A Gracefully Degradable Declustered RAID Architecture with near Optimal Maximal Read and Write Parallelism

Siu-Cheung Chau *
Dept. of Physics and Computing, Wilfrid Laurier University,
Waterloo, Ontario, Canada, N2L 3C5.
email: schau@wlu.ca

Ada Wai-Chee Fu
Dept. of Computer Science and Engineering,
Chinese University of Hong Kong, Shatin, Hong Kong
e-mail: adafu@cse.cuhk.edu.hk

Abstract *A new layout method Prime-groups is proposed to evenly distribute parity groups for declustered RAID. Prime-groups satisfies most of the layout goals for a good declustered RAID layout. For the goals that are not satisfied, it is near optimal. A new layout goal maximal write and reconstruction parallelism is also proposed. If a layout satisfies the new goal, all the surviving disks can be read in parallel and can be rewritten in parallel during reconstruction and reconfiguration. Prime-groups satisfies the new goal when the write request begins in the first disk of the array. It is also near optimal in term of declustering ratio when p is a prime. Prime-groups can compliment the layouts proposed by Alvarez et. el.[2] for the criteria to obtain a good layout is quite different between Prime-groups and those proposed by Alvarez et. el.[2].*

Keywords: Fault-Tolerant, RAID, Declustered RAID, Disk Arrays

1 Introduction

Many applications such as on-line transaction processing, image processing, and scientific computing, require high throughput from cost-effective storage sub-systems. The mass storage sub-systems for traditional mainframe computers have a much higher mean time between failure (MTBF) rating than disk systems of personal computers. However, the cost of the mass storage sub-systems for mainframes is much higher than those used for personal computers. A Redundant Array of Independent Disks (RAID) [16] offers significant advantages over individual disks. RAID is a method to achieve the same or even higher level of reliability as the mass storage systems for mainframes using only an array of inexpensive disks. By using multiple disk drives, RAID can achieve better transfer bandwidths and reductions on seek and rotational latencies compared to an individual disk. The reduction in both latencies are possible because a file can be stored in multiple disks. Reading and writing to the file can be done in parallel in all the disks.

A survey of the issues and methods of RAID can be found in [6]. A RAID system is described in [18]. Some other references on RAID are [11, 5]. The basic method of RAID is to dedicate one disk as the parity disk for the remaining data disks. If a data disk fails, the data of the failed disk can be reconstructed from the information stored in the parity disk and the other data disks. Therefore, higher reliability is achieved. However, with this method, the parity disk may be-

*This research was supported by the National Sciences and Engineering Research Council of Canada

come the bottleneck and slow down the system substantially during failure. This happens because every write to any data disk requires a write to the parity disk. An enhanced approach is to eliminate the dedicated parity disk by distributing the parity information among all the disks.

For example, the RAID level-5 left-symmetric parity placement is one such method. In this method, data is interleaved across disks in **blocks** of arbitrary size, a block is also called a **striping unit**. In Figure 1, each vertical column in the table represents a disk, consecutive numbers refer to consecutive blocks of data and the P_i blocks are the parity blocks for data blocks in each row. In such a RAID system, a *small read request* that requires only data within a single block accesses only a single data disk. A *small write request* must update the requested data block and must compute the parity and update the parity block. The parity is computed by noting the difference between the new and the old data and applying the difference to the parity block. A write request thus requires four disk I/Os: two to read the old data and the old parity, one to write the new data, and one to write the new parity. This is referred to as a **read-modify-write** procedure.

We refer to a set of blocks, along with the parity block, a **parity group**. The table in Figure 1 can be used as a lookup table to store the distribution of the parity groups and group members among 5 disks. If each disk contains more than 5 blocks, we may logically duplicate the table as many times as necessary to ensure a mapping of the parity groups to all the disk blocks. This table is used for each read and write operation to look up the set of blocks to access. We shall call such a table a **parity groups' distribution lookup table**, or a **lookup table**, for short. In Figure 1, each row of the table contains exactly one parity group.

Unfortunately, with this approach, a disk failure still cause great performance degradations. If the disk represented by the leftmost column has failed, each read access to this disk will trigger a read access to each of the remaining disks to reconstruct the data stored in the failed disk. Suppose that the load is evenly distributed among all disks, the failure of a disk will result in a 100% increase in load to all the remaining disks.

