Design and Evaluation of Gracefully Degradable Disk Arrays*

A. L. Narasimha Reddy
IBM Almaden Research Center
650 Harry Road, K56/802
San Jose, CA 95120.
reddy@almaden.ibm.com

John Chandy and P. Banerjee
University of Illinois
1101 W. Springfield Ave.
Urbana, IL 61801.
{jchandy,banerjee}@crhc.uiuc.edu

*This work was supported in part by an IBM graduate fellowship and by a National Science Foundation Presidential Young Investigator's award NSF MIP 86-57563 PYI

Running head: Gracefully degradable disk arrays

Contact author: A. L. Narasimha Reddy,
IBM Almaden Research Center
650 Harry Road, K56/802
San Jose, CA 95120.
Phone: (408) 927-1357
FAX: (408) 927-2100

Abstract: The availability of inexpensive, small, magnetic disks, has made possible the building of a reliable, high-performance disk system by integrating a number of such disks in an array. To achieve high reliability in such systems, equivalent to that of larger disks, parity or other error-correcting codes may be used. In systems where data availability is critical, traditionally dual copy methods have been used. Recently some parity-based schemes have been proposed for providing fault tolerance with much less hardware. However, these new techniques do not provide good performance under a failure due to the increase in workload on the functional disks during a failure in the array. The dual copy methods degrade much more gracefully compared to the new techniques. In this paper, we propose a new technique for making disk arrays fault-tolerant which combines the advantages of both the parity schemes and the dual copy methods. The proposed technique offers a wide variety of options in providing fault-tolerance, dual copy methods and single parity schemes being two extreme cases. We present results from simulations to show that the proposed technique offers better performance during all phases of operation: in normal operation, during a failure and while reconstructing data on a failed disk. We also show that the proposed scheme allows faster reconstruction of data on the failed disk and thereby improves the data availability.

List of Symbols:

λ (lambda)
λ_1 (lambda subscript 1)
λ_0 (lambda subscript 0)
$\lfloor$ (left floor), $\rfloor$ (right floor)

Legends of Figures:

1 Introduction

Traditionally, large computer systems such as banking/transaction systems have used highly reliable disks in their system to ensure high data availability. These disks typically have a high data capacity and high mean time to data loss. However, these disks are very expensive. The availability of inexpensive, small, magnetic disks, has made possible the building of a reliable, high-performance disk system by integrating a number of such disks in an array. The reliability of these disks is not as high as the larger disks. Also, since the smaller disks can hold only a fraction of data of the larger disks, data need to be distributed (declustered) across multiple disks. Data declustering across multiple disks may also be done to achieve high data throughput [1, 2, 3, 4, 5]. If the data in a file are distributed on a number of disks, the failure of any disk will render parts of the file unavailable. Some kind of fault tolerance needs to be provided to ensure that the mean time between data loss is acceptably high. Mean time between data loss is defined as the mean time to an unrecoverable loss of data due to failures in the system. When data are distributed on a number of devices, as in a disk array, the mean time to data loss would be considerably shorter compared to a traditional system where the data resides on a single disk, if no mechanism for tolerating failures is provided. To provide high mean time to data loss (MTTDL) on these systems, several approaches have been proposed.

In this paper, a failure of a disk corresponds to the total loss of function of the disk. The disk failures are detected by the electronics in the device. Hence, we are only concerned with protecting data against such failures and not in detecting such failures. Disk mirroring or disk shadowing is an approach that has been traditionally used for tolerating disk failures [6]. In these systems, each disk has a shadow disk or a mirror disk which has an exact copy of the data residing on that disk. A fault tolerant system based on mirrored disks is shown in Fig. 1, where the hosts and the controllers are duplicated as well for fault tolerance. Each update to the disk also involves an update to the mirror disk. These two copies of the data are used to provide fault tolerance. If one of the disks fails, the data from the second disk can be used until the data on the failed disk are reconstructed. If the second disk also fails before the first disk is restored, the data may be lost. Disk mirroring requires 100% resource overhead to achieve fault tolerance. Variants of disk mirroring as reported in [7, 8] also suffer from the 100% overhead penalty.

The RAID approach [1] reduces this overhead considerably by employing a single parity disk. In this approach, parity of data across $n-1$ disks is stored on another disk. Whenever an update is made on one of the data disks, the parity information on the parity disk is updated. If one of the disks fails, the data on that failed disk can be obtained by an XOR operation of the operational disks. Hence this system can tolerate one failure. If another

disk fails before the failed disk is restored, data will be lost. To distribute the load more evenly, the parity data can be distributed across all the n disks in the system as shown in Fig. 2. These systems normally are provided with an extra disk as a hot spare, which is used as replacement for the failed disk. The overhead is 2 disks for n data disks in this system. Other forms of providing redundancy in disk arrays, such as 2-dimensional parity, full-2 codes, 3-dimensional parity, full-3 codes and additive-3 codes were considered in [9]. The parity striping approach [10] is another approach for achieving high data availability. This approach is similar to RAID in achieving the fault-tolerance and differs mainly in the data organization on the disks. For our discussion, the above description of these techniques is sufficient.

One of the main disadvantages of the two latter approaches, RAID and parity striping, is that when one of the disks in the system fails, the load on the functional disks doubles. To read the data from the failed disk, the data from all the other disks have to be read and XORed. When the data are to be written to the the spare disk, to compute the correct parity information, the data from all the functional disks have to be read and XORed together with the new data being written. Hence, for each read and write to the failed disk, each of the functional disks sees a corresponding read/write operation. If initially the system is load balanced, under a failure the functional disks would observe twice as many requests. Hence, the functional disks see a 100% increase in their load under a single failure in the system. The increased workload on the system could lead to further degradation of the reliability of the system as reported in [11, 12], where it was shown that increased I/O rates correlated with increased failure rates in the system. To avoid unacceptably long delays in service and possible degradation in reliability due to higher loads, these systems then would have to operate under less than 50% utilization. The 100% load increase, even though for short periods of time, imposes severe restrictions on the system design if the system has to be available continuously. With mirroring, when one disk fails, the read load on the disk doubles, but the write load is not affected. When a disk fails, the mirror sees a request traffic of $w + r$ instead of the usual request traffic of $w + r/2$. If the system normally sees equal read and write traffic, the net change in load is typically a 33% increase on one disk rather than a 100% increase on $n - 1$ ($n > 10$, typically) disks, as in RAID and parity striping.

This is the problem we address in this paper: how do we design fault-tolerant disk arrays that are not susceptible to 100% load increases on the functional disks when one of the disks in the system fails? In other words, how do we design gracefully degradable disk arrays? Section 2 outlines the basic approach we adopt to solve this problem. Sections 3, 4 and 5 describe constructive solutions based on different techniques. Section 6 looks at the problem of reconstruction of data and the reconfiguration of the system under a failure. Section 7 presents simulation results to show the advantages of the proposed technique. Section 8

summarizes the results reported in the paper.

