t_{rot}, and transfer time, denoted by t_{xfr}:

$$t_{req} = \alpha + t_{seek} + t_{rot} + t_{xfr}. \quad (7)$$

Seek time depends on the seek distance and it can be modeled by a seek function $f_{seek}(n)$, where n is the number of tracks to seek. For rotational latency, the random variable t_{rot} will be uniformly distributed between 0 and W^{-1}. Finally, the transfer time t_{xfr} comprises three components:

$$t_{xfr} = \frac{Q}{X} + t_{hsw} + t_{track}, \quad (8)$$

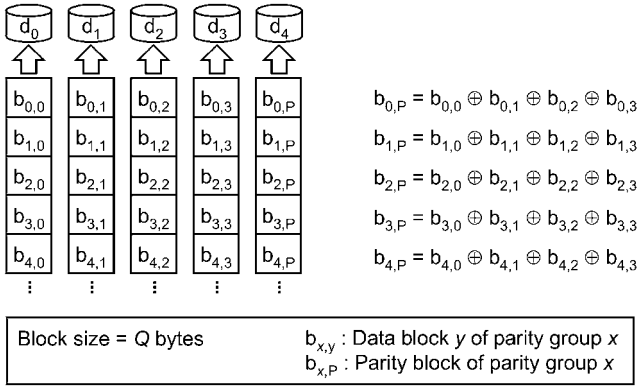


Fig. 1. Storage organization under RAID-3 striping scheme.

where the first term is the time it takes to read the media block of Q bytes from the disk surface, the second term is the total head-switching time incurred in case the media block spans more than one track in the cylinder, and the last term is the total track-to-track seek time incurred in case the media block spans more than one consecutive cylinders.

Take the Quantum Atlas-10K disk model as an example. The transfer time for retrieving a 64KB media block ranges from 4.59 ms to 6.69 ms depending on the zone, the head-switching time is 0.176 ms, and the track-to-track seek time is 1.25 ms. Therefore, unless one sacrifices some storage (7-10 percent depending on zone) to prevent a media block spanning two tracks, the effect of track-crossing should not be ignored.

We note that the previous disk model is only an approximation and is chosen for sake of simplicity. The results can be extended to more detailed and complex models for more accurate performance predictions.

3.4 Capacity Dimensioning

The goal of capacity dimensioning is to determine the maximum number of concurrent media streams that can be sustained with deterministic performance guarantee. Based on the previous disk model, we first obtain an upper bound for the length of a service round. Rewriting (7) by replacing t_{xfr} with the transfer time formula in (8), we have

$$t_{req} = \alpha + t_{seek} + t_{hsw} + t_{track} + t_{rot} + \frac{Q}{X}. \quad (9)$$

Assuming the use of CSCAN serving k requests per round and the seek time function is concave, then the seek time

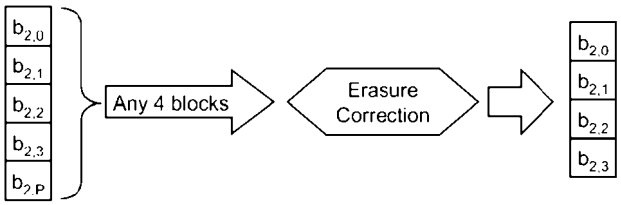


Fig. 2. Reconstructing media data with erasure correction.

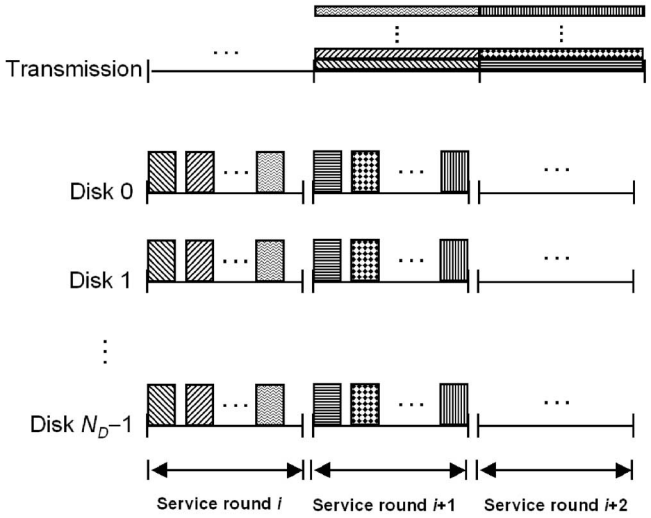


Fig. 3. Parallel-schedule retrieval algorithm.

overhead is maximized when the requests are evenly spaced along the disk surface, i.e.,

$$t_{seek}^{\max} = \max\{t_{seek}\} = f_{seek}(N_{trk}/(k+1)). \tag{10}$$

Note the use of $(k+1)$ instead of k to account for effect of the head-repositioning delay in CSCAN [19], [20].

To determine an upper bound for t_{hsw} and t_{track}, we note that a media block of size Q bytes can span at most

$$n_{hsw} = \lceil Q/(SY_{\min}) \rceil \tag{11}$$

recording surfaces. Therefore, the worst-case total head-switching time can be computed from

$$\max\{t_{head}\} = \left\lceil \frac{Q}{SY_{\min}} \right\rceil t_{hsw}. \tag{12}$$

Similarly, the same request can span at most

$$n_{cyl} = \left\lceil \frac{Q}{SY_{\min} N_{suf}} \right\rceil \tag{13}$$

cylinders. Hence, the worst-case total track-to-track seek time can be computed from

$$\max\{t_{track}\} = \left\lceil \frac{Q}{SY_{\min} N_{suf}} \right\rceil f_{seek}(1). \tag{14}$$

Lastly, the maximum rotational latency is simply given by W^{-1} while the maximum reading time can be obtained from

$$\max\left\{\frac{Q}{X}\right\} = \frac{Q}{X_{\min}}. \tag{15}$$

While summing the previous upper bounds will also bound the request service time, we note that most, if not all, modern disk drives support a read-on-arrival feature [3] (all five disk drive models studied in Section 7 supports read-on-arrival). Specifically, the service time model assumes that if the disk head arrives to the track to find it passing through the middle of the media block, it will wait until the first sector of the media block rotates back to the head position before commencing reading. The read-on-arrival feature removes this restriction and allows the disk to start reading data immediately, even in the middle of the requested media block. This avoids the worst-case scenario of waiting one complete rotation before reading.

Therefore, for a service round of k requests, the round length will be bounded from above by

$$t_r(k) = k\left(\begin{array}{l} \alpha + t_{seek}^{\max} + \left\lceil \frac{Q}{SY_{\min}} \right\rceil t_{hsw} \\ \quad + \left\lceil \frac{Q}{SY_{\min} N_{suf}} \right\rceil f_{seek}(1) + \left\lceil \frac{Q}{SY_{\min}} \right\rceil W^{-1} \end{array}\right) + t_{seek}^{\max}, \tag{16}$$

where the first term represents the worst-case time to read k requests using CSCAN and the second term is the head-repositioning time.

Reconsider the scheduling algorithm in Section 3.2, the server needs to ensure that a complete parity group must be retrieved within time T_r to maintain continuous transmission:

$$t_r(k) \le T_r. \tag{17}$$

This timing constraint, also known as the continuity condition, determines the maximum number of concurrent media streams, denoted by K, that can be supported by the server:

$$K = \max\left\{ k \mid t_r(k) \le \frac{(N_D - 1)Q}{R_v}, k = 1, 2, \ldots \right\}. \tag{18}$$

4 Automatic Data Rebuild

A system is said to operate under normal mode when there is no failure. The system switches to degraded-mode of operation once a disk failure occurs. Under this degraded-mode of operation, unavailable data are recomputed in real-time from the remaining disks to sustain service. Although it is still operational, the system must return to normal-mode of operation as soon as possible because any more disk failure will cripple the entire system. The goal of data rebuild is to bring the system back to normal-mode of operation by reconstructing data lost in the failed disk into spare storage hardware.