0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

Figure 1: RAID5 left-symmetric parity placement

g0	g0	g0	g0	g1	g1	g1	g1
g2	g2	g2	g2	g3	g3	g3	g3
g4	g4	g4	g4	g5	g5	g5	g5
g6	g6	g6	g6	g7	g7	g7	g7

Figure 2: Standard Multiple RAID-5 left-symmetric parity placement

Muntz and Lui [14, 10, 6] proposed a **declustered RAID** to solve the problem of doubling in load when a disk failure occurs. There are two major enhancements comparing a declustered RAID to a RAID-5.

1. The number of disks n, is greater than the parity group size k. If the parity groups are distributed evenly, concurrent write to the disk array may be possible.

2. The parity groups are distributed evenly among all the disk blocks.

For example in Figure 2, gi represents members of parity group i, the first enhancement allows concurrent write to a disk in the first four columns and a disk in the last four columns. However, It does not contribute to any improvement if a disk fails. If the leftmost disk fails, the failed disk triggers a 100% increase in load

g0	g0	g0	g0	g1	g1	g1	g1
g2	g2	g3	g3	g2	g2	g3	g3
g4	g5	g4	g5	g4	g5	g4	g5
g6	g7	g7	g6	g6	g7	g7	g6

Figure 3: Declustered parity placement

to the next 3 disks but none to the disks in the last four columns. The second enhancement decreases the load increases to the surviving disks if a disk fails. An example is shown in Figure 3. If the leftmost disk has failed, the increase in load to the surviving disks is at most 75%. If we have an efficient method to evenly distribute the parity groups, the declustered RAID method will be much better than RAID-5. That is, we should have a good method to map data and redundant information onto the disks of the disk array. The mapping is called a **disk array layout**. The goals for a good layout as stated in [10] are listed below.

1. *Single failure correcting.* No two data blocks in the same parity group are mapped to the same physical disk. This makes recovery possible from a single disk failure.

2. *Efficient mapping.* The mapping of a logical block address to the actual physical disk blocks should be efficiently computable, with low time and space requirement.

3. *Large write optimization.* Each parity group contains a contiguous interval of data and a large write operation that writes the contiguous data of the entire parity group does not have to read the prior contents of any disk.

4. *Maximal read parallelism.* A read of contiguous user data with size equal to the size of a data block times the number of disks should induce a single block read on all the disks in the array.

5. *Distributed degradation and distributed reconstruction.* Distributed degradation means that when any disk fails, its workload should be evenly distributed across all other disks in the array. Distributed reconstruction means that when a failed disk is replaced or repaired, its reconstruction workload should be evenly distributed across all other disks in the RAID. These two requirements are the same since the same disk blocks are accessed for both cases.

6. *Distributed parity.* Parity information should be evenly distributed across the array because write operations access the parity block. With evenly distributed parity, the disk array will have a higher probability of allowing more concurrent write operations and will avoid having a bottleneck.

For very large write where we might want to write to all the disks in parallel. We define one more layout goal called maximal write and reconstruction parallelism.

7. *Maximal write and reconstruction parallelism.* A write of contiguous data and parity with size equal to the size of a data block times the number of disks should induce a single block write on all the disks in the array.

This would be very useful when a disk fails and we want fast parallel reconstruction. All the surviving disks can be read in parallel and can be rewritten in parallel. Layout that satisfies the above goal can also allow large write optimization to more than one parity groups simulataneously. The left-symmetric RAID-5 layout satisfies all of the above goals. However, the increase in load is 100% to all surviving disks if a disk fails.

Let b be the number of blocks in each disk, n be the number of disks in the array, k be the maximum size of each parity group, and p be the maximum number of parity groups that appears in a row of the lookup table. The value of $\frac{k-1}{n-1}$ determines the fraction of each surviving disk that must be read during the reconstruction of a failed disk. This value is called the **declustering ratio** in [14].