2 Basic Approach

Definition 1 *The* disk array width *of a system is defined as the total number of disks, including the failed disk, that are affected due to a single failure.*

Definition 2 Parity group length *is defined as the total number of disks, including the parity disk, that are in a single parity group.*

The basic approach we adopt is to allow a disk to belong to more than one parity group. Specifically, we consider the case of allowing two parity groups. Depending on the track number (or data block number), the disk belongs to either the first parity group (denoted 0) or the second parity group (denoted 1). It is to be noted that each update to a disk results in updating only one parity disk depending on which parity group the updated block on that disk belongs to. This is to be contrasted with other approaches such as 2-dimensional parity where each disk belongs to two parity groups and each update to the disk results in updates at two parity disks. In our system, if a disk fails, the data on that disk interact with all the other disks in the system depending on the block number even though for a particular block the disk may interact with only some of the disks in the system. Hence, the array width is still equal to the total number of disks in the system. Hence, the effective reliability of the system is not changed. However, this arrangement of allowing the disk to belong to two parity groups has a number of desired properties as will be shown later.

The disks in a single parity group are said to interact with each other since the failure of any disk in the group requires that the data from the other disks be read for reconstructing the failed data. If a block of data needs to be read from disk A for every reconstruction of a block of data on a failed disk B, then disks A and B are said to interact 100%.

Definition 3 *The fraction of interaction between two disks is the percentage of blocks of a disk that need to be read for reconstructing the blocks on the other disk.*

By utilizing multiple parity groups in a single array, we can control the interaction between the various disks in the array by proper mapping of the disks into the different parity groups. When the parity group length is smaller than the array width, by proper choice of these parity group mappings, we can potentially reduce the interaction to the ratio of $(l-1)/(w-1)$, where l is the parity group length and w is the array width. This is the intuitive idea behind our approach. The main problem here is to obtain the required mapping of the disks into parity groups to reduce the interaction between different disks.

We set forth the following conditions on the desired design: (a) less than 50% load (of load before the failure) increase on the functional disks with a single failure; (b) fast identification of the parity group to which a disk belongs; (c) fast identification of the parity disk in that group. (d) balanced load during normal operation and (e) balanced load under a single failure. The designs should be as general as possible, i.e., they should be applicable in as many cases as possible. For the discussion below, we assume that the parity information is stored on a block basis, the block size normally being a track. We use block and track interchangeably in this paper to mean the unit of data on which parity is computed. The results presented here restrict the load increase to be less than 50%. The results can be generalized to other cases when the load increases have to be restricted to smaller percentages.

To illustrate the basic ideas, we present an example in Fig. 3. The 0's and 1's represent the parity group to which each disk belongs for a particular block number. The bold 0's and 1's represent the parity disks for each row. The txn matrix of 0's and 1's, indicating the parity group information of the disks for different tracks, is called the *code matrix*, where n is the number of disks in the array and t is called the *code size*. If a data block is written to track number y, then $y \bmod t$ (t = code size) is used to index into the code matrix to obtain the parity group organization for that particular track. It is assumed in this paper that the code matrix is stored in memory (rather than computed). Whenever the parity group information of a track on a disk is needed (during writes or reconstruction), this code matrix is referred to obtain that information. It is necessary that the code sizes be short. If the code size is too large, storing the code matrix will be an expensive option. The code size t should be a function of the number of disks, rather than the number of data blocks in the system. The code shown in Fig. 3., has the desired properties (b), (c) and (d), but not (a) and (e). As shown in Fig. 3, when the disk marked X fails, the increase in load is proportional to the proximity of the other disks to the failed disk. The numbers next to X in different columns indicate the relative frequency at which the failed disk interacts with different disks. This example is presented to show that a simple regular distribution of disks into two parity groups does not yield the required solution and to illustrate some of the concepts involved in the code design. Below, we present a general framework for a solution to the proposed problem. The same problem has been independently considered in [13] where the main emphasis was on minimizing the reconfiguration time after a failure. Muntz and Liu [13] conjecture the use of block designs for solving the same problem, but a complete solution is not presented. The solution presented here is based on our earlier independent work [14].

3 Block Designs

In this section, we show that the required solution to the disk array problem is a formulation of *block designs*. A balanced incomplete block design is an arrangement of v distinct objects into b blocks such that each block contains exactly k distinct objects, each object occurs in exactly r different blocks, and every pair of distinct objects a_i, a_j occurs together in exactly λ blocks [15] . When $v = b$, $r = k$, the block design is said to be symmetric and is denoted by a 3-tuple (v, k, λ). There are two elementary relations on the five parameters for a block design:

$$bk = vr \tag{1}$$

$$r(k-1) = \lambda(v-1) \tag{2}$$

The first relation counts the total number of incidences in two ways, b blocks each containing k objects and v objects being incident each in r blocks. The second relation counts the occurrences of pairs containing a particular object. Each object interacts with $v-1$ other objects λ times each and it interacts with $k-1$ other objects in each of the r blocks it is incident in. A block design is described by its incidence matrix. This is a matrix $A = (a_{ij})$, $i = 1, \ldots,$ v, j= $1, \ldots, b$, where if $a_1, \ldots, a_v$ are the objects and $B_1, \ldots, B_b$ are the blocks, we have

$$a_{ij} = 1, \text{ if } a_i \in B_j,$$

$$= 0, \text{ otherwise.}$$

The code matrix that needs to be derived for our problem is similar to the incidence matrix of the block design. The parity group 1 in ith track can be seen as the a_{ij} that are equal to 1 and similarly, the parity group 0 can be seen as the a_{ij} that are 0. Hence, the problem of dividing the disks into two parity groups can be solved by an appropriate block design. The parity group 1 indicates the incidence of objects in the blocks and the parity group 0 indicates the nonincidence. The λ parameter indicates only the number of times a pair of disks occurs together in a group and hence the interaction in parity group 1. However, we are also interested in the number of times the disks occur together in parity group 0. These two numbers added together will indicate the frequency at which a pair of disks interact with each other in the same parity group. We shall denote these two numbers by λ_1 and λ_0. When $v = b$, the block design is said to be symmetric. An example code matrix based on block designs is shown in Fig. 4. In the example with 7 disks, each pair of disks interacts in exactly 3 out of 7 data blocks. This can be verified by looking at each pair of columns and counting the common incidences in parity groups 0 and 1. The figure shows

the interaction of disk 3 with all the other disks. Location of parity disks is not shown in Fig. 4 and later code matrices since the design focus is to reduce the mutual interaction of the disks in parity groups irrespective of whether they contain parity information or data. The code matrix however has to contain this information.

Block designs or solutions based on Hadamard matrices and difference sets (described later) exist only for some specific values of v or b. However, this is not a problem for the disk array design. If a disk array with n disks needs to be built, all we need to do is to find a solution for any value of $m \geq n$. We can throw away the excess $(m-n)$ columns in such a design to obtain an mxn code matrix for n disks. Hence, the approach of using block designs or the later described techniques can be applied for any number of disks. Looking at the example in Fig. 4 again, if we throw away a column, we can obtain a solution for a 6-disk system with a 7x6 code matrix with each pair of disks still interacting in 3 out of 7 tracks.

If we count the incidences of any pair of disks in the two parity groups, then the following relation should be satisfied:

$$\lambda_1 + 2 * (k - \lambda_1) + \lambda_0 = b, \tag{3}$$

where λ_1 = number of common incidences or the number of blocks in which the two disks interact in parity group 1, $2 * (k - \lambda_1)$ = number of disjoint incidences of the two disks in parity group 1 and λ_0 = number of common incidences of the two disks in parity group 0. The sum of the above three terms should equal the total number of disks = b. Hence, we have $\lambda_0 = b - 2 * k + \lambda_1$. The frequency of interaction of a pair of disks is given by $\frac{(\lambda_1+\lambda_0)}{b}$.