A number of modern disks and disk controllers not only can detect a disk failure, but can also predict a disk failure in advance. This early-warning signal can be used to initiate data rebuild even before the actual failure occurs. However, there are also complications that must be handled properly. First, if the data disks are updated (i.e., being written to) during the rebuild process, then the spare disk will have to be updated accordingly as well. This is less of a problem in a CM server as the disks primary serve read requests. One can also disallow updating until the rebuild process completes. Second, if the actual failure occurs in a disk other than the predicted one, then the rebuild process will have to be aborted and then restarted to rebuild the disk that failed. Nevertheless, this is equivalent to the case

without failure prediction and, hence, will not degrade rebuild performance. For simplicity, we will not make use of this failure prediction feature in the rest of the paper.

4.1 Sparing Scheme

To support automatic data rebuild, a dedicated spare disk is reserved to store data reconstructed in the rebuild process. The spare disk is connected to the server at all times, but is not utilized during normal-mode and degraded-mode of operation. For this sparing scheme, the recomputed data will be stored to the spare disk, which will replace the failed disk once the rebuild process is completed. Note that human intervention is still required to replace the failed disk with another spare disk to cater for another disk failure, but this is less time critical.

4.2 Rebuild Algorithm

The challenge of automatic rebuild is to proceed with the rebuild process without interrupting user services. Specifically, all retrievals in a disk service round must finish within T_r seconds and the addition of rebuild requests must not violate this limit. Clearly, we can only utilize unused disk capacity for serving rebuild requests. Once rebuild blocks from the surviving disks are retrieved into memory, the server can then perform erasure-correction computation to reconstruct the lost media blocks and store them into the spare disk. This process repeats until all media blocks lost in the failed disk are reconstructed into the spare disk, which then simply replaces the failed disk to bring the system back into normal-mode of operation. The failed disk will later be replaced or repaired manually and a new spare disk will be reinserted into the system to prepare for the next rebuild cycle.

4.3 Analysis of Rebuild Time

A key performance metric in evaluating automatic data rebuild algorithms is rebuild time, defined as the time required to completely rebuild data in the failed disk to the spare disk. For a server with N_D disks (one of which has failed) and one spare disk, the rebuild process consists of reading $(N_D - 1)$ blocks for each parity group from the surviving $(N_D - 1)$ disks and reconstructing the lost media block for storage in the spare disk. Note that this is true even if the failed disk happens to be the parity disk because all $(N_D - 1)$ data blocks in a parity group are required to recompute the parity block for storage in the spare disk.

Let u, $0 \leq u \leq K$, be the number of active streams in the server. We define a server utilization ρ, $0 \leq \rho \leq 1$, as follows:

$$\rho = \frac{u}{K}. \tag{19}$$

Now, the number of rebuild blocks retrieved by a working disk in a service round, denoted by n_b, will be given by

$$n_b = K - u, \tag{20}$$

which is the same for all disks. Given that there are $(N_D - 1)$ working disks, the rate at which rebuild data are retrieved, denoted by R_{rb}, is then given by

$$\begin{aligned} R_{rb} &= \frac{(N_D - 1)n_b Q}{T_r} = \frac{(N_D - 1)n_b Q}{(N_D - 1)(Q/R_v)} \\ &= n_b R_v, \end{aligned} \tag{21}$$

where the numerator is the total amount of rebuild data retrieved in a service round and the denominator is the length of a service round. Note that R_{rb} is only the rate at which rebuild data is retrieved, the reconstruction process will consume $(N_D - 1)$ rebuild blocks to reconstruct each lost media block. Therefore, the rebuild rate $R_{rebuild}$, defined as the rate at which lost data are reconstructed, can be computed from R_{rb}:

$$R_{rebuild} = \frac{R_{rb}}{(N_D - 1)} = \frac{n_b R_v}{(N_D - 1)}. \tag{22}$$

Using (19) and (20), we can simplify (22) into

$$\begin{aligned} R_{rebuild} &= \frac{(K - u)R_v}{(N_D - 1)} \\ &= \frac{(1 - \rho)}{(N_D - 1)} K R_v. \end{aligned} \tag{23}$$

Equation (23) computes the achievable rebuild rate under a given server utilization. Let G be a disk's storage capacity. Assuming storage in the entire disk array is fully utilized, we can then calculate the rebuild time from

$$T_{rebuild} = \frac{G}{R_{rebuild}} = \frac{G(N_D - 1)}{K R_v (1 - \rho)}. \tag{24}$$

4.4 Buffer Requirement

Two types of buffers are required for a CM server supporting automatic rebuild. First, the server needs buffers for the normal retrieval and transmission of media blocks for playback (henceforth called *playback buffers*). Second, the server also needs buffers to support the rebuild process (henceforth called *rebuild buffers*).

The playback buffer requirement is given by (2) in Section 3. To determine the requirement for rebuild buffers, we consider the rebuild process depicted in Fig. 4. Note that write operations on the spare disk is scheduled using the same periodic scheduler as read operations on the data disks. Rebuild blocks retrieved in a round are used to reconstruct the lost media blocks for writing to the spare disk in the next round. This algorithm simplifies the server implementation as the same scheduler can be used for both read and write operations.

For every unavailable media blocks reconstructed, $(N_D - 1)$ blocks must be retrieved for erasure-correction computation. Therefore, the buffer requirement for rebuild is given by

$$B_r = K(N_D - 1)Q + KQ, \tag{25}$$

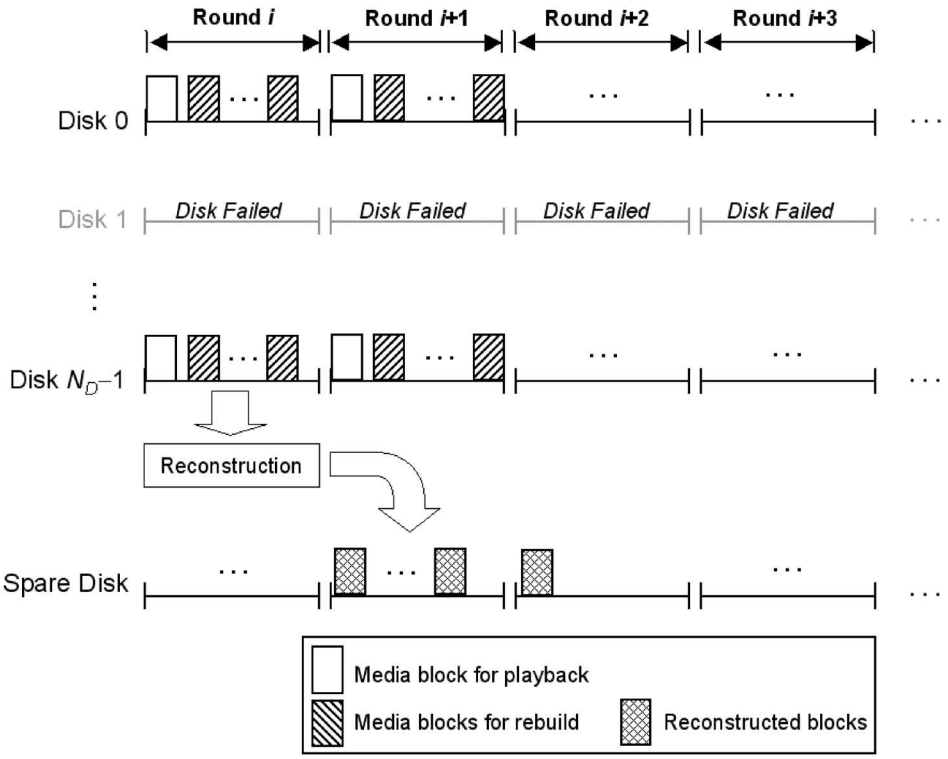


Fig. 4. Block-based automatic data rebuild.

where the first term is the buffer requirement for retrieval and the second term is the buffer requirement for storing the reconstructed media blocks to the spare disk. The multiplication factor K represents the maximum number of blocks that can be retrieved in a round for rebuild. The total buffer requirement can then be computed from the sum of (2) and (25):

$$\begin{aligned} B_{sum} &= B_p + B_r \\ &= K(3N_D - 1)Q. \end{aligned} \tag{26}$$

It may occur to the reader that, while simple in implementation, this periodic write scheduler is inefficient in buffer usage because reconstructed blocks are buffered for up to one cycle before writing to the spare disk. However, we discover that a simple buffer-sharing scheme will completely eliminate the additional buffer requirement.

