Efficient Placement of Parity and Data to Tolerate Two Disk Failures in Disk Array Systems

Chan-Ik Park

Abstract—**In this paper, we deal with the data/parity placement problem which is described as follows: how to place data and parity evenly across disks in order to tolerate two disk failures, given the number of disks N and the redundancy rate p which represents the amount of disk spaces to store parity information. To begin with, we transform the data/parity placement problem into the problem of constructing an $N \times N$ matrix such that the matrix will correspond to a solution to the problem. The method to construct a matrix has been proposed and we have shown how our method works through several illustrative examples. It is also shown that any matrix constructed by our proposed method can be mapped into a solution to the placement problem if a certain condition holds between N and p where N is the number of disks and p is a redundancy rate.**

Index Terms—**Data protection, disk array, disk failures, I/O performance, parity placement.**

I. Introduction

In the past few years, the performance of processors has been growing steadily, doubling approximately every two years [5], and with the advance of parallel processing technology, the processing power of a computer system is expected to be continuously increasing. However, the I/O performance has been far behind the processing power. With the widening gap between CPU and I/O, the performance of a computer system will eventually be limited by the performance of I/O [5], [16]. This bottleneck may be avoided by using more memory to buffer most of I/O activities. However, there are some applications in which I/O problem cannot be alleviated by adding more memory [7]. For such applications, it is very important to balance the I/O bandwidth and the processing power in order to improve the overall performance of a system.

There has been much research in improving I/O performance [6], [13], [16], [18], known as data declustering and disk striping in disk array systems. All of them utilize the parallelism among many disks in a system. However, incorporating such a large number of disks into a system makes the disk system more prone to failure than a single disk [13]. Assuming that the failure rate of a disk is constant and disk failures are statistically independent, the mean time to failure (MTTF) of the disk array system can be obtained simply by $MTTF_{single}/N$ where $MTTF_{single}$ represents the $MTTF$ of a single disk and N is the number of disks. High reliability can be achieved by using parity encoding of data to survive disk failures. For example, Patterson et al.,[13] has proposed redundant array of inexpensive disks (RAID). They defined six different RAID structures (RAID level 0 - 5) depending on the scheme of data and parity placement. Basically, RAID except the level 0 tolerates only one disk failure, i.e., data may be lost when two or more disks fail. However, since we believe that there is a growing demand in high reliability for user data, RAID has to be extended to recover data in the situation when two or more disks fail.

Any error correcting code such as Hamming and Reed-Solomon codes [15] can be used to tolerate disk failure(s) in disk array systems as long as a certain number of disks are exclusively reserved for storing redundant (e.g., parity) information [3]. For example, Hamming code is used in RAID level 2 to tolerate one disk failure. In the same way, any error correcting code with the capability of double error correction can be used to tolerate two disk failures in disk array systems. In [3] and [4], Gibson et al. show several codes like Extended Hamming code, Reed-Solomon code, and 2-d parity code which can correct double errors. In Fig. 1, we show how to place data and parity when 2-d parity code is applied to a disk array system consisting of 24 disks numbered as 0, 1, ⋯, and 23. The disk 16 stores parity information computed over disk 0, 1, 2, and 3, and the disk 20 stores parity information computed over disk 0, 4, 8, and 12, and so on. All disks over which a parity is computed and at which the parity is stored constitute an error correcting group. That is, the disk 0, 1, 2, 3, and 16 constitute one error correcting group and the disk 0, 4, 8, 12, and 20 constitute a different error correcting group as shown in Fig. 1. It is easily seen that the disk 16, 17, 18, 19, 20, 21, 22, and 23 are storing only parity information, which we call *parity disks*. The other disks storing only data information are called *data disks*. Note that each parity disk is included in only one error correcting group and each data disk is included in two error correcting groups, thus being able to tolerate two disk failures. However, parity disks used exclusively for storing parity information cause bottleneck, resulting in performance degradation of a disk array system [14]. To avoid this kind of bottleneck, parity and data should be distributed evenly across all disks. There are several schemes proposed in [8] to evenly distribute parity information in case of tolerating single disk failure. However, tolerating only one disk failure can not provide high reliability when the number of disks increases [3], [11]. Moreover, as previously mentioned, there is a growing demand in high reliability for user data.

This paper deals with the data/parity placement problem: how to place parity and data evenly across disks while tolerating two disk failures in disk array systems. There have been few research for this problem regardless of its importance. An error correcting

Manuscript received Dec. 3, 1993; revised Dec. 31, 1994.

C.-I. Park is with the Department of Computer Science and Engineering, Pohang University of Science and Technology, San 31 HyoJa Dong, Pohang KyungBuk 790-784, South Korea; e-mail: cipark@vision.postech.ac.kr.

To order reprints of this article, e-mail: transactions@computer.org, and reference IEEECS Log Number D95021.

code may be directly applied to solve the problem, but it requires the exclusive use of parity disks, causing the bottleneck. In [2], Frey solved the problem in a very limited situation consisting of only four disks. We propose a general method to solve the problem in any disk array systems.

The rest of the paper is organized as follows. Section II describes the problem in detail. Our method is presented in Section III. Section IV deals with our placement algorithm, and Section V gives some illustrative examples. Finally, Section VI makes conclusions and presents the future works that need to be done.