Lemma 1 *A pair of disks occur together in both the parity groups at less than 50% frequency, i.e., in less than half the number of the tracks, if* $\frac{b-\sqrt{b}}{2} \leq k \leq \frac{b+\sqrt{b}}{2}$.

Proof: The frequency of interaction between two disks is given by $\frac{\lambda_1+\lambda_0}{b} = \frac{\lambda_1+b-2k+\lambda_1}{b}$. For a symmetric block design, $v = b$, $r = k$ and from relation (2) we have $\lambda_1 = \frac{k(k-1)}{(b-1)}$. Substituting the value of λ_1 in $\frac{\lambda_1+\lambda_0}{b}$ and imposing the condition of 50% interaction, we arrive at the following inequality:

$$b - 2k + \frac{2k(k-1)}{(b-1)} \leq b/2.$$

The values of k that satisfy the inequality are given by

$$\frac{b-\sqrt{b}}{2} \leq k \leq \frac{b+\sqrt{b}}{2}.$$

□

Corollary 1 *When $v = b$ and $r = k = \lfloor b/2 \rfloor$, the frequency of interaction between a pair of disks is $\leq$ 50%.*

Lemma 2 *In an n-disk array system, when two parity groups are used, the minimum fraction of load increase due to a failure is given by $(n-2)/2(n-1)$ and this is possible when the lengths of the two parity groups are equal or differ by at most 1.*

Proof: Assume that the two parity groups are of length n_0 and n_1 such that $n_0 + n_1 = n$. Whenever a block of data on the failed disk is needed, the remaining disks in the parity group, to which the block of data belongs, have to read data. The data on the failed disk belongs to parity group 0 in $n_0/(n_0 + n_1)$ fraction of the blocks and to parity group 1 in $n_1/(n_0 + n_1)$ fraction of the blocks Hence, the increased number of accesses on the total system for each access to the failed disk is given by

$$\frac{(n_0 - 1)n_0}{n_0 + n_1} + \frac{(n_1 - 1)n_1}{n_0 + n_1}.$$

Hence, the load increase on each disk is given by

$$\frac{1}{(n-1)}\left(\frac{(n_0 - 1)n_0}{n_0 + n_1} + \frac{(n_1 - 1)n_1}{n_0 + n_1}\right).$$

This value is minimized when $n_0 = \lfloor n/2 \rfloor$. Hence, the lemma follows. □

Lemma 3 *When m parity groups are used in a design, the minimum load increase on the disks in an n-disk array is given by $\frac{n-m}{m(n-1)}$.*

Proof: Using a similar argument as in Lemma 2, the total interaction between the disks can be shown to be minimum when all the parity groups are of equal size or differ at most by 1 in size. For simplicity, consider the case when all the parity groups are of equal size. Then the total interaction among the disks is given by $m * \begin{pmatrix} n/m \\ 2 \end{pmatrix}$, since all the pairs of disks in a parity group (of size n/m) interact and there are m parity groups. Since there are totally $\begin{pmatrix} n \\ 2 \end{pmatrix}$ pairs of disks, the frequency of interaction of a pair of disks is at least $\begin{pmatrix} n/m \\ 2 \end{pmatrix} * m / \begin{pmatrix} n \\ 2 \end{pmatrix}$. This simplifies to $(n-m)/(m*(n-1))$. □

The above lemma tells us that the best we can do by using m parity groups is to limit the load increase due to a failure to $(n-m)/(m*(n-1))$.

Corollary 2 *Symmetric* $(v, \lfloor v/2 \rfloor, \lambda)$ *block designs minimize the load increase during a failure.*

Proof: From Lemma 3, for the case of two parity groups or when $m = 2$, the load increase can at best be limited to $(n-2)/(2*(n-1))$. In Lemma 2, we showed that the block designs achieve this result. Hence, solutions based on block designs are the best possible with respect to limiting the load increase during a failure. □

Code matrices based on symmetric block designs have all the desired properties (a)-(e). The code matrix is small, nxn, n being the number of disks in the array. The small code matrix can be stored in memory and retrieved fast to satisfy requirements (b) and (c). The system is load balanced in normal operation since each disk has parity data of both the parity groups 0 and 1 in a code matrix. By distributing the parity information uniformly over all disks, we guarantee that the write update traffic is uniform over all disks. Hence, the system is load balanced under normal operation. Under a failure, each disk sees at most a 50% increase in load since every pair of disks interacts only in at most 50% of the tracks. The system is load balanced under a single failure since the load increase due to failure is uniform across all the disks.

The proposed methodology of using multiple parity groups can be generalized to consider more than two parity groups to obtain solutions that restrict load increases on functional disks to percentages lower than 50%. When we require that all the disks belong to one parity group, we obtain the RAID solution. When we require that there be $n/2$ parity groups, when there are n disks, we obtain the mirroring approach. Several possible solutions exist between these two extremes. We highlighted one of those solutions with two parity groups.

The constructive techniques we present here are limited to two parity groups even though our techniques can be generalized to larger number of parity groups. More parity groups may be used to reduce the observed load overhead during a failure to below 50%. The block designs approach can be employed for more than two parity groups by choosing the k value appropriately in the block design to be n/m, where m is equal to the number of parity groups and n is the total number of disks. When more than two parity groups are considered, only the block incidences are considered as a parity group (unlike in the 2 parity group case, where nonincidences constituted the second parity group) and a number of parity groups are packed into a row of data blocks. An example using 3 parity groups and nine disks is shown in Fig. 5. In the example shown in Fig. 5, the load increase due to a failure is limited to 25%, the best achievable according to Lemma 3.

The block designs give a general framework for our problem. However, the disk array problem looks for a particular block design, namely, the symmetric $(v, \lfloor v/2 \rfloor, \lambda)$ block design. Though $k = \lfloor v/2 \rfloor$ is not a strict condition, finding block designs with such parameters is

easier. Also, from Lemma 2, this choice gives the best performance when two parity groups are considered. When there are simple constructive techniques for block designs with other parameters, they are mentioned. Also, the general framework does not give any indication about constructing such block designs. Below, we present two different ways of constructing such specific block designs based on *Hadamard matrices* and *difference sets* .

4 Hadamard Matrices

A Hadamard matrix of order m is an $m \times m$ matrix H of +1's and -1's such that $HH^T = mI$. This implies that any pair of columns have matching digits in exactly half the columns and differing digits in exactly half the columns. A Hadamard matrix of size 2x2 is shown in Fig. 6 which also shows how a Hadamard matrix of degree 2^k can be obtained from a Hadamard matrix of degree 2^{k-1}. Hadamard matrices constructed using such a procedure have an equal number of +1's and -1's in each column except for the first column. Given a Hadamard matrix, we can always find one equivalent to it whose first row and first column consist entirely of +1's. Such a Hadamard matrix is called "normalized." If we throw away the all-ones column and all-ones row from the Hadamard matrix, we arrive at a code that we shall term "Hadamard-derived" code. Such a code of degree 7 is shown in Fig. 7, where -1's of the Hadamard matrix are replaced by 0's in the H-derived code. It can be shown that if H is a Hadamard matrix of order $m > 2$, then m is necessarily a multiple of 4 [15] .

Lemma 4 *From a normalized H matrix of order $m = 4t$, we may construct a symmetric block design D with parameters $v = 4t-1$, $k = 2t-1$, $\lambda = t-1$, and conversely, from such a design D we may construct a normalized H.*

Proof: Refer to Hall [15]. □.