Specifically, we notice that the server would need fewer buffers for playback and more buffers for rebuild when the utilization is low and vice versa. This motivates us to investigate buffer-sharing between the retrieval process and the rebuild process. Specifically, the server can allocate a pool of say N_{share} buffers (each Q bytes) during initialization and then allocate the buffers to the retrieval process and the rebuild process in an on-demand manner.

Now, consider the retrieval process. Given u active streams, the number of playback buffers required is given by

$$B_p(\rho) = u(2N_D - 1)Q. \tag{27}$$

For the rebuild process, the buffer requirement when there are u active streams is given by

$$B_r = (K - u)N_D Q. \tag{28}$$

Hence, the combined buffer requirement with buffer sharing is then given by

$$\begin{aligned} B_{share} &= u(2N_D - 1)Q + (K - u)N_D Q \\ &= ((N_D - 1)u + KN_D)Q \\ &\le (2N_D - 1)KQ \qquad \forall u, \end{aligned} \tag{29}$$

which surprisingly equals to the buffer requirement for the retrieval process. Therefore, with the proposed buffer-sharing scheme, one can use the same round-based disk scheduler for both reading and writing without any additional buffer.

5 Track-Based Rebuild

Most, if not all, modern disk drives employ zoning to increase disk storage capacity. A side-effect of zoning is the variation in track size. In particular, inner tracks have less storage capacity than outer tracks. Due to this uneven track-size problem, disk scheduler in most continuous-media server designs retrieves media units in fixed-size blocks instead of tracks. While reading the entire track can eliminate the rotational latency, the amount of buffer required to maintain a balanced disk schedule is often prohibitively large [6].

Unlike the retrieval process, the rebuild process is a non-realtime process that does not require data retrieval at a constant rate. Consequently, we can employ track-based retrieval for the rebuild process to improve rebuild

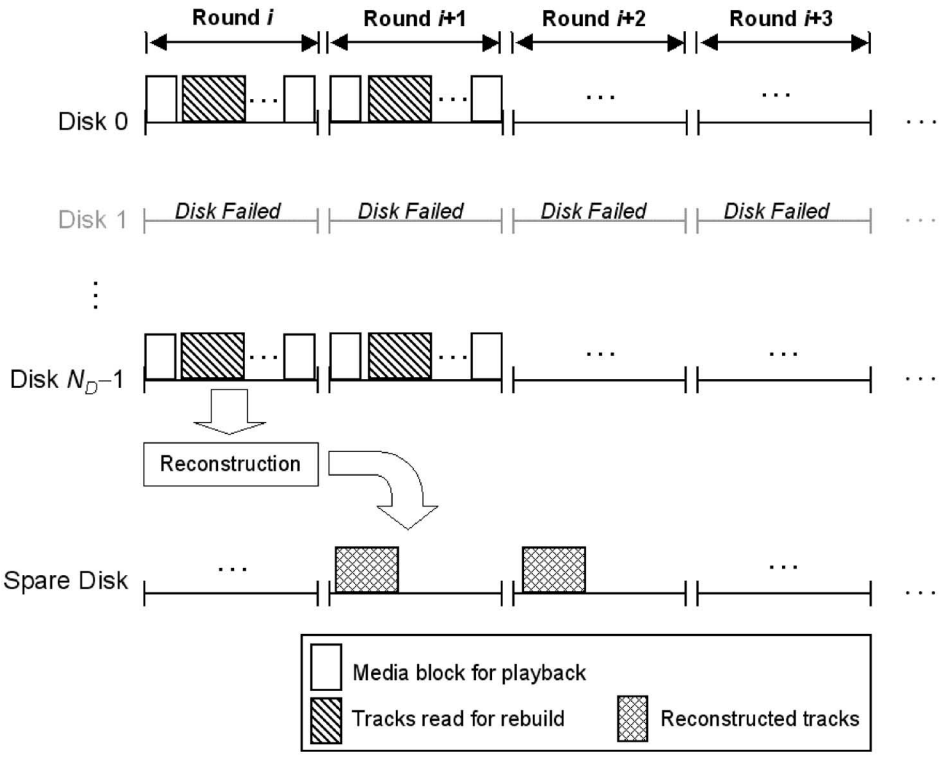


Fig. 5. Track-based automatic data rebuild.

performance while keep using block-based retrieval for the streaming process to maintain low buffer requirement.

5.1 Rebuild Algorithm

Fig. 5 depicts the track-based rebuild algorithm. In reading data from the data disks, playback data are still retrieved in fixed-size blocks but rebuild data are retrieved in tracks. This allows the elimination of rotational latency during rebuild data retrieval.

Specifically, in block-based retrieval, the disk head must wait for the required disk sector to rotate to beneath the disk head before data transfer can begin. In the worst case, where the required sector has just passed over the disk head after seeking is completed, the disk will have to wait for one complete round of rotation before beginning data transfer.

By contrast, under track-based retrieval, the disk head can start data transfer as soon as seeking is completed because the entire track is to be retrieved. Clearly, the reading time is simply the time for one disk rotation, i.e., W^{-1}. After reading the corresponding tracks from all $(N_D - 1)$ disks, the server can then reconstruct the lost track and write it to the spare disk.

Additionally, the rebuild process rebuilds tracks sequentially starting from one end of the disk surface with all track retrievals performed back-to-back in one go. For example, let y_i, $i = 0, 1, \ldots, (u-1)$ and $y_i \le y_j$ for $i < j$ be the track numbers for the u data blocks to be retrieved for playout in a round. Suppose that the next track to rebuild is track number x and a total of v tracks are to be rebuilt. Then, the order of retrievals will be $y_0, y_1, \ldots, y_i, x, y_j, \ldots, y_{u-1}$, where $y_i \le x \le y_j$. In other words, all v tracks are retrieved in one go between the retrievals of block i and j. Consequently, the seek time between track retrievals is reduced to $t_{seek}(1)$. The rebuild process will retrieve as many tracks as possible in a round for rebuild as long as retrieval performance for normal data blocks can still be guaranteed.

5.2 Analysis of Rebuild Time

To model the rebuild process, let $u (u \le K)$ be the number of media blocks to retrieve for playback and v be the number of tracks to retrieve for rebuild in a service round. Using the disk model in Section 3.3, the modified service round length is bounded from above by

$$\begin{aligned} t_r(u,v) = u & \left(\begin{array}{l} \alpha + f_{seek}(N_{trk}/(u+2)) + \left\lceil \frac{Q}{SY_{\min}} \right\rceil t_{hsw} \\ \quad + \left\lceil \frac{Q}{SY_{\min} N_{suf}} \right\rceil f_{seek}(1) + \left\lceil \frac{Q}{SY_{\min}} \right\rceil W^{-1} \end{array} \right) \\ & + f_{seek}(N_{trk}/(u+2)) + v\left(\alpha + t_{hsw} + W^{-1}\right) \\ & + \left\lceil \frac{v-1}{N_s} \right\rceil t_{seek}(1) \\ & + f_{seek}(N_{trk}/(u+2)). \end{aligned} \tag{30}$$

The first term is the service time for reading u media blocks. The second term is the additional seek time due to rebuild. The third term is the time for reading v tracks. The fourth term is the track-to-track seek time for reading rebuild tracks and the last term is the head-repositioning delay.