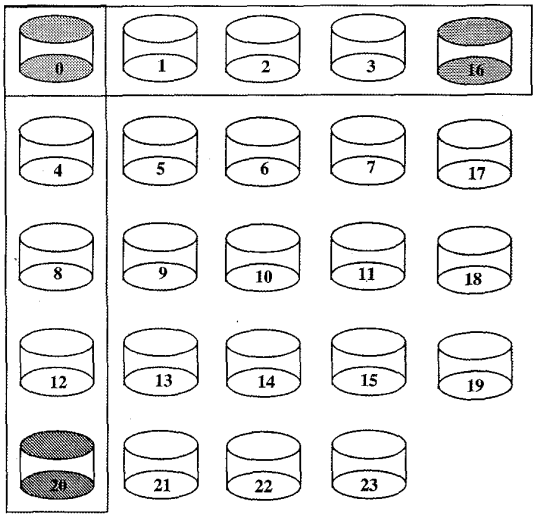

Fig. 1. Parity and data placement of 2-d parity: The disk 0, 1, ···, and 15 are data disks, and the disk 16, 17, ···, and 23 are parity disks.

II. Problem Formulation

Without loss of generality, consider N disks each of which contains M blocks shown in Fig. 2. For example, the ith disk has M blocks denoted as b_i^0, b_i^1, ···, and b_i^{M-1}. The disk block is a basic unit over which a parity is computed or at which the parity is stored. If a disk block contains a parity information, then it is called a parity block, otherwise, a data block. In order to guarantee even distribution of parity information across disks, let each disk contain one parity block for an error correcting group. Then, there are N error correcting groups and each of the $(M-1)N$ data blocks belongs to two such groups in order to tolerate two disk failures. More generally, note that if we make each of the $(M-1)N$ data blocks be related to k error correcting groups for $k \geq 2$, then k disk failures can be tolerated in that placement scheme. In this paper, we consider only $k = 2$. Assuming that all groups are of equal size, this implies $2(M-1)$ data blocks and one parity block per group, each stored on a distinct disk.

Since M represents the number of blocks in a disk, it can be used to represent how much disk spaces are occupied for storing parity information, called redundancy rate. For example, 50% redundancy rate for N disks means that half of total disk spaces is used to store parity information, i.e., total size of all parity blocks is equal to the half of total disk spaces. That means $M = 2$ since parity blocks are distributed evenly across all disks. In the same way, redundancy rate 25% means $M = 4$. Therefore, $M = 100/p$ where p denotes a redundancy rate.

Distributing parity blocks evenly across all disks reduces our original problem to the problem of finding how to relate each data block to two of N error correcting groups given a redundancy rate and the number of disks, which we call *double failure data placement* (DFDP).

Definition 1. *Double failure data placement* (DFDP): *Given a redundancy rate p and the number of disks N, construct N error correcting groups each of which consists of* $2(M-1)$ *data blocks and one parity block such that each data block should be included in two groups, where* $M = 100/p$.

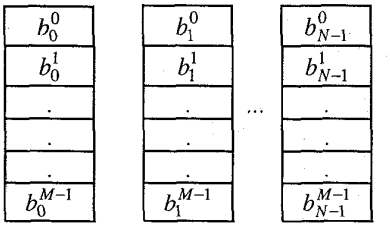

Fig. 2. Disk model: There are N disks, each of which has M blocks. b_i^j represents the jth block of the disk i.

For example, given $p = 50\%$ and $N = 4$, the solution to the DFDP problem is shown in Fig. 3, where P_i represents the parity block of the ith error correcting group and D_{ij} represents the data block which is included in both the ith and the jth groups, e.g., P_0 is computed over D_{30} and D_{01}, P_1 is computed over D_{12} and D_{01}, and so on. Now we show how this solution works in case of two disk failures. Suppose that the first two disks fail, i.e., the disk blocks, P_0, D_{12}, P_1, and D_{23} are lost. Then D_{23} can be reconstructed from D_{30} and P_3; next D_{12} can be reconstructed from D_{23} and P_2. In the same way, P_0 and P_1 can be recovered.

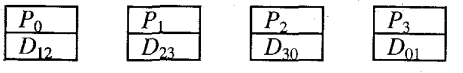

Fig. 3. A solution of DFDP problem for redundancy rate $p = 50\%$ and the number of disks $N = 4$: P_i represents the parity block of the ith error correcting group, and D_{ij} represents the data block which belongs to both the ith and jth groups.

In order to devise an algorithmic solution to the problem, we propose an $N \times N$ matrix to represent a solution to the problem called *redundancy matrix* (*RM*). Each column corresponds to a disk and each row corresponds to an error correcting group. Let RM_{ij} be an element of an *RM* which has a value ranging from -1 to $M-1$. According to the value of RM_{ij}, the interpretation differs as follows.

- $RM_{ij} = -1$: A parity block of the disk j belongs to the error correcting group i.
- $RM_{ij} = 0$: nothing.
- $RM_{ij} = k$ for $1 \leq k \leq M-1$: The kth data block of the disk j, b_j^k, belongs to the error correcting group i.

An *RM* has two properties as follows. First, each column has one entry with the value of -1 and two entries with the value of k

for $1 \le k \le M-1$ and each row has one entry with the value of -1 and $2(M-1)$ positive entries, since each row and column represents an error correcting group and a disk, respectively. Second, every pair of columns in such an *RM* are c-independent. The c-independence is to be defined later. For example, the solution shown in Fig. 3 can be represented by an 4×4 matrix shown in Fig. 4. In this example, there are only two blocks in each disk, one for parity block and the other for data block, i.e., $M = 2$. In Fig. 4, the second column tells us about the disk 1 that the parity block of the disk 1 belongs to group 1, and the data block belongs to groups 2 and 3. From the second row, we can tell that the group 1 consists of the data block of the disk 0 and 3, and the parity block of the disk 1. Note that the matrix of Fig. 4 becomes an *RM* since it satisfies two properties of an *RM*.

	0	1	2	3
0	−1	0	1	1
1	1	−1	0	1
2	1	1	−1	0
3	0	1	1	−1

Fig. 4. A 4×4 matrix representation of a solution for $p = 50\%$ and $N = 4$: Each column corresponds to a disk, and each row corresponds to one error correcting group.

We introduce several definitions for handling these two properties of an *RM*.