Our interest in H matrices arises from the above statement. Since the symmetric block design D corresponds to a solution to our disk array problem as shown earlier, any normalized H-matrix would give a solution to the disk array problem. The possible existence of Hadamard matrices for all multiples of 4 indicates that the code matrix for an n-disk array need not be any larger than $(n+3)$xn. Codes derived by throwing away the all-ones row and all-ones column from a normalized H-matrix gives a system that is load balanced. Properties (a) and (b) are satisfied by having a small code size = n = number of disks in the system. Due to the equivalence of Hadamard matrices and the block designs, H-derived codes also satisfy the other requirements (c), (d) and (e).

It is also possible to construct block designs from H matrices in another way. Without going into further mathematical details here, we can obtain block designs with parameters

$v = 4u^2$, $k = 2u^2 - u$, and $\lambda = u^2 - u$ [15]. Such designs exist and we show an example of this in Fig. 8, for $u = 2$ or $v = 16$. Note that these parameters are within the limits specified by Lemma 1 and hence satisfy our requirements (a)-(e). There are a number of methods for constructing Hadamard matrices. Interested reader is referred to [15] for other constructive techniques.

5 Difference Sets

Another method of constructing the required block designs is through *difference sets.*

Definition 4 *A set of k residues D: $a_1, \ldots, a_k$ modulo v is called a (v, k, λ)-difference set if for every $d \neq 0$ (mod v) there are exactly λ ordered pairs (a_i, a_j), a_i, $a_j \in D$ such that a_i-$a_j = d$ (mod v).*

For example, 5,6,8,1 is a (13,4,1) difference set as shown below: (6-5) mod 13 = 1, (8-6) mod 13 = 2, (8-5) mod 13 = 3, (5-1) mod 13 = 4, (6-1) mod 13 = 5, (1-8) mod 13 = 6, (8-1) mod 13 = 7, (1-5) mod 13 = 9, (5-8) mod 13 = 10, (6-8) mod 13 = 11, and (5-6) mod 13 = 12. The relation between difference sets and block designs is established through the following lemma [15] :

Lemma 5 *A set of residues D: $a_1, \ldots, a_k$ modulo v is a (v, k, λ)-difference set if and only if the sets B_i: $a_1 + i$, $a_2 + i$, ...,$a_k + i$ modulo v, $i = 0, \ldots, v$-1 are a cyclic (v, k, λ) block design B.*

A symmetric (v, k, λ) block design is called B-cyclic if B has an automorphism α that permutes the objects and also the blocks in a cycle of length v.

One advantage of difference sets is that if the difference set is specified, i.e., the incidence of the objects in one block is specified, the block design is completely specified. Hence, the code matrix can be compressed to a single row. For example, with the (13,4,1) difference set 1,5,6,8 (0100011010000), the incidence matrix of the block design can be simply obtained by circularly rotating the above row. A second advantage of the difference sets approach is that once we specify the location of parity disks in a row, we can obtain the location of parity disks in other rows by simple rotation. This guarantees that the parity data is uniformly distributed across all the disks in the system.

A number of constructive techniques exist for difference sets. We list one of these techniques that has the required parameters of our problem, $v = 4t-1$, $k = 2t-1$ and $\lambda = t-1$: Quadratic residues in GF(p^r) (Galois field of dimension p^r), $p^r = 3 \pmod 4$ and $v = p^r$. The integers $b \not\equiv 0 (\mathrm{mod} p)$ are divided into two classes called quadratic residues and quadratic

nonresidues according as $x^2 = b(\text{mod } p)$ does or does not have a solution $x(\text{mod } p)$. For example, modulo 7, 2 is a quadratic residue since $4^2 = 2(\text{mod } 7)$, and 3 is a quadratic nonresidue since $x^2 = 3(\text{mod } 7)$ does not have a solution. The above technique tells us that quadratic residues *modulo* p^r form a difference set. Consider for example, generating a difference set modulo 7. The quadratic residues in this field are 1, 2 and 4. Hence a code for 7 disks can be expressed as 0110100, whose complete code matrix is obtained by rotating this row circularly. The complete code matrix is the same as the one we considered earlier in Fig. 4. Other constructive techniques are more complicated and we will not dwell on the mathematical details any further. It suffices to say that the theoretical framework of the problem gives a number of constructive techniques for a solution.

6 Reconstruction

In this section, we will discuss how the data on the failed disk are reconstructed and how the system is reorganized under a failure. Below, we present two scenarios for reorganization.

If an extra disk is used as a hot spare (besides the two parity disks), the reconstruction is simple. Each block of data on the failed disk is reconstructed by a simple XOR of the elements in the parity group to which the disk belongs. The reconstructed data are written onto the hot spare and the system is reconfigured to be like the original design before failure. The references to data on the failed disk are simply diverted to the spare disk. For this system, the total number of extra disks used is then equal to 3. This is one higher than in approaches such as the RAID and parity striping. If the extra disk is too much overhead to pay, the following alternative can be considered.

If an extra disk is not used as a hot spare, the reconstruction algorithm is a little complicated. The parity groups are merged into one parity group under a failure to look like a single parity system as in the RAID approach. The system is configured differently under a failure compared to the original design. This overhead in system time is hard to quantify. However, this system does not have any other hardware overhead and uses only two extra disks as in the other approaches. We discuss below how this reconfiguration may be carried out.

The spare disk serves no purpose during regular operation of the system in the earlier approaches. In our approach, we use the spare disk also for parity purposes as will be explained below. For ease of explanation, first consider the RAID approach where the parity information resides on a single disk. Consider a system with 4 disks, a parity disk and a hot spare. This system can be configured as two subsystems each with 2 data disks and a single parity disk. Notice that the hot spare is being used as parity disk for the second group.

This is shown in Fig. 9. The system's reliability is improved with the new configuration since the array width is now smaller. However, the new configuration does not have a hot spare. Under a failure, the two separate parity groups are merged into a single group as shown in Fig. 10. If a data disk fails, the data on the failed disk are reconstructed onto the parity disk of the same group and the parity information on the other group is modified to reflect the parity of both the groups of disks. If the parity disk fails, the parity data in the other group is modified to contain the parity data of both the groups together. The reconfigured array would be as shown in Fig.10. Note that disks belonging to the failed-disk group still experience 100% load increase. However, the number of disks experiencing such a load increase is now smaller since the number of disks in each parity group is smaller. Hence, by utilizing the spare disk during the regular operation of the system, we have achieved two benefits: improved the reliability of the system and the load increase under a failure is now limited to fewer disks in the system.

Now consider a RAID5 system where parity is distributed across all the disks as shown in Fig. 11. We can use a similar idea of reconfiguring the system into one parity group under a failure. The reconfigured array by merging the two parity groups is shown in Fig. 12. On a particular track, if the the failed disk contains parity information, this parity information (after reconstructing from the data disks belonging to this group) is merged with the parity information of the second group. If the failed disk contains data on a particular track, the parity information of both the groups is again merged and the data on the failed disk are reconstructed onto the parity areas of either group. This idea of merging two parity groups can be generalized for other cases. When block designs are used, a similar procedure can be employed. However, the parity groups in each row of the original code matrix span different disks. Hence, to obtain a simple mechanism for reconfiguration, we treat reconfiguring each row of the original code matrix separately.

