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 \(\text{MTTF}_{\text{sys}} = \frac{\text{MTTF}_{\text{disk}}}{N}\) where \(\text{MTTF}_{\text{disk}}\) 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 cannot 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...
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 de-
scribes 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.

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 cor-
recting 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 related to k
error correcting groups for k ≥ 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 space are occupied for storing
parity information, called redundancy rate. For example, 50% re-
dundancy 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_j^i
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_k represents the data
block which is included in both the ith and the jth groups, e.g.,
P_0 is computed over D_9 and D_3, P_1 is computed over D_12 and
D_10, 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_9, P_1, and D_12 are lost. Then D_23 can be
reconstructed from D_9 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_k 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 × 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_j be an element of an RM which has a value
ranging from −1 to M − 1. According to the value of RM_j,
the interpretation differs as follows.

- RM_j = -1: A parity block of the disk j belongs to the error
correcting group i.
- RM_j = 0: nothing.
- RM_j = k for 1 ≤ k ≤ 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 ≤ k ≤ 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.

1 0 1 2 3
0 1 2 3 4
1 0 1 2 3
2 1 2 3 4
3 0 1 2 3

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, · · · , 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 2 2 2 2
1 2 1 1 1
2 1 1 1 1
2 1 2 2 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 = [c0, c1, · · · , cN−1], an i-edge (c0, c1) is defined between c0 and c1 if c0 = c1 > 0. Note that there exist M − 1 i-edges in C.

Definition 4. External edge (e-edge): Given two placement vectors C = [c0, c1, · · · , cN−1] and C = [c0, c1, · · · , cN−1], an external edge (c0, c1) is defined between c0 and c1 if c0 ≠ 0 or c1 ≠ 0. An e-edge is called a-edge if (c0 # 0 and c1 # 0), s-edge otherwise. Specifically, a-edge and s-edge are called ap-edge and sp-edge, respectively, if (c0 = 1 and c1 = −1).

Definition 5. Reachable and isolated path: Given two placement vectors C and C, 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 × N matrix. Then, an N × N matrix is c-independent if any pair of columns are c-independent.

For example, consider two placement vectors shown in Fig. 6a, C0 = [−1, 3, 2, 1, 0, −1, 3, 2, 1] and C1 = [1, 2, 3, 0, −1, 3, 2, 1]. The i-edges in C0 include (c0, c0), (c0, c2), and (c0, c4) while those in C1 include (c1, c1), (c1, c3), and (c1, c5). The a-edges for C0 and C1 include (c0, c0), (c0, c2), (c0, c4), (c0, c6), (c1, c1), and (c1, c3). Among these a-edges, (c0, c0) and (c1, c1) are called ap-edges since c0 = −1 and c1 = −1, respectively. The s-edges include (c0, c2) and (c1, c3). 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 (c6, c6), (c6, c6), (c6, c6), and (c6, c6) while the other consists of (c6, c6), (c6, c6), (c6, c6), and (c6, c6). Therefore, C0 and C1 are not c-independent. We could see that there is an isolated open path in Fig. 6b consisting of (c6, c6), (c6, c6), (c6, c6), and (c6, c6).

Theorem 1. Given any N × 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 c0 and c1 are 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 (c0, c1) 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 (c0, c1) 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 c0 and c1 are c-independent, the failure of disk i and j can be corrected.

Conversely, assume that c0 is isolated. Let φi be the set of nonzero entries connected to c0, and assume no s-edge is
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 \(1 \leq i, j \leq M - 1\), 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 \(1 \leq i \leq M - 1\). In other words, the group intervals of positive entries are determined as \(GI(M-1) = 2(M-2), GI(M-2) = 2(M-3), \ldots,\) 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 \(M\), \(R_{(M,N)} = [C_0, C_1, \ldots, C_{N-1}]\), as shown in Fig. 7. Note that the matrix is constructed simply by shifting down the seed column \(C_0\). At \(C_{p+2M/2}\) 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 \(C_{p+2M/2}, C_{p+2M+1}, \ldots\) 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 \(GZ(\cdot)\) instead of \(GI(\cdot)\) 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. To begin with, construct a placement may satisfy the following conditions.

1. \(R_{(M,N)}\) is \(c\)-independent.
2. \(R_{(M,N)}\) is \(c\)-independent, then it becomes an \(RM\) matrix.
3. \(R_{(M,N)}\) is \(c\)-independent, then it becomes an \(RM\) matrix.

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 \(N\):

3) Construct an \(N \times N\) matrix \(R_{(M,N)}\) column-by-column.
PARK: EFFICIENT PLACEMENT OF PARITY AND DATA TO TOLERATE TWO DISK FAILURES IN DISK ARRAY SYSTEMS

<table>
<thead>
<tr>
<th>-1</th>
<th>0</th>
<th>M-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-1</td>
<td>-1</td>
<td>M-2</td>
</tr>
<tr>
<td>M-2</td>
<td>M-1</td>
<td>M-3</td>
</tr>
</tbody>
</table>

Fig. 7. An $N \times N$ matrix for $N$ and $M$, $R(M,N)$.