Now, invoking the continuity condition in (17), we can determine the maximum number of tracks that can be retrieved for rebuild given there are already u data requests in a round, denoted by $V(u)$, from

$$V(u) = \max\left\{ v | t_r(u,v) \frac{(N_D - 1)Q}{R_v}, v = 0, 1, \ldots \right\}. \tag{31}$$

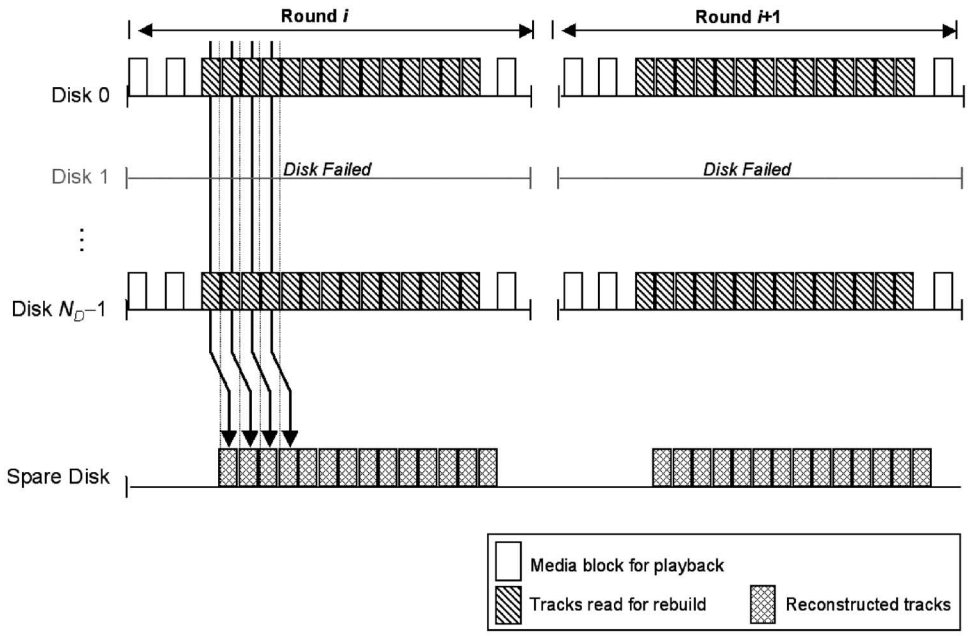


Fig. 6. Pipelined rebuild under ideal scenario of synchronized disks.

Given a disk with N_{suf} recording surfaces and N_{trk} tracks per surface, the rebuild time can then be computed from

$$T_{rebuild} = \frac{N_{trk}N_{suf}}{V(u)} \cdot \frac{(N_D - 1)Q}{R_v}. \tag{32}$$

5.3 Buffer Requirement

Under track-based rebuild, tracks retrieved in a service round will be consumed by the reconstruction process to compute the lost tracks for writing to the spare disk in the next service round. With a sector size of S bytes and up to Y_{max} sectors per track, the maximum buffer requirement for rebuild can be obtained from

$$B_r = V(0)(N_D - 1)SY_{\max} + V(0)SY_{\max}, \tag{33}$$

where the first term is the buffer for reading from the $(N_D - 1)$ working disks and the second term is the buffer for writing to the spare disk. Without buffer sharing, the total buffer requirement would be the sum of (2) and (33):

$$\begin{aligned} B_{sum} &= B_p + B_r \\ &= K(2N_D - 1)Q + V(0)N_D SY_{\max}. \end{aligned} \tag{34}$$

Using the buffer-sharing technique, we can compute the combined buffer requirement at a given server utilization from

$$B_{share}(u) = u(2N_D - 1)Q + V(u)N_D SY_{\max} \tag{35}$$

and the maximum buffer requirement can be computed from

$$B_{share} = \max\{B_{share}(u) | u = 0, 1, \ldots, K\}. \tag{36}$$

Intuitively, the larger the track size (i.e., SY_{max}), compared to the block size (i.e., Q), the more likely that the buffer requirement will be dominated by the rebuild process and vice versa. In the next section, we present a novel pipelined rebuild algorithm to reduce this buffer requirement.

6 Pipelined Rebuild

The possibility of track-based rebuild in multizone disks stems from the fact that rebuild requests are nonrealtime and, hence, can be served at variable rates. Another observation is that tracks are always retrieved sequentially to avoid seek overhead. This sequential property differs from normal data requests where the order of retrieval can change from round to round due to the CSCAN algorithm. We present in this section a pipelined rebuild algorithm to take advantage of this sequential property to reduce the buffer requirement to insignificant levels.

6.1 Buffer Requirement

Fig. 6 depicts the pipelined rebuild algorithm. The scheduling algorithm for retrieving data from the data disks are the same as track-based rebuild. The difference is in scheduling the write operations to the spare disk. Specifically, tracks reconstructed from track-based rebuild are buffered until all track retrievals are completed before writing to the spare disk. By contrast, in pipelined rebuild as soon as a track is retrieved from each of the $(N_D - 1)$ surviving disks, the server will reconstruct the lost track and store it to the spare server immediately. In this way, the track reading and writing processes operate simultaneously in a pipelined manner.

Under this pipelined rebuild algorithm, the rebuild buffer requirement is reduced to

$$B_r = (N_D - 1)SY_{\max} + SY_{\max}, \tag{37}$$

where the first term is the buffer required for reading and the second term is the buffer required for writing.

However, the scenario in Fig. 6 is idealized with the assumption that track retrievals for all surviving disks complete at the same instant. In practice, this is unlikely to be the case due to variations in disk rotational latencies incurred in reading media blocks prior to reading the

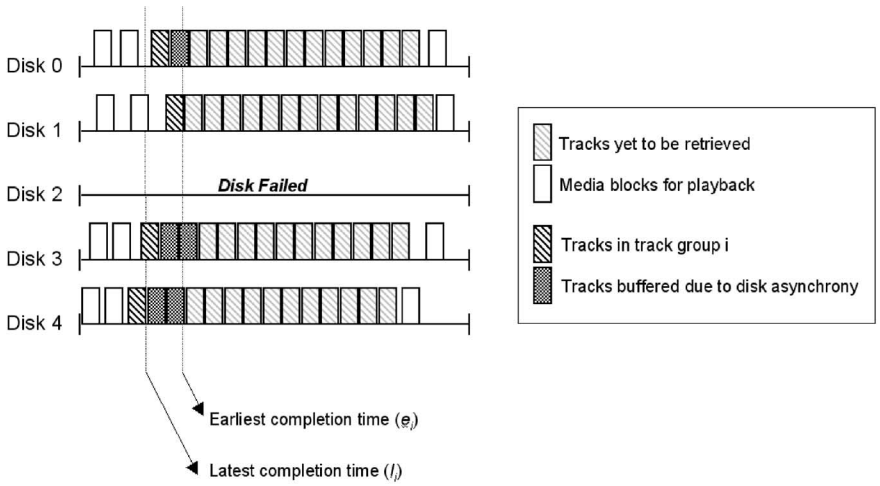


Fig. 7. A snapshot of track retrievals at time $t = l_i$ with disk asynchrony.

rebuild tracks. To account for this disk asynchrony, we introduce a deviation bound D_{asyn} defined as the maximum difference between the time the first track retrieval completes and the time the last track retrieval completes.