The idea of merging parity groups as a strategy for reconfiguration gives a number of possible tradeoffs. First, if a hot spare is needed in a single array system, the hot spare can be used during regular operation as a second parity disk to improve the performance of the system. This option uses no extra hardware, but by utilizing better data organization and the otherwise idle spare disk, improves the system performance during a failure and ensures the reliability of the system to be high. It can be argued that using the spare disk during normal operation enhances the system's reliability since detection of a failure of an otherwise idle spare disk is not possible. Second, a hot spare disk may not be needed if the system consists of multiple parity groups. In a multi-array system, it is possible to completely do away with hot spares. By automatically reconfiguring under a failure, the service cycle can be extended i.e., the service person does not have to visit the installation immediately after a single failure. Additionally, if block designs are used, the increased load immediately after

a failure and before reconfiguration can also be limited to tolerable levels.

7 Evaluation through simulations

In this section, we present results obtained through simulations to show the advantages of the proposed techniques.

7.1 Simulation model

We constructed a detailed disk array simulator using CSIM [16]. The configuration of each disk array is specified through a code matrix. This matrix identifies the disks in an array and also the location of parity blocks in the array. Each system may also specify if it has any spare disks.

For the results reported here, we modeled two systems. The first system has 14 data disks and 2 other disks. In the scheme we proposed here, these 2 other disks are used as 2 parity disks. The particular configuration for the proposed scheme is as shown in Fig. 8. For comparison, we simulated a RAID5 system with 14 data disks, 1 parity disk and 1 hot spare. The second system has 7 disks. For the proposed scheme, the configuration is as shown in Fig. 4.

Requests to the disk system are assumed to be random and uniformly distributed over the disk. The disk parameters used in this study are shown in Table i. We used a nonlinear model reported in [17] to model the seek time cost function. Each request asks for a single track of data. When the I/O system has a cache, performance of the I/O system may be optimized by transferring a block of data that may be different from the actual request size [18, 19, 17]. We chose this block to be a track in the simulations reported in this paper. The requests were chosen to be read requests with 70% probability and writes with 30% probability. Request rates are varied from 25 to 200 I/Os/sec.

Each system is simulated during normal operation and failure mode operation and during reconstruction phase. Failure mode operation is the time during which a disk has failed but the reconstruction has not been started. During reconstruction, the disk blocks on the failed disk are reconstructed and written to the spare space on the disk. In a RAID5 scheme, the spare disk serves as the spare space and in our scheme, the second parity serves as the spare space. Reconstruction is assumed to be done in the following way: the blocks on the failed disk are reconstructed serially. During the reconstruction phase, normal requests are given a higher priority than the reconstruction requests. If a request asks for a block of data on the failed disk, that block is reconstructed by doing proper exclusive-OR of the blocks in

its parity group. So reconstructed block is returned to the requester and also written to the spare space. This type of reconstruction is termed reconstruction with piggy-backing in [13]. In our model, we simulated this by providing two separate queues at each disk. The requests in the reconstruction queue are served only when there are no requests waiting in the normal queue. When a normal request arrives at a disk, if the disk is busy serving a reconstruction request, then the normal request waits till that reconstruction request is served i.e., service is non-preemptive. During reconstruction phase, the reconstruction queues are kept non-empty to complete the reconstruction as fast as possible.

The response time of normal requests during different phases of operation, normal mode, failure mode, and reconstruction mode, are measured. In our simulations, we declared disk 0 to be faulty for studying the failure mode and reconstruction modes of operation. The time to reconstruct the data on the failed disk to the spare space is also measured. During this rebuild time, if another failure occurs, we may lose data. Hence it is important to keep the time for reconstructing the data on the failed disk as low as possible. We present the results obtained through simulations in the next section to show that the proposed scheme offers advantages over the RAID scheme.

7.2 Simulation results

7.2.1 Normal-mode response time

Fig. 13 shows the response time of the two schemes during normal operation. The RAID scheme with one hot spare has a slightly higher response time than the proposed scheme. The proposed scheme makes use of all the disks(including the spare) in the system. During normal operation, the requests are served by $m + 1$ active disks compared to m in the RAID scheme. This results in slightly improved performance in the proposed scheme. This resulted in an observed response time improvement of 5% in System 1 at 200 I/Os/sec and an improvement of 11.2% in System 2 at 66.7 I/Os/sec. This improvement in response time is a direct result of making use of an otherwise spare disk during normal operation.

7.2.2 Failure-mode response time

Fig. 14 shows the response time of the two schemes when a disk in the system has failed and before the reconstruction process has been activated. The proposed scheme offers improved performance than the RAID scheme; in System 1, at 200 I/Os/sec, the response time is reduced by 54.7% in our scheme compared to that of RAID and in System2, at 66.7

I/Os/sec an improvement of 43.8% is observed. The performance improvement becomes more prominent at higher request rates. At 200 I/Os/sec request rate, the response time of the system employing the RAID scheme degrades from about 66 msec. to 200 msec. At the same request rate, the proposed scheme's performance degrades from about 63 msec. to only about 90 msec. This confirms our analysis that the proposed scheme should be better at handling the high request rates due to a failure. Fig. 14 validates the primary design goal of our scheme by the observed reduction in response times during a failure.

7.2.3 Reconstruction time

Fig. 15 shows the time taken to completely reconstruct the data on the failed disk to spare space. This time is an important metric since if another failure occurs during this reconstruction time, some of the data on the disk system will become unavailable. Smaller the reconstruction time, the higher the availability of the system. The proposed scheme requires less time to reconstruct the data on the failed disk than the RAID scheme; for example, in System 1, the reconstruction time is reduced from 1364 seconds in RAID scheme to 1056 seconds in our scheme, a reduction of about 22.6%. During a failure, the normal request load on each disk is lower in the proposed scheme than that in a system employing the RAID scheme. As a result, the disks can serve the reconstruction requests faster. This results in shorter reconstruction times. It is also observed that the proposed scheme is more effective with larger parity groups in reducing the reconstruction times.

7.2.4 Response time during reconstruction

Fig. 16 shows the response time of normal requests during the reconstruction phase. The proposed scheme offers better response times than the RAID scheme; in System 1, at 200 I/Os/sec, the average response time is improved by 27% and in System 2, at 66.7 I/Os/sec, an improvement of 18% in response times is observed. This again validates that our scheme has successfully met the design goals set forth earlier of reducing the impact on the request response time due to failures. This also shows that the proposed scheme does not sacrifice the response times of normal requests to reduce the reconstruction time. From figures 13, 14, and 16, it is observed that the proposed scheme has better performance than the RAID scheme during all phases of operation.

7.3 Discussion on results

In this section, we showed that the proposed scheme performs as predicted through analysis in earlier sections. Through simulations, it is demonstrated that the proposed scheme offers better response times during all phases of operation of an array. It is also shown that the proposed scheme results in smaller reconstruction periods and hence offers higher data availability. It was shown that the impact of the proposed scheme on array reconstruction times is more significant with larger parity groups and at higher workloads. All these performance benefits are achieved through the exploitation of idle hardware and through better data layout.