DEFINITION 2. *Placement array and vector: A placement array with parameters N and M is defined to be an array of length N with $2M-1$ nonzero entries, with one occurrence of -1 and two occurrences of k for $k = 1, \cdots$, and $M-1$. A placement vector is defined to be the ordered set on nonzero entries of a placement array. In addition, the entry with -1 is called parity element and the other nonzero entries are data elements.*

As an example, Fig. 5 shows several placement vectors for $M = 3$ and $N = 5$ when we assume that -1 entry is positioned at the top.

−1	−1	−1	−1	−1	−1
1	1	1	2	2	2
1	2	2	2	1	1
2	1	2	1	2	1
2	2	1	1	1	2

Fig. 5. Examples of placement vectors for $M = 3$ and $N = 5$, assuming that -1 entry is positioned at the top.

DEFINITION 3. *Internal edge (i-edge): For a placement vector $C_i = [c_i^0, c_i^1, \cdots, c_i^{N-1}]$, an i-edge (c_i^k, c_i^j) is defined between c_i^k and c_i^j if $c_i^k = c_i^j > 0$. Note that there exist $M-1$ i-edges in C_i.*

DEFINITION 4. *External edge (e-edge): Given two placement vectors C_i and $C_j, C_i = [c_i^0, c_i^1, \cdots, c_i^{N-1}]$ and $C_j = [c_j^0, c_j^1, \cdots, c_j^{N-1}]$, an external edge (c_i^k, c_j^k) is defined between c_i^k and c_j^k if $c_i^k \neq 0$ or $c_j^k \neq 0$. An e-edge is called a-edge if $(c_i^k \neq 0$ and $c_j^k \neq 0)$, s-edge otherwise. Specifically, a-edge and s-edge are called ap-edge and sp-edge, respectively, if $(c_i^k = -1$ or $c_j^k = -1)$.*

DEFINITION 5. *Reachable and isolated path: Given two placement vectors C_i and C_j, a path consisting of any type of edges is called reachable if it includes at least one s-edge, isolated otherwise. Similarly, a nonzero entry is called reachable if it is visited by a reachable path, isolated otherwise.*

Note that isolated paths cannot include s-edges. There are two types of isolated paths. One is a closed path among isolated positive entries, consisting of i-edges and a-edges. The other is an open path starting and ending with ap-edge, consisting of i-edges and a-edges in between.

DEFINITION 6. *c-independence: Two placement vectors are c-independent if all their nonzero entries are reachable, i.e., there are no isolated paths. A placement vector can be used as a column to construct an $N \times N$ matrix. Then, an $N \times N$ matrix is c-independent if any pair of columns are c-independent.*

For example, consider two placement vectors shown in Fig. 6a, $C_0 = [-1, 3, 2, 1, 1, 2, 3, 0]$ and $C_1 = [1, 2, 3, 0, -1, 3, 2, 1]$. The i-edges in C_0 include (c_0^1, c_0^6), (c_0^2, c_0^5), and (c_0^3, c_0^4) while those in C_1 include (c_1^0, c_1^7), (c_1^1, c_1^6), and (c_1^2, c_1^5). The a-edges for C_0 and C_1 include (c_0^0, c_1^0), (c_0^1, c_1^1), (c_0^2, c_1^2), (c_0^4, c_1^4), (c_0^5, c_1^5), and (c_0^6, c_1^6). Among these a-edges, (c_0^0, c_1^0) and (c_0^4, c_1^4) are called ap-edges since $c_0^0 = -1$ and $c_1^4 = -1$, respectively. The s-edges include (c_0^3, c_1^3) and (c_0^7, c_1^7). Note that there is no sp-edge. In addition, it is easily seen that there exist two isolated closed path consisting of two i-edges and two a-edges: One consists of (c_0^1, c_1^1), (c_0^1, c_0^6), (c_0^6, c_1^6), and (c_1^6, c_1^1) while the other consists of (c_0^2, c_0^5), (c_0^5, c_1^5), (c_1^5, c_1^2), and (c_0^2, c_1^2). Therefore, C_0 and C_1 are not c–independent. We could see that there is an isolated open path in Fig. 6b consisting of (c_0^0, c_1^0), (c_1^0, c_1^3), (c_1^3, c_0^3), (c_0^3, c_0^4), and (c_0^4, c_1^4).

THEOREM 1. *Given any $N \times N$ matrix H, the following properties are equivalent if it is an RM.*

1) *H allows any two disk failures to be corrected.*
2) *Any pair of two columns selected from H is c-independent.*

PROOF. Assume that disks i and j have failed. If c_i^k and c_j^k is an *s-edge* then exactly one of the disks i and j contains a block from group k, and that block can be reconstructed. If (c_i^k, c_j^k) is an *a-edge*, and then both disks contain data blocks in group k; if one on these two blocks is reconstructed then the other can be, too. If (c_i^k, c_j^k) is an *i-edge* then both entries represent the same data block. It follows that each block that corresponds to a reachable entry can be computed. Thus, if c_i and c_j are c-independent, the failure of disk i and j can be corrected.