Mathematically, let $t_{i,j}$ be the retrieval completion time for reading rebuild track i, $i = 0, 1, \ldots, N_{trk}$, by disk j, $j = 0, 1, \ldots, (N_D - 1)$, as shown in Fig. 7. We define a *track group* as the set of corresponding tracks from all $(N_D - 1)$ surviving disks that forms a parity group. For example, track group i comprises track i from each of the $(N_D - 1)$ disks.

Let e_i and l_i be the earliest completion time and latest completion time, respectively, for track group i:

$$e_i = \min\{t_{i,j} | \forall j\} \text{ and } l_i = \max\{t_{i,j} | \forall j\}. \tag{38}$$

Then, D_{asyn} can be computed from

$$D_{asyn} = \max\{l_i - e_i | \forall i\}. \tag{39}$$

Let b_r be the number of buffers (each SY_{max} bytes) allocated for the rebuild process. Then, at time $t = l_i$, all $(N_D - 1)$ tracks for track group i are completely retrieved. Due to disk asynchrony, some of the disks may have completed retrieving track i earlier than l_i and have started reading subsequent tracks. In particular, the earliest time for a disk to start reading tracks $i + 1$ will simply be equal to e_i. Let Δ be the minimum time for retrieving a track:

$$\Delta = \alpha + t_{hsw} + W^{-1}, \tag{40}$$

then a disk can retrieve up to track

$$i + \left\lceil \frac{l_i - e_i}{\Delta} \right\rceil \tag{41}$$

by time $t = l_i$. In the worst case, all but the last disk have performed early retrievals and the buffer requirement (in number of tracks) will be given by

$$b_r = (N_D - 1) + \left\lceil \frac{l_i - e_i}{\Delta} \right\rceil (N_D - 2) + 1, \tag{42}$$

where the first term is the buffers for reading track group i, the second term is the buffers for early retrievals, and the last term is the buffer for writing to the spare disk.

Finally, the maximum buffer requirement can be obtained from

$$\begin{aligned} B_r &= \max\left\{ (N_D - 1) + \left\lceil \frac{l_i - e_i}{\Delta} \right\rceil (N_D - 2) + 1 \middle| \forall i \right\} SY_{\max} \\ &= \left((N_D - 1) + \left\lceil \frac{D_{asyn}}{\Delta} \right\rceil (N_D - 2) + 1 \right) SY_{\max} \end{aligned} \tag{43}$$

of which (37) becomes a special case of (43) with $D_{asyn} = 0$.

6.2 Active Disk Synchronization

In deriving the buffer requirement in (43), we assumed that disks that have completed reading a track earlier than others will continue reading the subsequent tracks. While this appears to be making efficient use of disk time, it is in fact counter-productive. Unlike transaction processing (OLTP) applications, residual disk time in a continuous-media server will not be used for retrieving additional media blocks due to the periodicity of the disk schedule. Therefore, even if there is residual disk time after reading all media blocks and rebuild tracks, the disk will just sit idle until the next service round.

This observation motivates us to propose an active disk synchronization (ADS) scheme to further reduce the buffer requirement in (43). Specifically, track retrievals for the surviving disks under ADS are actively synchronized according to the slowest disk. For example, in reading track group i, all disks will start their retrieval for track i at time $t = l_{i-1}$ instead of $t_{i-1,j}$ for disk j, as shown in Fig. 8. Note that the added delay will not affect the normal retrieval process nor the rebuild process as they are dimensioned according to the worst-case scenario.

Theoretically, with ADS, the deviation bound D_{asyn} will become zero. In practice, small deviations might still exist because the server is likely to send disk commands serially

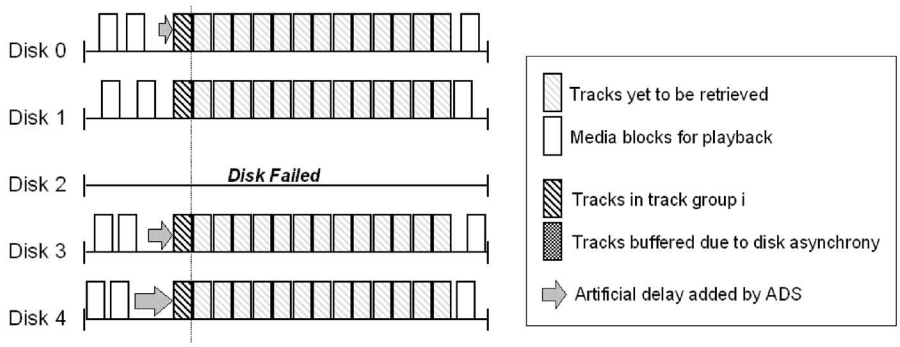


Fig. 8. A snapshot of track retrievals at time $t = l_i$ with Active Disk Synchronization.

to each of the surviving disks. Assuming this deviation is small compared to Δ, then

$$\left\lceil \frac{D_{asyn}}{\Delta} \right\rceil = 1 \qquad \text{for } D_{asyn} \leq \Delta \tag{44}$$

and the buffer requirement is reduced to

$$B_r = (2N_D - 2)SY_{\max}. \tag{45}$$

7 Performance Evaluation

Using the performance models derived in the previous sections, we present in this section numerical results computed for five disk drive models to quantitatively compare the studied algorithms. The disks' parameters are extracted from the disk specifications in Ganger et al. [21] and summarized in Table 2. Unless stated otherwise, the results are computed using a disk array configuration of four data disks, one parity disk, and one spare disk.

7.1 Comparison of Rebuild Time

Figs. 9 and 10 show the rebuild time versus server utilization for block-based rebuild and track-based rebuild, respectively. We observe that the rebuild time increases modestly until around a utilization of 0.8, after which, it increases rapidly due to limited capacity available for rebuild. For example, rebuild time for the Quantum Atlas-10K disk increases from 44.4 minutes at $\rho = 0.5$ to 221.9 minutes at $\rho = 0.9$ for block-based rebuild. Comparing Fig. 9 with Fig. 10, it is clear that track-based rebuild significantly outperforms block-based rebuild. With the same disk model, the rebuild time for track-based rebuild is only 12.8 minutes at $\rho = 0.5$ and 87.7 minutes at $\rho = 0.9$.

This result is encouraging, as rebuilding a failed disk requires less than 1.5 hours even at a server utilization of 0.9. Given that a service provider is likely to dimension a system to operate well below such a high utilization to minimize blocking, the rebuild time in practice is likely to be even shorter.

7.2 Sensitivity to Server Utilization

Fig. 11 plots the reduction in rebuild time by track-based rebuild versus server utilization. We observe that track-based rebuild consistently achieves significant rebuild-time reductions over a wide range of server utilization. This result demonstrates that performance gain of the proposed track-based rebuild is stable with respect to server utilization.