8 Summary

In this paper, a new technique for providing fault-tolerance in disk arrays has been presented. It was shown that by appropriately dividing the disks into two parity groups, the load increase on the functional disks under a failure can be limited to less than 50%. A theoretical framework for solving the problem is presented. A number of constructive techniques have been proposed. It is also shown that by utilizing the same amount of hardware as the earlier methods, but through a better data organization and a different reconstruction technique the system can achieve better performance during a failure. The idea of merging two parity groups as a reconfiguration strategy is shown to have a number of benefits such as reduced hardware overhead and improved reliability. Simulation results show that a combination of block designs and the proposed reconfiguration strategy yield a highly reliable disk array with same or less overhead as the earlier approaches and better performance during all modes of operation. It is also shown that the proposed scheme offers faster reconstruction of data on the failed disk and thereby improves the availability of the data in the system.

Acknowledgements

Discussions with Dick Mattson, Spencer Ng and Jai Menon at IBM Almaden Research Center have helped to clarify the reconfiguration strategy presented in Section 6. Referees' comments have helped in improving the quality of the presentation.

Table i. Disk parameters.

Avg. latency	8.3 ms
Avg. seek	14.2 ms
sectors/track	52
sector size	512 bytes
tracks/cylinder	14
cylinders/disk	1258

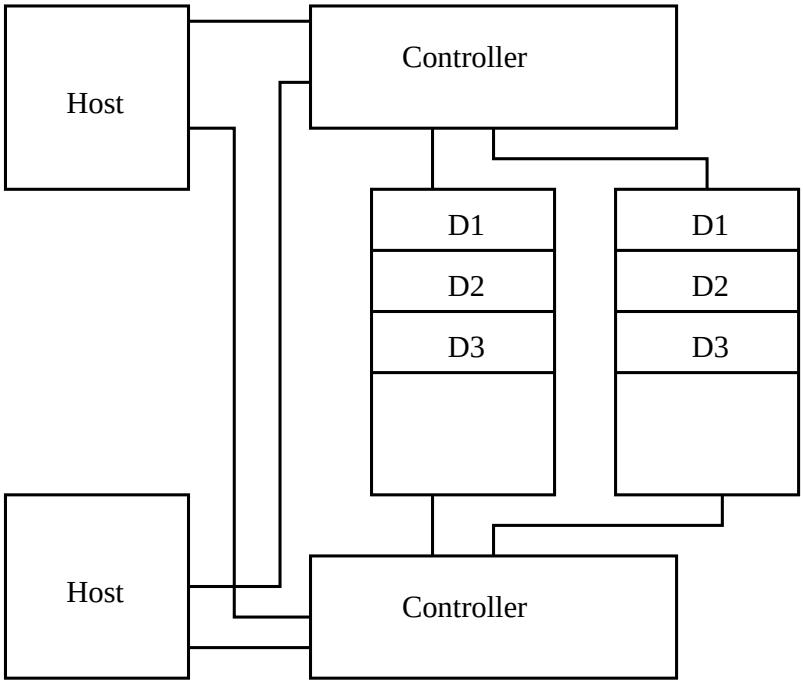

Fig. 1. A Disk-Mirror System.

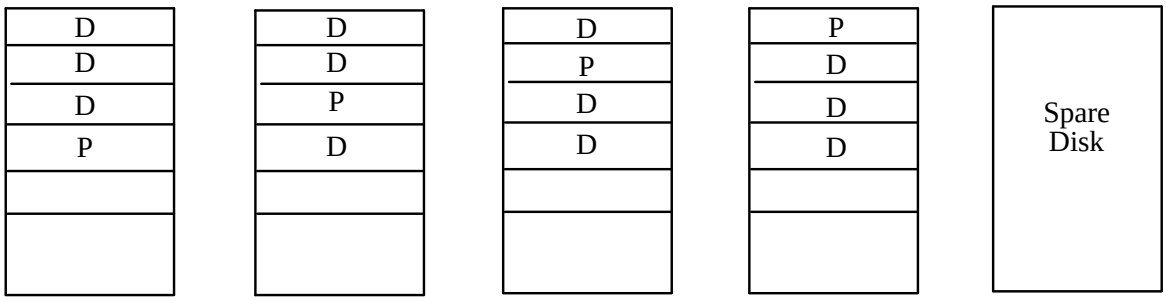

Fig. 2. The RAID Approach.

		Disk #						
		1	2	3	4	5	6	7
T	1	**0**	0	0	0	**1**	1	1
R	2	1	**0**	0	0	0	**1**	1
A	3	1	1	**0**	0	0	0	**1**
C	4	**1**	1	1	**0**	0	0	0
K	5	0	**1**	1	1	**0**	0	0
	6	0	0	**1**	1	1	**0**	0
#	7	0	0	0	**1**	1	1	**0**
		X	5	3	1	1	3	5

Fig. 3. Circular Double Parity.

	i object (disk)						
	1	2	3	4	5	6	7
	1	1	0	1	0	0	0
	0	1	1	0	1	0	0
j	0	0	1	1	0	1	0
block	0	0	0	1	1	0	1
track	1	0	0	0	1	1	0
	0	1	0	0	0	1	1
	1	0	1	0	0	0	1
	3	3	X	3	3	3	3

Fig. 4. An Example Block Design with $v = b = 7$, $r = k = 3$, $\lambda = 1$.

$$
\begin{array}{ccccccccc}
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 \\
1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 \\
0 & 1 & 2 & 1 & 2 & 0 & 2 & 0 & 1 \\
2 & 0 & 1 & 0 & 1 & 2 & 1 & 2 & 0 \\
1 & 2 & 0 & 2 & 0 & 1 & 0 & 1 & 2 \\
0 & 1 & 2 & 2 & 0 & 1 & 1 & 2 & 0 \\
2 & 0 & 1 & 1 & 2 & 0 & 0 & 1 & 2 \\
1 & 2 & 0 & 0 & 1 & 2 & 2 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 \\
2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}
$$

Fig. 5. An example with 3 parity groups.

$$
\begin{aligned}
H_2 &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\
H_k &= \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix}
\end{aligned}
$$

Fig. 6. Hadamard Matrices.

$$
\begin{array}{ccccccc}
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0
\end{array}
$$

Fig. 7. An H-Derived Code.

$$
\begin{array}{cccccccccccccccc}
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

Fig. 8. A Hadamard Matrix of Order 16.

$$
\begin{array}{ccccccc}
D_1 & D_1 & D_1 & P_1 & D_2 & D_2 & P_2 \\
D_1 & D_1 & D_1 & P_1 & D_2 & D_2 & P_2 \\
D_1 & D_1 & D_1 & P_1 & D_2 & D_2 & P_2 \\
X & & & & & &
\end{array}
$$

Fig. 9. Two arrays before failure.

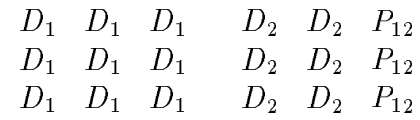

Fig. 10. Reconfigured array by merging.

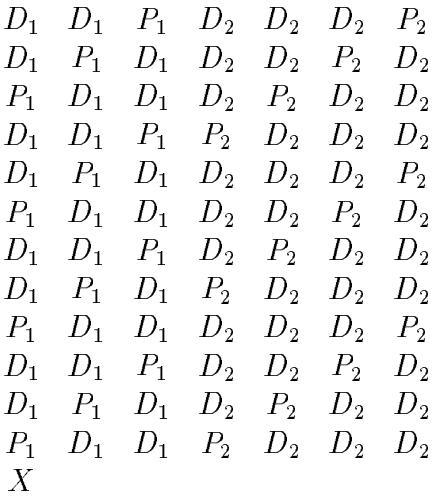

Fig. 11. Two parity groups with rotated parity.