To achieve some of the layout goals, Reddy and Banerjee [17] proposed a method to solve the distribution problem. Their method uses some results of Hadamard matrices and difference sets to solve the problem for $\frac{n}{2}$. Merchant and Yu [12] proposed a method which uses Thorps's shuffle to distribute the parity groups evenly. The method works for any n and k, and requires $O(log\ b + log\ n)$ operations and $O(log\ b + n\ log\ n)$ storage space to find out which parity group a block is in. Holland and Gibson [10] in 1992, proposed the use of the balanced block design method and balanced incomplete block design (BIBD) method to find an evenly distributed parity groups' lookup table. Ng and Mattson [15] also proposed the same method in 1992. However, a balanced block can be too large and balanced incomplete block designs are not known for a large number of possible values of n and k.

Alvarez, Burkhard, and Cristian[3] in 1997 presented the first method that can mask any given number of failures. Their method requires an optimal amount of redundant storage space and spreads reconstruction accesses uniformly over all the disks in the presence of a disk failures without requiring large layout tables in the disk controller memory. They based their method on information dispersal[1] where a sequence $F = (d_1, d_2, ..., d_m)$ of m integers is coded into a sequence of $m+f$ integers in such a way that any m of the $m+f$ integers are enough to recover F. However, they did not consider read and write parallelism in their paper.

Alvarez, Burkhard, Stockmeyer and Cristian[2] in 1998 presented a method that satisfies all six layout goals. They also proved that it is not possible to find a layout that satisfies all the goals for arbitrary n and k. Let the notation $x|y$ means that y is divisible by x. That is, $y = cx$ where x, y, and c are integers. The ideal layout is only possible when $k|b$, $(n-1)|(b(k-1))$, and for very limited values of n and k. The values are listed below.

1. $k = n$, which is RAID-5.
2. $k = n - 1$.
3. $k = 2$.
4. $k = 3, n = 5$ or 7.
5. $k = 4, n = 7$.

In the same paper, they also presented two other layouts. The first one, Prime satisfies five of the seven goals, and the maximal read parallelism is very close to optimal. For a layout L, they define the function $\tau_L(o)$, the *parallel read count*, to be the maximum number of data units that any disk must supply when reading any o consecutive units of data. It is obvious that $\tau_L(o) \geq \lceil o/n \rceil$. If $\tau_L(o) = \lceil o/n \rceil$, the layout L has maximal read parallelism. The layout created by Prime has a parallel read count of $\tau_L(o) \leq \lceil o/n \rceil + 1$. However, in order for Prime to work, n has to be a prime number.

The second layout presented is Relpr - relatively prime, which satisfies four out of the seven layout goals. The maximal read parallelism is again very close to optimal. Layout by Relpr has a parallel read count of $\tau_L(o) \leq \lceil 0/n \rceil + 1$ and a reconstruction workload bounded above by $(k-1)/\phi(n)$. $\phi(n)$ is the Euler's totient function where ϕ counts the number of positive integers less than n that are relatively prime to n. The Relpr method generates layout with good declustering ratio when $\phi(n)$ is very close to n.

In this paper, we propose a new layout scheme Prime-groups which can satisfy five of the seven layout goals when the number of parity groups p, in a row of the table is prime and $n = p^m$. If p is not prime or $n \neq p^m$, four of the seven layout goals are satisfied. The resulting layout is also near optimal to the other three goals. The new layout compliment the Prime and Relpr methods on cases where they have no solution or if Relpr layout has a poor distributed reconstruction workload.

This paper is organized as follows. Section 2 describes the basic approach. Section 3 presents the new method Prime-groups for the special case of $n = p^2$. Section 4 generalizes this to the case of $n \leq p^m$, for any integer $m \geq 2$. Section 5 discusses the properties of the Prime-groups. Section 6 concludes the paper.

For most of this paper, we make the assumption below and the same assumption is made in all previous work that we know, including [3, 13, 9, 10]. A reason behind this assumption is that contiguous user data is typically distributed over the disks to improve the performance and this will make data accesses more evenly distributed.

Assumption: Each disk has the same read and write load, and the amount of read and/or write operations in a disk are uniformly distributed among all the blocks.

2 Basic approach

We propose the use of a small lookup table to store the distribution of the parity groups. We shall show that for a write operation, we only have to perform a few simple calculations to determine which row of the table we should use. Since the write operation takes much longer time compared to a few processor operations, the increase in access time would be negligible. Furthermore, the small lookup table can be stored at the disk controller level.