7.3 Sensitivity to Media Block Size

In Fig. 12, we plot the rebuild time versus the media block size at three server utilizations of 0, 0.25, and 0.5, respectively. We observe that rebuild time for block-based rebuild decreases with increases in media block size as larger block size increases the overall disk efficiency. By

TABLE 2
Parameters of Five Disk Models (from Ganger et al. [21])

Parameter	Quantum Atlas-III	Quantum Atlas-10K	Seagate Barracuda	Seagate Cheetah	IBM 9ES
No. of tracks	8057	10042	5172	6581	11474
No. of surfaces	10	6	5	8	5
Fixed overhead	0 ms	0 ms	0 ms	0 ms	0 ms
Head-switching time	0.999 ms	0.176 ms	0.1 ms	0.195 ms	0.062 ms
Spinning speed	7200 rpm	10025 rpm	7200 rpm	10033 rpm	7200 rpm
Sector size	512 bytes	512 bytes	512 bytes	512 bytes	512 bytes
Max track size	256 sectors	334 sectors	186 sectors	195 sectors	390 sectors
Min track size	168 sectors	229 sectors	119 sectors	131 sectors	247 sectors
Disk capacity	9.1GB	9.1GB	2GB	4.5GB	9GB
Track-to-track seek	1.663 ms	1.245 ms	1.943 ms	0.636 ms	1.086 ms

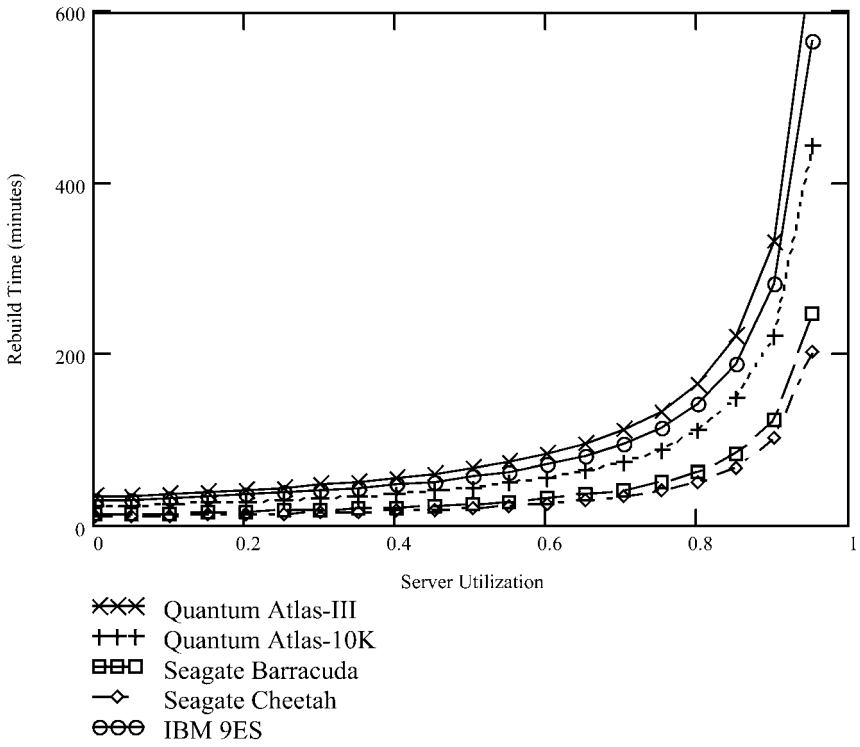


Fig. 9. Rebuild time versus server utilization for block-based rebuild ($Q = 64KB$, $N_D = 5$).

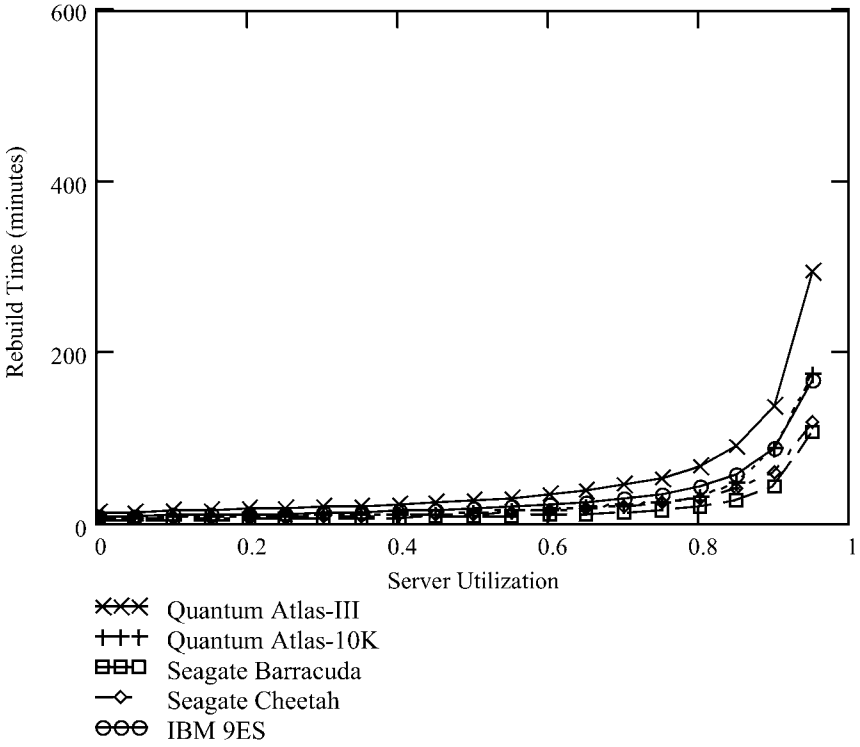


Fig. 10. Rebuild time versus server utilization for track-based rebuild ($Q = 64KB$, $N_D = 5$).

contrast, rebuild time for track-based rebuild is relatively insensitive to the media block size used as retrievals are done in whole tracks instead of blocks.

Fig. 13 plots the reduction in rebuild time versus the media block size. As expected the reduction decreases for increases in media block size as rebuild performance for block-based rebuild improves. However, even at a very large block size of 640KB, track-based rebuild still outperforms block-based rebuild by about 30 percent.

7.4 Buffer Requirement

We plot the buffer requirement for the studied algorithms versus number of disks in Fig. 14 for the Quantum Atlas-10K disk model. We observe that track-based rebuild, without

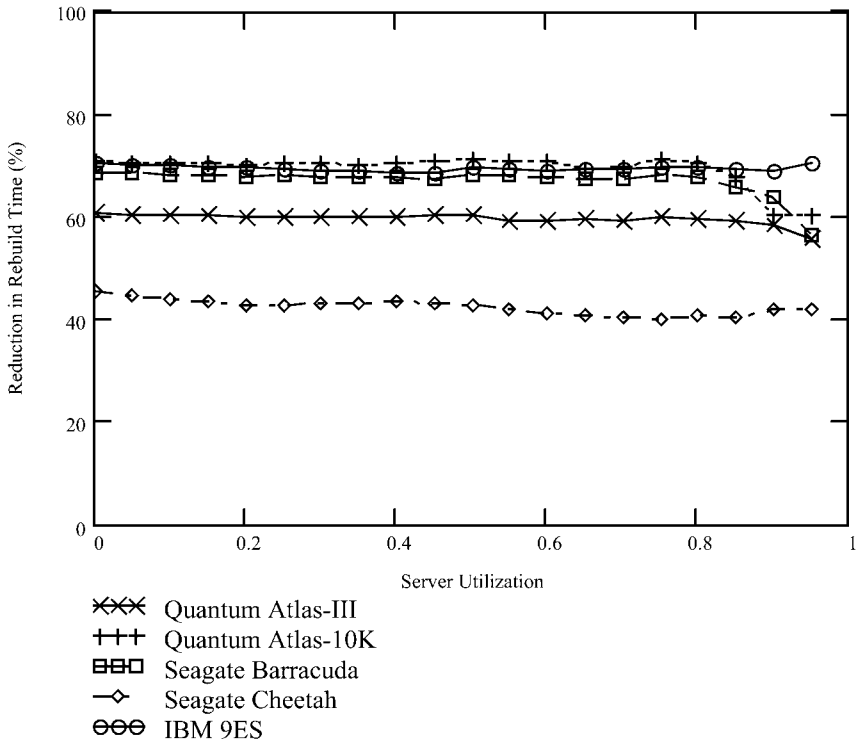


Fig. 11. Comparison of rebuild time reduction by track-based rebuild ($Q = 64KB$, $N_D = 5$).