Conversely, assume that c_i^k is isolated. Let ϕ_C be the set of nonzero entries connected to c_i^k, and assume no s-edge is

−1	1
3	2
2	3
1	0
1	−1
2	3
3	2
0	1

(a)

−1	2
3	3
2	1
1	2
1	−1
2	3
3	0
0	1

(b)

Fig. 6. Example of types of isolated paths: (a) Two closed paths exist. One consists of $\left(c_0^1, c_1^1\right)$, $\left(c_0^1, c_0^6\right)$, $\left(c_0^6, c_1^6\right)$, and $\left(c_1^6, c_1^1\right)$ while the other consists of $\left(c_0^2, c_0^5\right)$, $\left(c_0^5, c_1^5\right)$, $\left(c_1^5, c_1^2\right)$, and $\left(c_0^2, c_1^2\right)$. (b) An open path exist which consists of $\left(c_0^0, c_1^0\right)$, $\left(c_1^0, c_1^3\right)$, $\left(c_1^3, c_0^3\right)$, $\left(c_0^3, c_0^4\right)$, and $\left(c_0^4, c_1^4\right)$.

incident to an entry in ϕ_C. Then all groups corresponding to entries in ϕ_C have lost two blocks. The value of the block represented by c_i^k can be set arbitrarily, without affecting the values of remaining blocks. This proves the theorem. □

III. Parity and Data Placement

With the help of Theorem 1, the DFDP problem for a given redundancy rate p and the number of disks N can be transformed into the problem of constructing an $N \times N$ matrix such that it satisfies two properties of an RM: each column conforms to a placement vector and any pair of columns is *c-independent*. To begin with, construct a placement array consisting of $2M - 1$ nonzero elements whose values are -1, 1, ..., and $M - 1$, where $M = 100/p$. Then, a placement vector is obtained from an ordering on the positive elements in a placement array assuming that -1 is positioned at the first entry. For example shown in Fig. 5, there are $4!/(2! \cdot 2!)$ possible ways to determine the ordering on the positive entries of a placement vector for $M = 3$ and $N = 5$. Note that -1 is positioned at the first entry and an ordering on the positive elements forms a placement vector. Any placement vector can be used as a seed column to construct an $N \times N$ matrix.

In order to devise a systematic method to design a placement vector which will be used as a seed column, the term *group interval* is introduced which is conceptually similar to the length of i-edges.

DEFINITION 7. *Group interval (GI): In a placement vector C_i for N and M, $C_i = \left[c_i^0, c_i^1, \ldots, c_i^{N-1}\right]$, each entry has the value among -1, 0, 1, ..., and $M - 1$. Remember that a positive value occurs twice in a placement vector. Then, the group interval of each entry of C_i is defined as*

$$GI(k) = \begin{cases} -1 & \text{if } k = -1, \\ 0, & \text{if } k = 0 \\ |h-g|-1, & \text{if } 1 \le k \le M-1 \text{ and } c_i^h = c_i^g = k. \end{cases}$$

There are $(N - 1)!/(2^{M-1}\ (N - 2M + 1)!)$ orderings on the positive entries in a placement vector for N and M since positive entries occur twice and the number of 0 entries is equal to $N - 2M + 1$. Among these orderings, we consider an ordering which satisfies the following condition called interval condition : if $i \neq j$ for $1 \le i, j \le M - 1$, then $GI(i) \neq GI(j)$. For example in Fig. 5, we can easily see that there are two valid orderings, the third and the sixth, which satisfy the interval condition, i.e., $GI(1) \neq GI(2)$. These two orderings are logically the same. From now on, a placement vector is assumed to be obtained from an ordering which satisfies the interval condition. There may be several orderings which satisfy the interval condition. In this paper, we consider a specific ordering such that -1 entry is positioned at the first element of the vector and each positive entries of i are positioned such that $GI(i) = 2(i - 1)$ for $1 \le i \le M - 1$. In other words, the group intervals of positive entries are determined as $GI(M - 1) = 2(M - 2)$, $GI(M - 2) = 2(M - 3)$, ..., and $GI(1) = 0$. Fig. 8 shows a placement vector obtained from the specific ordering just described. We will use this placement vector as a seed column to construct a matrix. It makes the matrix construction quite simple and efficient.

By using the seed column as C_0, we construct an $N \times N$ matrix for N and M, $R_{(M,N)} = [C_0, C_1, \cdots, C_{N-1}]$, as shown in Fig. 7. Note that the matrix is constructed simply by shifting down the seed column C_0. At C_{N-2M+2} during shifting down, we could see that $M - 1$ entry has to be positioned at the top, so that the group interval of $M - 1$ entry is changed. Similarly, it is easily seen that the group intervals of some positive entries are changed in the following columns C_{N-2M+3}, C_{N-2M+4}, ..., and C_{N-1}. The columns in which the group intervals have to be changed are called wrapped-around columns. We will denote the changed group interval as $GI^*(\ )$ instead of $GI(\ )$ in a wrapped-around columns as shown in Fig. 9. Fig. 10 shows a schematic view of group intervals in a wrapped-around column, showing that there are two centers among group intervals, whereas there is one center among the group intervals in non wrapped-around columns as shown in Fig. 8.

If $R_{(M,N)}$ is c-independent, then it becomes an RM matrix which corresponds to a solution to the DFDP problem. In order to prove the c-independence property of $R_{(M,N)}$, we need to verify the nonexistence of isolated paths in any pair of columns. There are $N(N - 1)/2$ pairs of columns in $R_{(M,N)}$ to be checked. Considering the structure of columns in $R_{(M,N)}$, however, it is enough to check only $(N - 1)$ pairs which include (C_0, C_1), (C_0, C_2), ..., and (C_0, C_{N-1}). For these $N - 1$ pairs of columns, we have to check whether there exist any isolated paths. If there exist no isolated paths, then $R_{(M,N)}$ can be transformed into a solution to the DFDP problem for the given M and N.

From the above procedure, we can devise a simple placement algorithm which generates a solution of how to place parity and data in order to provide fault tolerance against *two disk* failures, given a redundancy rate p and the number of disks N. Each step of the algorithm is described as follows.

1) Calculate the number of disk blocks M by $M = 100 / p$.
2) Construct a placement vector for M and N:

−1	$M-1$	$M-2$	⋯	2	1	1	2	⋯	$M-2$	$M-1$	⋯	0

3) Construct an $N \times N$ matrix $R_{(M,N)}$ column-by-column

−1	0		M−1
M−1	−1		M−2
M−2	M−1		M−3
.	.	....	.
.	.	....	.
.	.	....	.
1	2		1
1	1		2
.	.	....	.
.	.	....	.
.	.	....	.
M−1	M−2		0
0	M−1		0
.	.	....	.
.	.	....	.
.	.	....	.
0	0		−1

Fig. 7. An $N \times N$ matrix for N and M, $R_{(M,N)}$.