The lookup table is constructed by using **circulant matrices**. A circulant matrix is a matrix of size $m \times m$. Each column consists of the integers from 1 to m. Once the value of the first column is chosen, the values of the elements for the next column are obtained by performing a **rotation** with respect to the previous column. The rotation is done by shifting the value of the element at row $i+1$ to row i for $i \in \{2..m\}$ and shifting the value of the element at the first row to the last row. The first column of a circulant matrix is either the column $\{1, 2, 3, ..., m\}^{-1}$, or the result of a number of rotations with respect to this column. For example, the following are 3×3 circulant matrices.

$$M_{3,1} = \begin{matrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{matrix} \qquad M_{3,2} = \begin{matrix} 2 & 3 & 1 \\ 3 & 1 & 2 \\ 1 & 2 & 3 \end{matrix} \qquad M_{3,3} = \begin{matrix} 3 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 1 \end{matrix}$$

If the values of the first column of a circulant matrix are all distinct, then the values stored in each row are also all distinct. Furthermore, the resulting matrix will have the property that each value appears only once in each row or column. Note that the matrix $M_{3,2}$ can be obtained from $M_{3,1}$ by rotating the entire rows of the matrix. We say that $M_{3,2}$ is the result of one **rotation w.r.t.** $M_{3,1}$. With the three circulant matrices $M_{3,1}, M_{3,2}, M_{3,3}$, and three other matrices $I_{3,1}, I_{3,2}$, and $I_{3,3}$, we can construct the parity groups' distribution lookup table for 9 disks with 3 parity groups in each row. The matrices $I_{3,1}, I_{3,2}$, and $I_{3,3}$, are listed below.

$$I_{3,1} = \begin{matrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{matrix} \qquad I_{3,2} = \begin{matrix} 2 & 2 & 2 \\ 3 & 3 & 3 \\ 1 & 1 & 1 \end{matrix} \qquad I_{3,3} = \begin{matrix} 3 & 3 & 3 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{matrix}$$

The lookup table is constructed by combining these matrices together as follows:

	Disk								
row	1	2	3	4	5	6	7	8	9
0	**1**	**2**	**3**	1_{1}	2_{3}	3_{5}	1_{2}	2_{4}	3_{6}
1	2_{7}	3_{9}	1_{11}	**2**	**3**	**1**	2_{8}	3_{10}	1_{12}
2	3_{13}	1_{15}	2_{17}	3_{14}	1_{16}	2_{18}	**3**	**1**	**2**
3	**1**	**2**	**3**	2_{19}	3_{21}	1_{23}	3_{22}	1_{24}	2_{20}
4	2_{25}	3_{27}	1_{29}	**3**	**1**	**2**	1_{30}	2_{26}	3_{28}
5	3_{31}	1_{33}	2_{35}	1_{34}	2_{36}	3_{32}	**2**	**3**	**1**
6	**1**	**2**	**3**	3_{37}	1_{39}	2_{41}	2_{42}	3_{38}	1_{40}
7	2_{43}	3_{45}	1_{47}	**1**	**2**	**3**	3_{46}	1_{48}	2_{44}
8	3_{49}	1_{51}	2_{53}	2_{54}	3_{50}	1_{52}	**1**	**2**	**3**
9	**1**	1_{55}	1_{56}	**2**	2_{57}	2_{58}	**3**	3_{59}	3_{60}
10	2_{61}	**2**	2_{62}	3_{63}	**3**	3_{64}	1_{65}	**1**	1_{66}
11	3_{67}	3_{68}	**3**	1_{69}	1_{70}	**1**	2_{71}	2_{72}	**2**

Figure 4: The layout for n=9 and p=3

$$\begin{matrix} M_{3,1} & M_{3,1} & M_{3,1} \\ M_{3,1} & M_{3,2} & M_{3,3} \\ M_{3,1} & M_{3,3} & M_{3,2} \\ I_{3,1} & I_{3,2} & I_{3,3} \end{matrix}$$

When the matrices in the above table are expanded, we get the table in Figure 4. In this table, an entry of i ($1 \leq i \leq 3$) represents a block of data belonging to parity group i in each row. The entries of i's in a row together form a parity group. The entries in bold font are the parity blocks and the subscripts indicate the order of the user data units. Suppose disk 4 has failed. A read operation to block (row) 5 of disk 4 requires an operation to disk 2 and disk 9. It is easy to see that the increase in load to each surviving disk is 3/12=25%. In this case, $n = 9$, $k = 3$, and $\frac{k-1}{n-1} = 0.25$. The layout above has optimal distributed degradation and distributed reconstruction.