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

4) We have to check $N - 1$ pairs of columns of $R(M,N)$ ($C_0$, $C_1$), ..., ($C_0$, $C_{N-1}$) 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.

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 ($C_0$, $C_1$), ($C_0$, $C_2$), ..., and ($C_0$, $C_{M-1}$) since they are not wrapped-around columns and the number of 0 entries are $M + k + 1 - 1$. Thus, we will consider only the following column pairs ($C_0$, $C_{M+k}$), ..., and ($C_0$, $C_{M-1}$). 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 $C_{M+1}$, ..., 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
LEMMA 3. Isolated open paths cannot exist for any pair of columns of \( R_{MN} \) if \( N \geq 3M - 2 \) 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 \( j \) such that \( G(i) = G(j) \). Considering the minimum and maximum values of \( G(i) = G(j) \), we get that \( 0 \leq G(i) \leq 2(M - 2) \) and \( M + k \leq G(j) \leq 3M - 4 + k \). If we set the value of \( k \) such that the minimum value of \( G(i) \) is greater than the maximum value of \( G(j) \), then it is impossible for \( G(i) = G(j) \) 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 4. Any pair of columns of \( R_{MN} \) is \( c \)-independent if either \( N \) is an odd number such that \( N \geq 3M - 2 \) or \( N \geq 4M - 5 \).

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
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 \( M = 100p \).

2) Construct a placement vector for \( M \) and \( N \):

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 \( N_{\text{required}} \) such that \( N_{\text{required}} \geq 3M - 2 \). This number becomes the required number of disks.

5) If \( N \geq N_{\text{required}} \) 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 \( R_{(M,N)} \) (\( C_0, C_1 \)), \( \ldots \), (\( C_{N-2}, C_{N-1} \)) 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**

<table>
<thead>
<tr>
<th>( M )</th>
<th>Minimum ( N ) obtained by the algorithm</th>
<th>Minimum ( N ) obtained by Theorem 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>29</td>
</tr>
</tbody>
</table>

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) \( M = 100p = 4 \).
2) Construct a placement vector for \( M = 4 \) and \( N = 8 \):

\[
\begin{array}{cccccccc}
-1 & 3 & 2 & 1 & 1 & 2 & 3 & 0 \\
\end{array}
\]

3) Construct an \( 8 \times 8 \) matrix \( R_{(4,8)} \) by using a placement vector as a seed column.
1) By Theorem 2, choose \( N_{\text{reqired}} \). Then, \( N_{\text{required}} = 11 \).
2) Since \( N < N_{\text{requird}} \) we cannot guarantee that a constructed 
\( 8 \times 8 \) matrix \( R_{(4,3)} \) is \( c \)-independent. Thus, we have to check \( N-1 \) pairs of columns \((C_0, C_1), \ldots, (C_{N-1}, C_{N-1})\) for 
\( c \)-independence.
3) Since there is an isolated closed path for \( C_0 \) and \( C_4 \), the 
\( R_{(4,3)} \) 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 \( N = 7 \):

\[
\begin{array}{cccccccc}
0 & -1 & 0 & 2 & 1 & 1 & 2 & 0 \\
1 & 2 & -1 & 0 & 0 & 2 & 1 & 1 \\
2 & 1 & 2 & -1 & 0 & 0 & 2 & 1 \\
3 & 1 & 1 & -1 & 0 & 0 & 2 & 1 \\
4 & 2 & 1 & 1 & 2 & 1 & 1 & 2 \\
5 & 0 & 0 & 1 & 1 & 2 & -1 & 0 \\
6 & 0 & 0 & 1 & 1 & 2 & -1 & 1 \\
\end{array}
\]

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

\[
\begin{array}{cccccccc}
0 & -1 & 0 & 2 & 1 & 1 & 2 & 0 \\
1 & 2 & -1 & 0 & 0 & 2 & 1 & 1 \\
2 & 1 & 2 & -1 & 0 & 0 & 2 & 1 \\
3 & 1 & 1 & -1 & 0 & 0 & 2 & 1 \\
4 & 2 & 1 & 1 & 2 & 1 & 1 & 2 \\
5 & 0 & 0 & 1 & 1 & 2 & -1 & 0 \\
6 & 0 & 0 & 1 & 1 & 2 & -1 & 1 \\
\end{array}
\]

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

\[
\begin{array}{cccccccc}
P_0 & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 \\
D_{13} & D_{14} & D_{15} & D_{16} & D_{17} & D_{18} & D_{19} & D_{20} \\
D_{21} & D_{22} & D_{23} & D_{24} & D_{25} & D_{26} & D_{27} & D_{28} \\
\end{array}
\]

**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**


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.