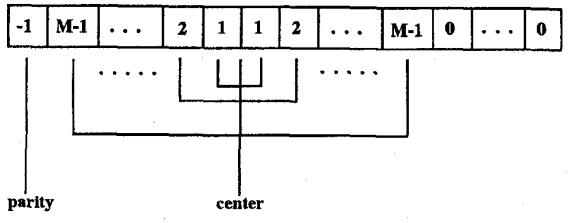

Fig. 8. Schematic view of group interval: The lines show i-edges in the placement vector. There is a center in the group intervals denoted as *center*.

by using the placement box just constructed as a seed column.

4) We have to check $N - 1$ pairs of columns of $R_{(M,N)}$ $(C_0, C_1), \cdots, (C_0, C_{N-1})$ for c-independence.
5) If $R_{(M,N)}$ is c-independent, i.e., turns out to be an RM, place parity and data according to the $R_{(M,N)}$ matrix.

It is easily seen that the operation of checking c-independence has the time complexity of $O(M)$. Since there are $(N - 1)$ pairs of columns to check, then the placement algorithm has $O(N\,M)$ time complexity. In the next section, we will show that the operation of checking c-independence is not needed when a certain condition between N and M holds.

IV. A Condition Guaranteeing C-Independence

The operation of checking c-independence for a pair of columns can be easily implemented with the time complexity of $O(M)$. However, this section derives a condition between N and M which guarantees $R_{(M,N)}$ to be c-independent without the explicit checking operations. To begin with, remember that there exist two types of isolated paths: a closed path and an open path.

LEMMA 1. *Isolated closed paths cannot exist for any pair of columns of $R_{(M,N)}$ if $N \geq 4M - 5$.*

PROOF. There may be a lot of complicated patterns of closed paths in a column pair. For example, Fig. 11 shows schematic views of two possible patterns of closed paths. If we impose a certain restriction on the number of disks N, then we can make most patterns impossible to occur.

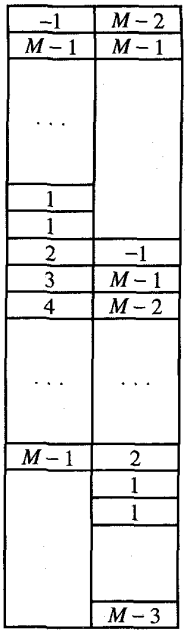

Fig. 9. Change of group interval in a wrapped-around column C_{N-2M+3}: The example shows that the group intervals of $(M - 1)$ and $(M - 2)$ entries are changed to $GI^*(M - 1)$ and $GI^*(M - 2)$, respectively.

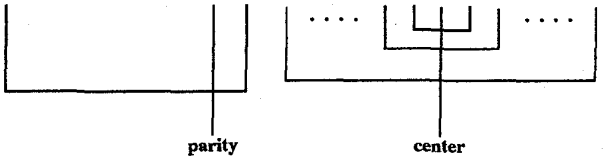

Fig. 10. Schematic view of group interval for a wrapped-around column: The lines show i-edges and note that there is a changed group interval shown around parity.

Without loss of generality, let $N = 3M - 2 + k$ for $k \geq 0$. It is trivial to show that there are no isolated closed paths in any pair of columns (C_0, C_1), (C_0, C_2), $\cdots$, and (C_0, C_{M+k-1}) since they are not wrapped-around columns and the number of 0 entries are $M + k - 1$. Thus, we will consider only the following column pairs (C_0, C_{M+k}), $\cdots$, and (C_0, C_{N-1}). Since the number of 0 entries are $(M - 1 + k)$, there are at least M entries between the center of C_0 and that of C_{M+k}. In case of $k = 0$, it is easily seen from Fig. 12 that the distance between the center of C_0 and that of C_M is M. Thus, we can see that the center of C_M is located out of the reach of $M - 1$ entry of C_0 because the maximum distance of $M - 1$ entry from the center in C_0 is equal to $M - 1$. This is also true for the other columns C_{M+1}, $\cdots$, and C_{N-1} which come after C_M. Therefore, simply by imposing the value restriction on N $(N \geq 3M - 2)$, we can exclude the occurrence of patterns shown in Fig. 11a as well as other more complicated patterns consisting of more than four i-edges and a-edges. Now, we will check for the simplest pattern of closed paths shown in Fig.11b.

To begin with, note that $GI(M - i) = 2(M - i - 1)$ and $GI^*(M - j) = M + 2j - 2 + k$ for $1 \leq i, j \leq M - 1$ when $N = 3M - 2 + k$. The

pattern of Fig. 11b occurs if there exist i and j such that $GI(i) = GI^*(j)$. Considering the minimum and maximum values of $GI(M-i)$ and $GI^*(M-j)$, we get that $0 \le GI() \le 2(M-2)$ and $M+k \le GI^*() \le 3M-4+k$. If we set the value of k such that the minimum value of $GI^*(\,)$ is greater than the maximum value of $GI(\,)$, then it is impossible for $GI(i) = GI^*(j)$ to be true. Thus, $2(M-2) < M+k$, i.e., $k > M-4$. That is, if $N = 3M-2+k > 4M-6$, then there exist no closed paths. □

LEMMA 2. *Isolated closed paths cannot exist for any pair of columns of $R_{(M,N)}$ if $N \ge 3M-2$ and N is an odd number.*