The parity groups' distribution lookup table can be used for other blocks besides block 0 to block 11. We can use the lookup table for blocks 12 to 23, 24 to 35 and so on. After a disk failure, there will be 3/12 increase in load in each segment for each surviving disk. Furthermore, it is quite easy to determine which row of the lookup table should be used for a particular block. For data in block i, only one operation, i mod 12, is necessary to determine the row in the lookup table which should be used. The size of the lookup table is also independent of the number of blocks b, in a disk.

3 A new layout scheme for declustered RAID

We now develop a general scheme which we called Prime-groups, based on the findings from the example in the previous section. Let us define a number of terms. Given a matrix column $\{A_1, A_2, A_3, ..., A_{m-1}, A_m\}^{-1}$, a **rotation with respect to (w.r.t.)** this matrix column yields a matrix column $\{A_2, A_3, A_4, ..., A_m, A_1\}^{-1}$. If C_m is a rotation w.r.t. C_{m-1}, C_{m-1} is a rotation w.r.t. C_{m-2}, ..., and C_2 is a rotation w.r.t. C_1, then we say that C_m is obtained by $m-1$ **rotations w.r.t.** C_1. For simplicity, unless otherwise specified, the term rotation in the following would refer to a rotation w.r.t. the corresponding first matrix sub-column in the lookup table. Given a matrix A, we shall refer to the element at the the i-th column of the j-th row the element $A[i, j]$. The first column (row) is column 0 (row 0).

A **circulant matrix** $M_{i,j}$ is an $i \times i$ matrix. The first column, column 0, of this matrix contains the values 1 to i, which is obtained by $j-1$ rotations w.r.t. $\{1, 2, 3, ...i\}^{-1}$, for some j where $1 \le j \le i$. (Hence $M_{i,j}[0, 0] = j$.) The m-th column is obtained by $m-1$ rotations w.r.t. the first column. By the definition of circulant matrix, we can see that for $M_{i,j}$, row m of column 0 ($M_{i,j}[0, m]$) contains the value r where $r = (j + m) \bmod i$ if $(j + m) \bmod i$ is non-zero, and $r = i$ otherwise.

A **compensation matrix** $I_{i,j}$ is an $i \times i^m$ matrix. The first column of this matrix contains the values 1 to i, which is obtained by $j-1$ rotations w.r.t. $\{1, 2, 3, ...i\}^{-1}$, for some j where $1 \le j \le i$. Hence $I_{i,j}[0, 0] = j$. All the other columns of $I_{i,j}$ are identical to the first column.

Let p be the number of parity groups in each row of the lookup table. We assume that p is a prime number and $n = p^2$. Some of these restrictions will be relaxed in Section 4. The parity groups' distribution lookup table T can be constructed by combining the matrices $M_{p,j}$ and $I_{p,j}$, for $1 \le j \le p$. T can be considered as a matrix M whose elements are matrices of the form $M_{p,j}$ and $I_{p,j}$, for $1 \le j \le p$. We call $M_{p,j}$ and $I_{p,j}$ the **sub-matrices** of M. Let us call the rows and columns of such sub-matrices in M the **matrix rows** of M and **matrix columns** of M, respectively. We call the first row of matrices that make up the lookup table, **matrix row** 0; and the first column of matrices, **matrix column** 0. We also call a column of integers in T which corresponds to a disk a **disk column**.

Column-wise construction: Matrix column 0 consists of p sub-matrices $M_{p,j}$ in the first p rows and one $I_{p,j}$ in the last matrix row, for some value of j, where $1 \le j \le p$. Let the matrices in matrix column m be $\{M_{p,j_1}, M_{p,j_2}, M_{p,j_3}, ..., M_{p,j_p}, I_{p,j'}\}^{-1}$, then the matrices in matrix column $m+1$ will be $\{M_{p,j_1}, M_{p,j_2+1}, M_{p,j_3+2}, ...M_{p,j_p+(p-1)}, I_{p,j'+1}\}^{-1}$.