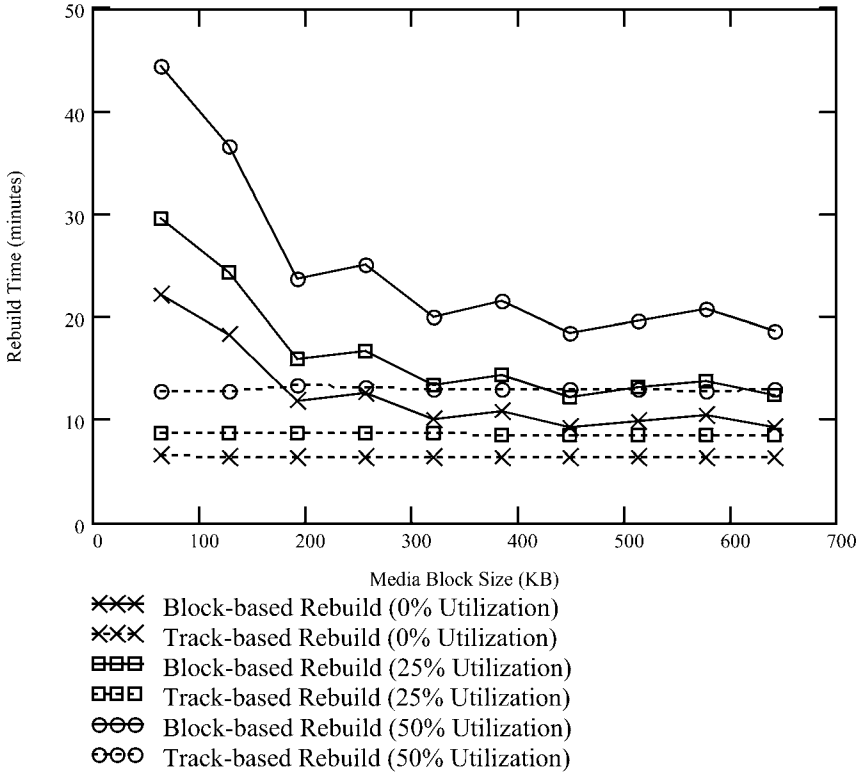


Fig. 12. Rebuild time versus media block size (Quantum Atlas-10K).

buffer sharing, has the largest buffer requirement as expected. However, even with buffer sharing, track-based rebuild still requires more buffers than block-based rebuild. This is due to the fact that the block size (64KB) used is smaller then the track size (varies from 114.5KB to 167KB) and, hence, the rebuild buffers dominate the buffer requirement.

By contrast, the proposed pipelined rebuild algorithm has only slightly larger buffer requirement than the best scheme—block-based rebuild with buffer sharing. For a five-disk server, pipelined rebuild requires only 0.7MB to

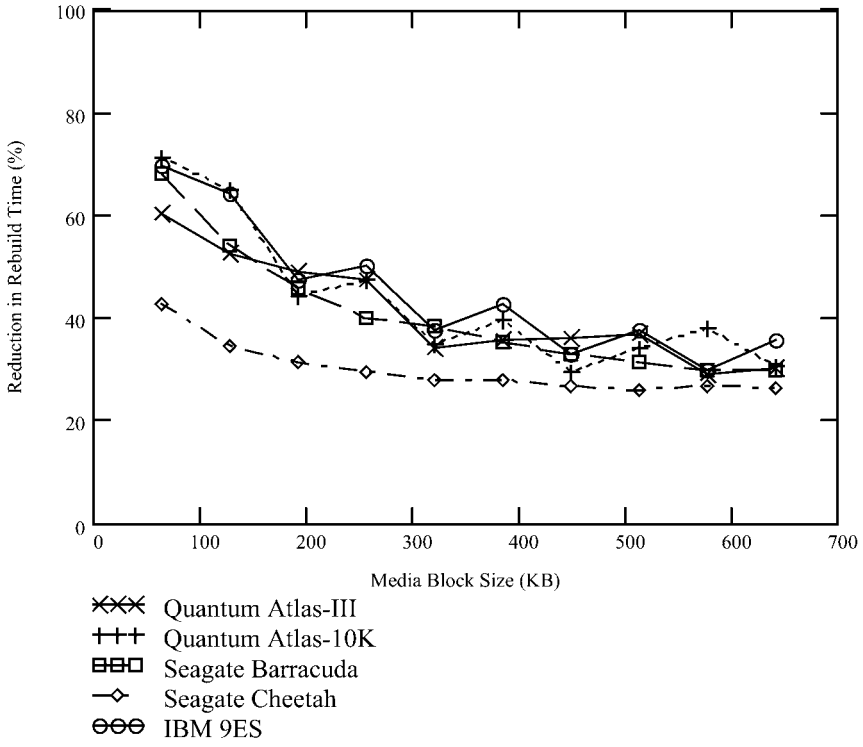


Fig. 13. Reduction in rebuild time versus media block size ($N_D = 5$, $\rho = 0.5$).

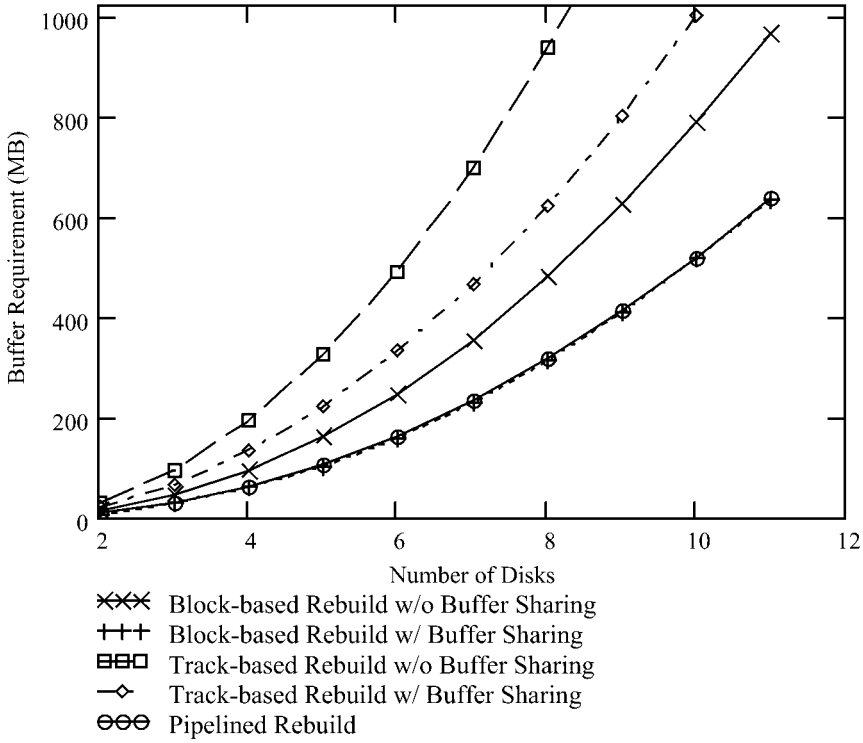


Fig. 14. Buffer requirement versus number of disks ($Q = 64KB$, Quantum Atlas-10K).

1.5MB more buffers than block-based rebuild with buffer sharing (see Table 3). Note that block-based rebuild with buffer sharing is already optimal because the same server will require just as much buffer without the rebuild option. Therefore, with pipelined rebuild, we can achieve significant gain in rebuild performance through track rebuild and at the same time avoid the large buffer requirement.