PROOF. From the proof procedure of Lemma 1, we know that a closed path like Fig. 11b occurs if there exist i and j values such that $GI(M-i) = GI^*(M-j)$ for $1 \le i, j \le M-1$. Note that $GI(M-i) = 2(M-i-1)$ and $GI^*(M-j) = M+2j-2+k$ when $N = 3M-2+k$. Since $GI(M-i) = 2(M-i-1)$, it is always an even number. If we make $GI^*(M-j)$ be an odd number, then $GI(M-i)$ cannot be equal to $GI^*(M-j)$. In order to make $GI^*(M-j)$ be an odd number, $M+k$ must be an odd number because $GI^*(M-j) = M+k+2j-2$ and $2j-2$ is always an even number. Remember that $N = 3M-2+k = 2(M-1)+M+k$. Thus, if N is an odd number, then $M+k$ is always an odd number. Then, there exist no isolated closed paths. □

LEMMA 3. *Isolated open paths cannot exist for any pair of columns of $R_{(M,N)}$ if $N \ge 3M-2$.*

PROOF. An isolated open path is a path starting and ending with *ap*-edges, possibly connected by a few number of *i*-edges and *a*-edges inbetween. If there exists an *sp*-edge in a column pair, then it is obvious that an open path cannot exist. In addition, if there are two *ap*-edges and either one of the two is included in a reachable path, then an open path cannot exist. Note that *ap*-edge comes from parity element.

Let $N = 3M-2+k$ for $k \ge 0$. For the pairs of (C_0, C_1), (C_0, C_2), $\cdots$, and (C_0, C_{M+k-1}), i.e., non wrapped-around column pairs, it is trivial to show that there is at most one *ap*-edge because *sp*-edge exists for the parity element in C_0. Similary, for the pairs of (C_0, C_{2M-1+k}), (C_0, C_{2M+k}), $\cdots$, and (C_0, C_{N-1}), *sp*-edge exists for the parity element in $C_{2M-1+k+i}$ for $1 \le i \le M-1$. Therefore, there cannot exist open paths in these column pairs.

For the pairs of (C_0, C_{M+k}), (C_0, C_{M+k+1}), $\cdots$, and (C_0, C_{2M-2+k}), we will show the nonexistence of an open path only for the case of $k = 0$ because $k = 0$ is the worst case to check for an open path. Without loss of generality, consider a pair of columns (C_0, C_{M+j-1}) shown in Fig. 13 for $1 \le j \le N-M$.

From Fig. 13 if $M-j-1 \ge j$, then it is easy to see that the *ap*-edge $(j, -1)$ for the parity element of C_{M+j-1} is connected to an *s*-edge $(j, 0)$ in the region II by one *i*-edge (j, j) of C_0. Similarly, if $M-j-1 < j$, it is easy to see that the *ap*-edge $(-1, M-j)$ for the parity element of C_0 is connected to an *s*-edge $(M-j, 0)$ in the region V by one *i*-edge $(M-j, M-j)$ of C_{M+j-1}. Thus, if $N = 3M-2+k$ for $k \ge 0$, then there exist no isolated open paths. □

LEMMA 4. *Any pair of columns of $R_{(M,N)}$ is c-independent if either N is an odd number such that $N \ge 3M-2$ or $N \ge 4M-5$.*

PROOF. Combining the results of Lemma 2 and Lemma 3 leads us to prove the *either* part of this lemma. It is also easy to prove the *or* part of this lemma by using the results of Lemma 1 and Lemma 3. □

THEOREM 2. *For a given M and N where M is the number of blocks per disk and N is the number of disks, if we*

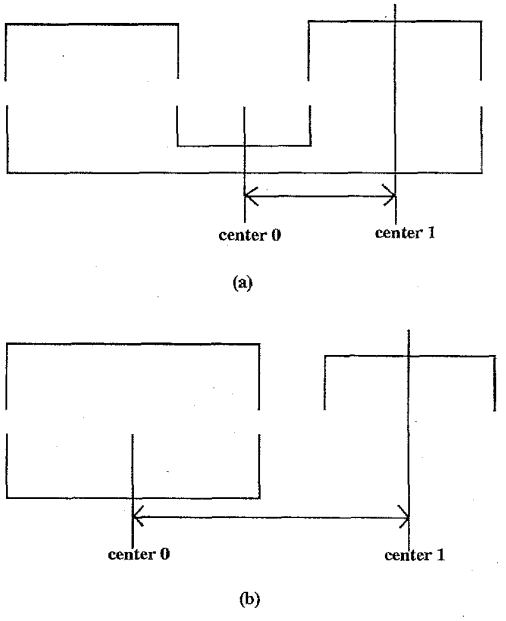

Fig. 11. Possible patterns of isolated closed paths: The upper column is a wrapped-around column, while the lower column is not. (a) Shows a pattern possible to occur when the distance between two centers of both columns is less than $M-1$ and the upper column is a wrapped-around column, and (b) shows a pattern possible to occur when two group intervals happen to be equal by wrapped-around operation.

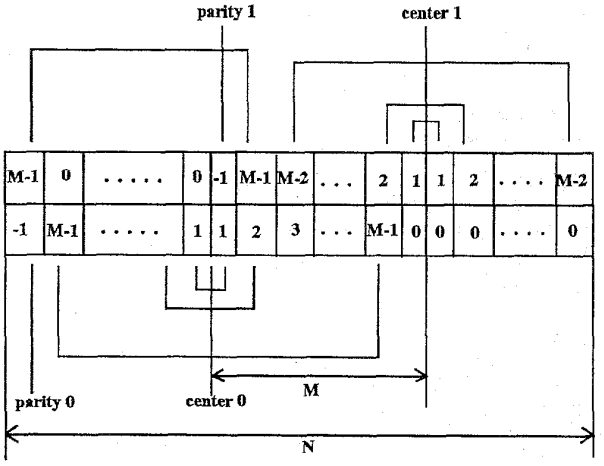

Fig. 12. Separation of two centers by M for a column pair C_0 and C_M when $N = 3M-2$: Center0 is the center of C_0 while Center1 is the center of C_M. Note that C_M is a wrapped-around column.