D_1	D_1	D_2	D_2	D_2	P_{12}
D_1	D_1	D_2	D_2	P_{12}	D_2
D_1	D_1	D_2	P_{12}	D_2	D_2
D_1	D_1	P_{12}	D_2	D_2	D_2
D_1	D_1	D_2	D_2	D_2	P_{12}
D_1	D_1	D_2	D_2	P_{12}	D_2
D_1	D_1	D_2	P_{12}	D_2	D_2
D_1	D_1	P_{12}	D_2	D_2	D_2
D_1	P_{12}	D_2	D_2	D_2	D_1
D_1	P_{12}	D_2	D_2	D_1	D_2
P_{12}	D_1	D_2	D_1	D_2	D_2
P_{12}	D_1	D_1	D_2	D_2	D_2

Fig. 12. Reconfigured parity group of Fig. 11.

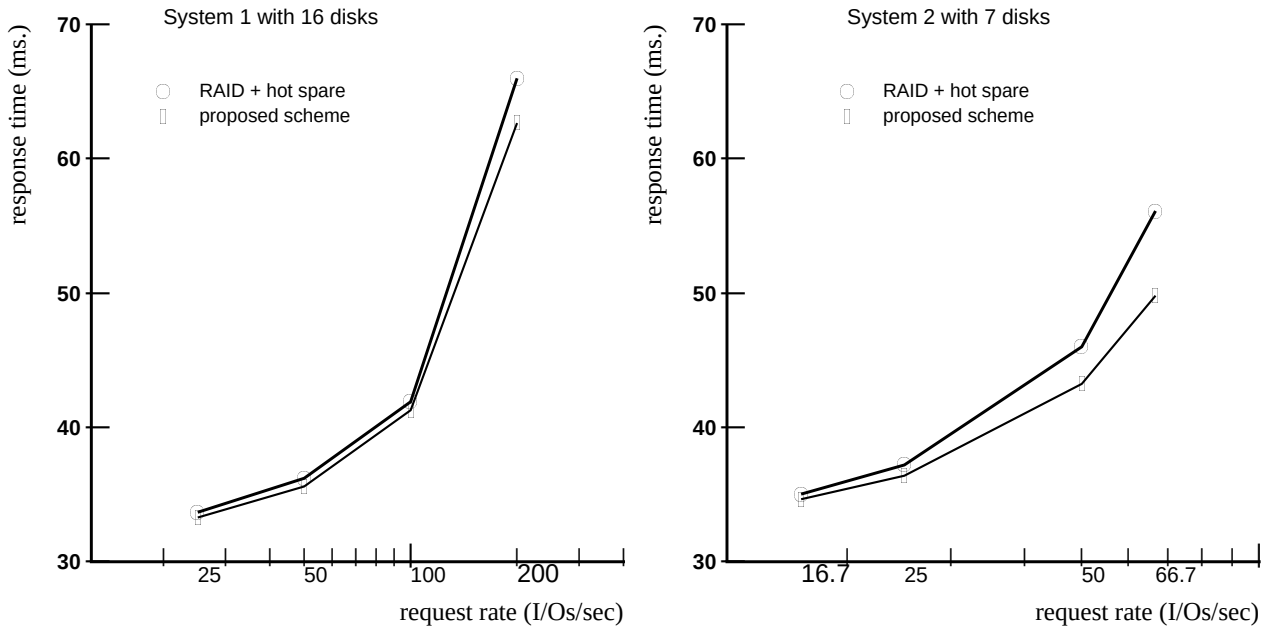

Fig. 13. Response time during normal operation.

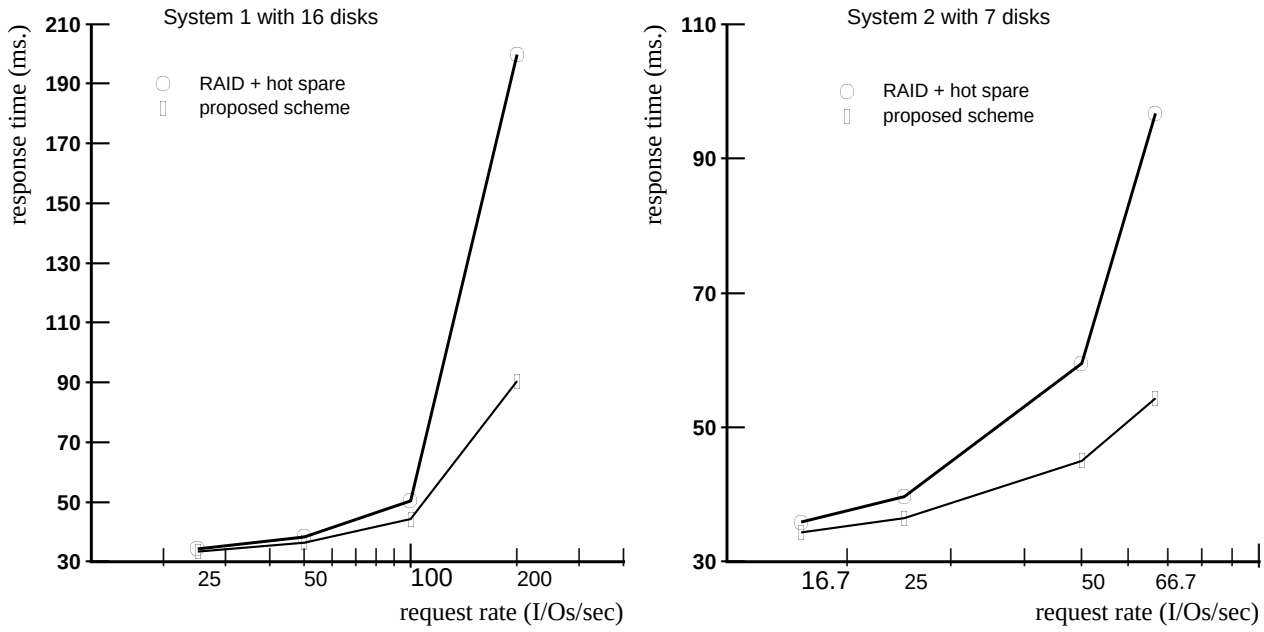

Fig. 14. Response time during a failure.