8 Conclusions and Discussions

In this paper, we investigate two algorithms for rebuilding data lost in a failed disk to a spare disk automatically and transparently. We first present a block-based rebuild algorithm derived from the conventional CSCAN disk scheduler and analyze its performance. A buffer-sharing

TABLE 3
Buffer Requirements ($Q = 64KB$, $N_D = 5$)

Rebuild Algorithm	Quantum Atlas-III	Quantum Atlas-10K	Seagate Barracuda	Seagate Cheetah	IBM 9ES
Block-based Rebuild w/o Buffer Sharing	107 MB	161 MB	67 MB	175 MB	126 MB
Block-based Rebuild with Buffer Sharing	69 MB	104 MB	43 MB	112 MB	81 MB
Track-based Rebuild w/o Buffer Sharing	182 MB	326 MB	132 MB	244 MB	272 MB
Track-based Rebuild with Buffer Sharing	114 MB	222 MB	89 MB	132 MB	191 MB
Pipelined Rebuild	70 MB	105 MB	44 MB	113 MB	83 MB

scheme is then proposed to eliminate the additional buffer requirement during rebuild. Next, we propose a track-based rebuild algorithm that can reduce the rebuild time by 70-80 percent. The large buffer requirement incurred in track-based rebuild is then reduced to insignificant levels by a novel pipelined rebuild algorithm. Numerical results show that it is feasible to completely rebuild a failed disk using the proposed algorithms in a practical amount of time without causing any performance degradation to the CM server.

While this study has been focused on CM servers serving CBR video streams, the proposed rebuild algorithms can also be extended to CM servers serving variable-bit-rate (VBR) video streams. One possible approach is to replace fixed-size block retrievals with variable-size block retrievals, with the block size corresponding to the instantaneous video bitrate. As long as the sizes of the blocks to be retrieved in a disk round is known, we can use the same worst-case analysis as in Section 5 to determine how much disk time to allocate for rebuild. The rest of the rebuild process will be the same.

In addition to the rebuild algorithms studied in this work, there are also other techniques that may further improve rebuild performance. In particular, when serving active media streams in rebuild mode, the system has to recover lost data blocks for playback purposes. By storing these already reconstructed data blocks to the spare disk, one may be able to further shorten the rebuild time.

However, storing individual data blocks to the spare disk might also adversely affect disk efficiency in track-based rebuild. First, compared to track-based rebuild, more time is spent seeking rather than data transfer in block-based rebuild. Hence, the reduction in reading from the data disks is offset by the lost in disk efficiency in the spare disk. Second, depending on the placement policy, rebuilding individual blocks may also require changes to the track-based rebuild algorithm as some tracks will have some of the blocks already reconstructed. Therefore, the performance impact is not obvious and more work is required to determine the applicability of such techniques.

ACKNOWLEDGMENTS

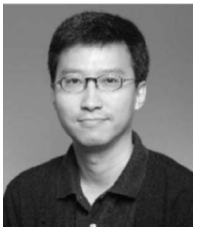
The authors would like to thank the anonymous reviewers for their insightful comments and suggestions in improving this paper. This work is partially funded by the Research Grants Council Earmarked Grant # CUHK4209/01E and a grant from the Area-of-Excellence in Information Technology established by the University Grants Council of the Hong Kong SAR Government.

REFERENCES

[1] F.A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAIDTM—A Disk Array Management System for Video Files," *Proc. ACM Conf. Multimedia '93,* pp. 393-400, Aug. 1993.
[2] S. Berson, L. Golubchik, and R.R. Muntz, "Fault Tolerant Design of Multimedia Servers," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 364-375, May 1995.
[3] A. Cohen, W.A. Burkhard, and P.V. Rangan, "Pipelined Disk Arrays for Digital Movie Retrieval," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems,* pp. 312-317, 1995.
[4] 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," *Proc. ACM Multimedia '95,* Nov. 1995.
[5] B. Özden, R. Rastogi, P. Shenoy, and A. Silberschatz, "Fault-tolerant Architectures for Continuous-Media Servers," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 79-90, June 1996.
[6] A.N. Mourad, "Issues In the Design of a Storage Server for Video-on-Demand," *ACM Multimedia Systems,* vol. 4, pp. 70-86, 1996.
[7] A. Cohen and W.A. Burkhard, "Segmented Information Dispersal (SID) for Efficient Reconstruction in Fault-tolerant Video Servers," *Proc. Fourth ACM Int'l Multimedia Conf.,* pp.227-286, Nov. 1996.
[8] J. Korst, "Random Duplicated Assignment: An Alternative to Striping in Video Servers," *Proc. Fifth ACM Int'l Conf. Multimedia,* pp. 219-226, 1997.
[9] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys,* pp. 145-186, June 1994.
[10] J. Menon and D. Mattson, "Performance of Disk Arrays in Transaction Processing Environments," *Proc. 12th Int'l Conf. Distributed Computing Systems,* pp. 302-309, 1992.
[11] J. Menon and D. Mattson, "Distributed Sparing in Disk Arrays," *Proc. 37th IEEE Computer Society Int'l Conf. (COMPCON '92),* pp. 410-421, Feb. 1992.
[12] R.Y. Hou and Y.N. Patt, "Comparing Rebuild Algorithms for Mirrored and RAID5 Disk Arrays," *Proc. 1993 ACM SIGMOD Int'l Conf. Management of Data,* pp. 317-326, May 1993.
[13] R.Y. Hou, J. Menon, and Y.N. Patt, "Balancing I/O Response Time and Disk Rebuild Time in a RAID5 Disk Array," *Proc. 26th Hawaii Int'l Conf. System Sciences,* pp. 70-79, Jan. 1993.
[14] A. Thomasian, "Performance Analysis of RAIDS Disk Arrays with a Vacationing Server Model for Rebuild Mode Operation," *Proc. 10th Int'l Con. Data Eng.,* pp. 111-119, Feb. 1994.

[15] A. Thomasian, "Rebuild Options in RAID5 Disk Arrays," *Proc. Seventh IEEE Symp. Parallel and Distributed Processing*, pp. 511-518, Oct. 1995.
[16] A. Thomasian and J. Menon, "RAID5 Performance with Distributed Sparing," *IEEE Trans. Parallel and Distributed Systems*, vol. 8, no. 6, pp. 640-657, June 1997.
[17] K. Mogi and M. Kitsuregawa, "Hot Mirroring: A Method of Hiding Parity Update Penalty and Degradation During Rebuilds for RAID5," *Proc. 1996 ACM SIGMOD Int'l Conf. Management of Data*, pp. 183-194, June 1996.
[18] D.A. Patterson, G.A. Gibson, and R.H. Katz, "A Case for Redundant Array of Inexpensive Disks (RAID)," *Proc. ACM Conf. Management of Data*, pp. 109-116, 1988.
[19] A.L.N. Reddy and J.C. Wyllie, "I/O Issues in a Multimedia System," *Computer*, vol. 27, no. 3, pp. 69-74, Mar. 1994.
[20] D.J. Gemmell, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe, "Multimedia Storage Servers: A Tutorial," *Computer*, vol. 28, no. 5, pp. 40-49, May 1995.
[21] G.R. Ganger, B.L. Worthington, and Y.N. Patt, *The DiskSim Simulation Environment Version 2.0*. Available at http://www.ece.cmu.edu/~ganger/disksim, Dec. 1999.

Jack Y.B. Lee (M '95) is with the Department of Information Engineering at the Chinese University of Hong Kong. He directs the Multimedia Communications Laboratory (http://www.mcl.ie.cuhk.edu.hk) and conducts research in distributed multimedia systems, fault-tolerant systems, multicast communications, and Internet computing. He is member of the IEEE and the IEEE Computer Society.



John C.S. Lui (M '94) received the PhD degree in computer science from UCLA. When he was a graduate student, he participated in a parallel database project in the IBM Thomas J. Watson Research Center. After his graduation, he joined a team at the IBM Almaden Research Laboratory/San Jose Laboratory and participated in research and development of a parallel I/O architecture and file system project. He later joined the Department of Computer Science and Engineering of the Chinese University of Hong Kong. His current research interests are in communication networks, distributed multimedia systems, OS design issues, parallel I/O and storage architectures, and performance evaluation theory. He is a member of the IEEE.