region I	-1	$M-j$	j
	$M-1$	$M-j+1$	
	⋯	⋯	
	$M-j+1$	$M-1$	
region II	$M-j$	0	$M-j$
	⋯	⋯	
	1	0	
region III	1	0	j
	⋯	⋯	
	$j-1$	0	
	j	-1	
region IV	$j+1$	$M-1$	$M-j-1$
	⋯	⋯	
	$M-1$	$j+1$	
region V	0	j	j
	⋯	⋯	
	0	1	
region VI	0	1	$M-j-1$
	⋯	⋯	
	0	$M-j-1$	

Fig. 13. C_0 and C_{M+j-1} in $R_{(M,N)}$ when $N = 3M - 2$.

choose N as an odd number such that $N \geq 3M - 2$, then a matrix $R_{(M,N)}$ becomes a redundancy matrix RM which can be transformed into a solution to the DFDP problem.

PROOF. It is obvious from Lemma 4. . □

Considering the condition given in Theorem 2, the placement algorithm explained in the previous section is now rewritten as follows for a given redundancy rate p and the number of disks N. Of course, the time complexity of the resulting algorithm remains the same.

1) Calculate the number of disk blocks M by $M = 100 / p$.
2) Construct a placement vector for M and N:

-1	$M-1$	$M-2$	⋯	2	1	1	2	⋯	$M-2$	$M-1$	⋯	0

3) Construct an $N \times N$ matrix $R_{(M,N)}$ column-by-column by using the placement box just constructed as a seed column.
4) By Theorem 2, choose the least odd number $N_{required}$ such that $N_{required} \geq 3M - 2$. This number becomes the required number of disks.
5) If $N \geq N_{required}$ where N is the given number of disks, then we guarantee that $R_{(M,N)}$ is c-independent. Otherwise, we have to check $N - 1$ pairs of columns of $R_{(M,N)}$ (C_0, C_1), ⋯, (C_0, C_{N-1}) for c-independence.
6) If $R_{(M,N)}$ is c-independent, i.e., turns out to be an RM, place parity and data according to the $R_{(M,N)}$ matrix.

It is obvious that N should be greater than $2M$ in order to tolerate two disk failures. Although we provide Theorem 2 describing a condition between the number of disks N and the number of blocks M in a disk which guarantees c-independence, note that it is possible to tolerate two disk failures by using the lower value of N. By simple enumeration of our placement algorithm, Table I is obtained which shows the minimum value of N such that an RM matrix can be constructed. For comparison, the minimum value of N obtained by Theorem 2 is also shown.

TABLE I
THE MINIMUM VALUE OF N FOR A GIVEN M SUCH THAT $R_{(M,N)}$ CAN BE AN RM MATRIX

M	minimum N obtained by the algorithm	minimum N obtained by Theorem 2
3	7	7
4	10	11
5	11	13
6	13	17
7	16	19
8	17	23
9	22	25
10	22	29

However, it is very hard to derive a general formula for the minimum number of N. Note that the minimum value of N shown above cannot be the lower bound because there is a case which cannot construct an RM matrix to tolerate two disk failures. For example, in case of $M = 8$, the minimum value of N is 17, but we can't construct an RM when $N = 18$. Thus the value of N determined by Theorem 2 has generality although it requires more number of disks than the minimum.

In addition, our method can be easily extended to handle more than two disk failures. In order to handle three disk failures, the modification is needed only in the definition of a placement array and vector. Current definition requires positive entries to occur twice in a placement vector. If we extend the number of occurrence of positive entries from two to the general number r, then the new definition requires positive entries to occur r times in a placement array and vector where $r \geq 2$. With this extension, we can devise a systematic method to check whether a certain placement of data and parity can handle r disk failures where $r \geq 2$. There will, however, exist much more complicated patterns for isolated paths, thus making very difficult to analyze the patterns. The following section describes how to apply our method to the DFDP problem to tolerate two disk failures.

V. ILLUSTRATIVE EXAMPLES

The following two examples show how our algorithm works. The first example is for $N = 8$ and $p = 25\%$ and the second one is for $N = 7$ and $p = 33.33\%$.

- **Example 1.** $N = 8$ and $p = 25\%$

1) $M = 100/p = 4$.
2) Construct a placement vector for $M = 4$ and $N = 8$.

-1	3	2	1	1	2	3	0

3) Construct an 8×8 matrix $R_{(4,8)}$ by using a placement vector as a seed column.

	0	1	2	3	4	5	6	7
0	−1	0	3	2	1	1	2	3
1	3	−1	0	3	2	1	1	2
2	2	3	−1	0	3	2	1	1
3	1	2	3	−1	0	3	2	1
4	1	1	2	3	−1	0	3	2
5	2	1	1	2	3	−1	0	3
6	3	2	1	1	2	3	−1	0
7	0	3	2	1	1	2	3	−1

4) By Theorem 2, choose $N_{required}$. Then, $N_{required} = 11$.
5) Since $N < N_{required}$, we cannot guarantee that a constructed 8×8 matrix $R_{(4,8)}$ is c-independent. Thus, we have to check $N-1$ pairs of columns $(C_0, C_1), \cdots, (C_0, C_{N-1})$ for c-independence.
6) Since there is an isolated closed path for C_0 and C_4, the $R_{(4,8)}$ matrix is not c-independent.