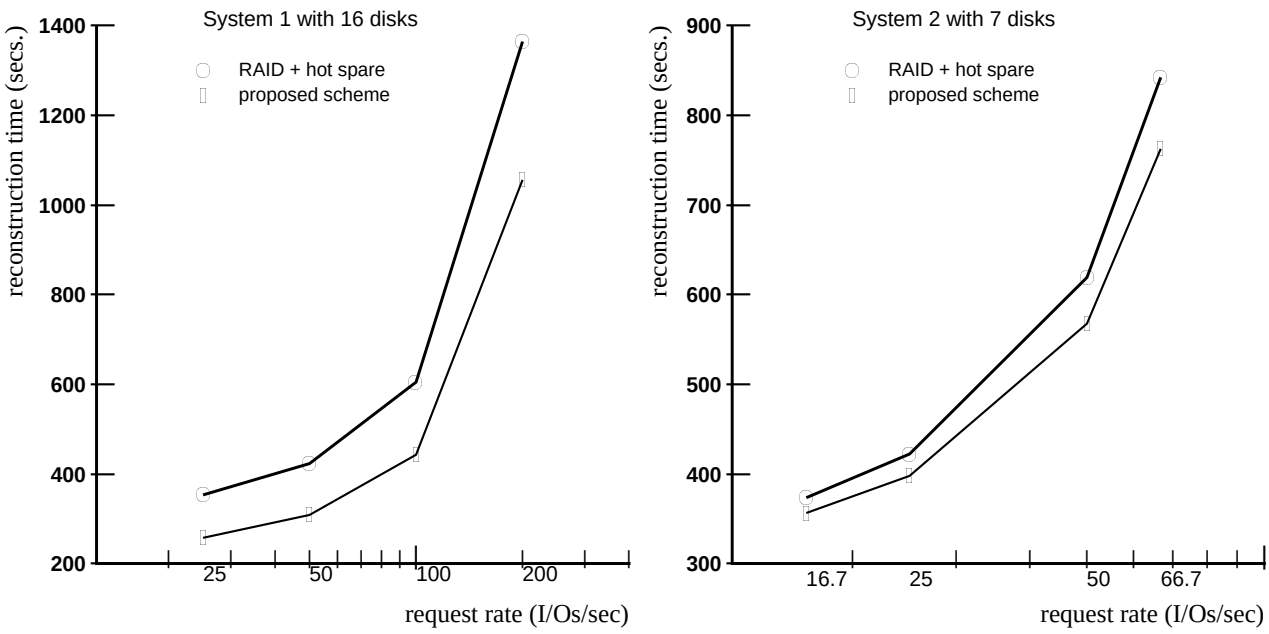

Fig. 15. Reconstruction time at different request rates.

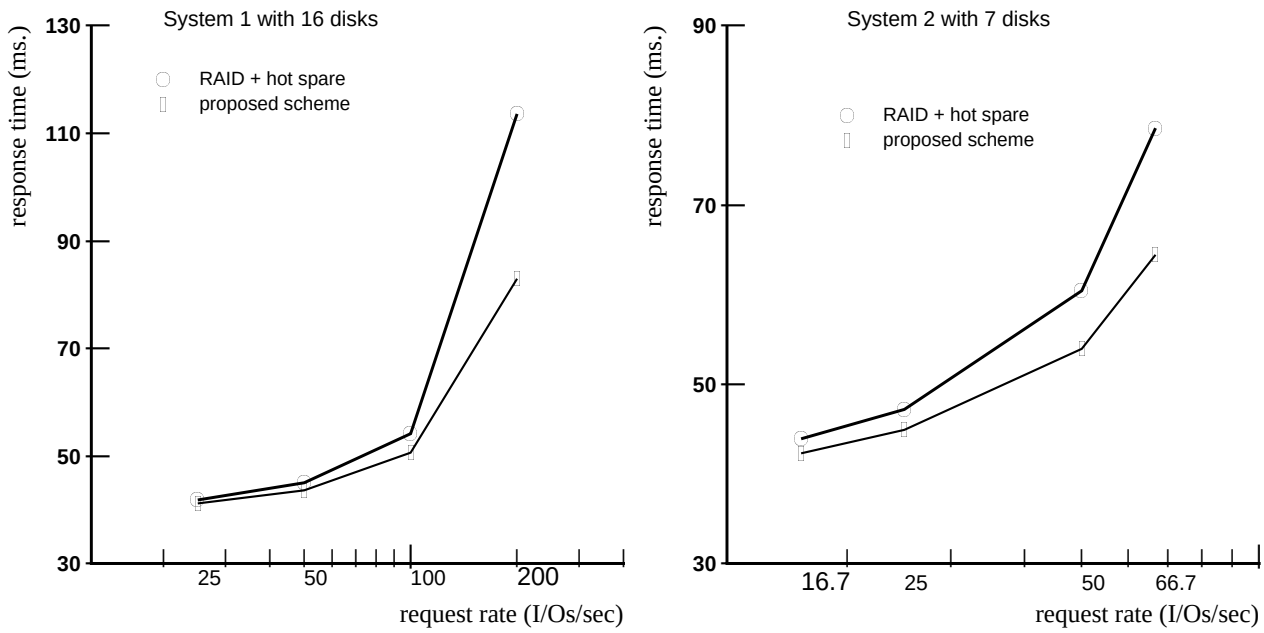

Fig. 16. Response times during reconstruction.

References

[1] Patterson, D. A., Gibson, G., and Katz, R. H. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Conference*, June 1988.

[2] Kim, M. Y. Synchronized disk interleaving. *IEEE Trans. Comput.*, C-35, no. 11:978–988, Nov. 1986.

[3] Livny, M., Khoshafian, S., and Boral, H. Multi-disk management algorithms. *Proc. ACM SIGMETRICS Conf.*, pages 69–77, May 1987.

[4] Reddy, A. L. N., and Banerjee, P. An evaluation of multiple-disk I/O systems. *IEEE Trans. Comput.*, C-38, no. 12:1680–1690, Dec. 1989.

[5] Salem, K., and Garcia-Molina, H. Disk striping. *Int. Conf. on Data Engineering*, pages 336–342, 1986.

[6] Katzman, J. A. A fault tolerant computing system. *Proc. 11th Hawaii Int. Conf. on System Sciences*, pages 85–102, Jan. 1978.

[7] Hsiao, H., and Dewitt, D. J. Chained declustering: A new availability strategy for multiprocessor database machines. *Proc. of Data Engineering*, 1990.

[8] DBC/1012 database computer system manual release 2.0, 1985. Teradata Corp., LA.

[9] Gibson, G. et al. Failure correction techniques for large disk arrays. *Proc. 3rd Int. Conf. on Architectural Support for Programming Languages and Operating Systems ASPLOS*, April 1989.

[10] Gray, J., Horst, B., and Walker, M. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. *Tech. Rep.: 90.2, Tandem Computers, Cupertino, CA 95104*, Jan. 1990.

[11] Castillo, X., and Siewiorek D. P. Workload, performance, and reliability of digital computing systems. *Proc. 11th Fault-Tolerant Comput. Symp.*, pages 84–89, 1981.

[12] Iyer, R. K., Butner, S. E., and McCluskey, E. J. A statistical failure/load relationship: Results of a multi-computer study. *IEEE Trans. Comput.*, C-31, no. 7:697–706, July 1982.

[13] Muntz, R. R, and Lui, J. Performance analysis of disk arrays under failure. *Proc. of 16th VLDB Conf.*, 1990.

[14] Reddy, A. L. N. Parallel input/output architectures for multiprocessors. *Ph. D Thesis, CRHC Tech. Rep.:90-5*, 1990. Coordinated Science Lab., University of Illinois, Urbana-Champaign.

[15] Hall, M., Jr. *Combinatorial Theory.* Blaisdell Publishing Company Waltham: MA, 1967.

[16] Schwetman, H. D. CSIM: A C-based, process-oriented simulation language. *Tech. Rep. PP-080-85, Microelectronics and Computer Technology Corp., Austin, Texas.*

[17] Chen, P. M., and Patterson, D. Maximizing performance in a striped disk array. *Proc. 17th Ann. Int. Symp. on Computer Architecture*, June 1990.

[18] Reddy, A. L. N. A study of I/O system organizations. *Proc. of Int. Symp. on Computer Architecture*, May 1992.

[19] Smith, A. J. Disk cache-miss ratio analysis and design considerations. *ACM Trans. on Comput. Systems*, 3, no. 3:161–203, Aug. 1985.