- **Example 2.** $N = 7$ and $p = 33.33\%$

1) $M = 100/p = 3$.
2) Construct a placement vector for $M = 3$ and $N = 7$:

−1	2	1	1	2	0	0

3) Construct 7×7 $R_{(3,7)}$ from the placement vector just constructed:

	0	1	2	3	4	5	6
0	−1	0	0	2	1	1	2
1	2	−1	0	0	2	1	1
2	1	2	−1	0	0	2	1
3	1	1	2	−1	0	0	2
4	2	1	1	2	−1	0	0
5	0	2	1	1	2	−1	0
6	0	0	2	1	1	2	−1

4) Choose $N_{required}$ by Theorem 2. Then, $N_{required} = 7$.
5) Since $N = N_{required}$, then we guarantee that the constructed 7×7 matrix $R_{(3,7)}$ is c-independent. Thus, $R_{(3,7)}$ becomes an RM.
6) Place parity and data according to the $RM_{(3,7)}$:

P_0	P_1	P_2	P_3	P_4	P_5	P_6
D_{23}	D_{34}	D_{45}	D_{56}	D_{60}	D_{01}	D_{12}
D_{14}	D_{25}	D_{36}	D_{40}	D_{51}	D_{62}	D_{03}

VI. Conclusion and Future Works

There have been growing demand in high reliability beyond what current RAID technology can provide. This paper has proposed an efficient algorithm of placing parity and data evenly across all disks in order to tolerate two disk failures in disk array systems. By representing a solution as a matrix, we transform the problem of placing parity and data into the problem of constructing a matrix called a redundancy matrix. We have proposed a method how to construct such a matrix which corresponds to a solution to the original placement problem. It is also shown that if a certain condition between the number of disks and a redundancy rate holds for a matrix constructed by our method, then the matrix can be directly transformed into a solution to the problem. Even though our method is proposed for handling two disk failures, it can be easily extended to handle more than two disk failures. Our current scheme, however, has practicality drawbacks since we assume that a disk can be partitioned into any number of blocks M.

Future works include the reduction of upper bound of N for a given p. One way of doing this is to investigate several different ways to construct placement vector which is used as a seed column to construct a solution matrix. In addition, the investigation on performance of the resulting placement as well as practicality drawbacks will be very interesting problems.

Acknowledgment

This work was supported in part by the Korea Research Foundation, the IBM Thomas J. Watson Research Center, and the Agency for Defense Development, Korea.

References

[1] P.C. Dibble, "A Parallel interleaved file system," *Computer Science*, Tech. Report 334, Univ. of Rochester, Mar. 1990.
[2] S. Frey, "DATACUBE distributed system," IBM Los Angeles Scientific Center, Mar. 1991.
[3] G. Gibson, et al., " Coding techniques for handling failures in large disk arrays," Computer Science Tech. Report CSD88-477, Univ. of California, Berkeley, Dec. 1988.
[4] G. Gibson, "Redundant disk arrays: Reliable, parallel secondary storage," PhD dissertation, Univ. of California at Berkeley, Dec. 1990.
[5] R.H. Katz, G.A. Gibson, and D.A. Patterson, "Disk system architectures for high performance computing," *Proc. IEEE*, vol. 77, no. 12, pp. 1,842-1,858, Dec. 1,989.
[6] M.Y. Kim, "Synchronized disk interleaving," *IEEE Trans. Computers*, vol. 35, no. 11, pp. 978-988, Nov. 1986.
[7] H.T. Kung, "Memory requirements for balanced computer architectures," *Proc. 13th Ann Int'l Symp. Computer Architecture*, pp. 49-54, 1986.
[8] E.K. Lee and R.H. Katz, "Performance consequences of parity placement in disk arrays," *Proc. Fourth ACM Arch. Supp. Prog. Lang. Oper. Syst.*, Santa Clara, Calif., pp. 190-199, 1990.
[9] E. K. Lee and Randy H. Katz, "The performance of parity placements in disk arrays," *IEEE Trans. Computers*, vol. 42, no. 6, pp. 651-664 June 1993.
[10] S.W. Ng, "Sparing for a redundant disk array," *IBM Research Report RJ* 7621, 1990.
[11] C.-I. Park, "Comparison of MTTF of various coding techniques in disk arrays," internal memo, System Software Lab., Dept. of Computer Science, POSTECH, 1992.
[12] A.M. Patel, "Error and failure-control procedure for a large-size bubble memory," *IEEE Trans. Magnetics*, vol. 18, no. 6, pp. 1,319-1,321, 1982.
[13] D.A. Patterson, G. Gibson, and R.H. Katz, "A Case for redundant arrays of inexpensive disks (RAID)," *Proc. ACM SIGMOD Conf.*, Chicago, pp. 109-116, 1988.
[14] D.A. Patterson, et al., "Introduction to redundant arrays of inexpensive disks (RAID)," Computer Science Report CSD88-479, Univ. of Calif. Berkeley, 1988.
[15] W.W. Peterson and E.J. Weldon Jr., *Error-Correcting Codes*, MIT Press, 1972.
[16] A.L.N. Reddy and P. Banerjee, "An evaluation of multiple-disk I/O systems," *IEEE Trans. Computers*, vol. 38, no. 12, pp. 1,680-1,690, Dec. 1989.
[17] A.L.N. Reddy and P. Banerjee, "Gracefully degradable disk arrays," *Proc. FTCS*, pp. 401-408, 1991.
[18] K. Salem and H. Garcia-Molina, "Disk striping," *Proc. Int'l Conf. Data Eng.*, pp. 336-342, 1986.

Chan-Ik Park received the BS degree in electronics engineering from the Seoul National University, Seoul, Korea, in 1979, and the MS and PhD degrees in electrical and electronics engineering from the Korea Advanced Institute of Science and Technology, Seoul, Korea, in 1985 and 1988, respectively. He joined the Department of Computer Science, Pohang University of Science and Technology in 1989, where he is currently associate professor. He was a visiting scientist in Parallel Systems Group at the IBM Thomas J. Watson Research Center from February 1991 until February 1992. His research interests include operating systems, parallel processing, distributed real-time systems, and combinatorial optimization.