For example, a lookup table $M^2_{5,1}$ for $n = 25$ and $p = 5$ is shown below.

$$
\begin{array}{ccccc}
M_{5,1} & M_{5,1} & M_{5,1} & M_{5,1} & M_{5,1} \\
M_{5,1} & M_{5,2} & M_{5,3} & M_{5,4} & M_{5,5} \\
M_{5,1} & M_{5,3} & M_{5,5} & M_{5,2} & M_{5,4} \\
M_{5,1} & M_{5,4} & M_{5,2} & M_{5,5} & M_{5,3} \\
M_{5,1} & M_{5,5} & M_{5,4} & M_{5,3} & M_{5,2} \\
I_{5,1} & I_{5,2} & I_{5,3} & I_{5,4} & I_{5,5}
\end{array}
$$

Let $M^2_{p,j}$ be the lookup table obtained by the above method, if we leave out matrix row p, which contains the compensation matrices, the resulting matrix is denoted by $K^2_{p,j}$.

Lemma 1 *For a matrix $M^2_{p,j}$, for the entry $M_{p,i}$ at matrix row r and matrix column c, $i = (j + rc)$ mod p, and the number of rotations w.r.t. $M_{p,j}$ is given by (rc) mod p.*

For example, for $M^2_{5,1}$, for the entry at row 2 and column 3 is $M_{5,2}$, we have $2 = (1 + 2 \times 3)$ *mod* 5. The number of rotations of $M_{5,2}$ w.r.t. $M_{5,1}$ is given by (2×3) *mod* 5.

Lemma 2 *The matrix $M_{p,j}$ is symmetrical.*

Lemma 3 *The matrix $K^2_{p,j}$ has distinct matrices in matrix rows 2 to $(p-1)$ and matrix columns 2 to $(p-1)$.*

Lemma 4 *Assuming $n = p^2$ and the lookup table $K^2_{p,j}$ is used. If a disk has failed, the load increase to each of the remaining disks is at most $1/p$.*

Theorem 1 *Assuming $n = p^2$ and the lookup table $M^2_{p,j}$ is used. If a disk has failed, the load increase to each of the remaining disks is $\frac{1}{p+1}$ which is optimal in terms of distributed degradation and distributed reconstruction.*

If p is not prime, we can still construct $M^2_{p,j}$. The lookup table $M^2_{4,1}$ is shown below. However, since 4 is not prime, Lemma 3 and 4 do not apply to $M^2_{4,1}$. For example, in row 2 of $M^2_{4,1}$, there are two occurrences of $M_{4,1}$ and $M_{4,3}$.

$M_{4,1}$	$M_{4,1}$	$M_{4,1}$	$M_{4,1}$
$M_{4,1}$	$M_{4,2}$	$M_{4,3}$	$M_{4,4}$
$M_{4,1}$	$M_{4,3}$	$M_{4,1}$	$M_{4,3}$
$M_{4,1}$	$M_{4,4}$	$M_{4,3}$	$M_{4,2}$
$I_{4,1}$	$I_{4,2}$	$I_{4,3}$	$I_{4,4}$

For $n < p^2$, we can construct the lookup table as if $n = p^2$. We then delete the rightmost $p^2 - n$ columns in the table. Let us call the resulting table $M^2_{p,j,n}$ It is obvious that the proofs in Lemma 4 and Theorem 1 still apply and the increase in load is still $\frac{1}{p+1}$ for each surviving disk. Therefore we have the following theorem.

Theorem 2 *Assuming $2p \leq n \leq p^2$ and the lookup table $M^2_{p,j,n}$ is used. If a disk has failed, the load increase to each of the remaining disks is $\frac{1}{p+1}$.*

4 Generalization to $p^{m-1} < n \leq p^m$

For $n > p^2$ we can apply a similar approach as in $n = p^2$. We define recursively the matrices $M^m_{p,j}$. We have already defined $M^2_{p,j}$. Let $M^m_{p,j}$ be a matrix. The matrix has matrices $M^{m-1}_{p,j}$ as its elements. We call these the **sub-matrices** of $M^m_{p,j}$. A row (column) of sub-matrices in $M^m_{p,j}$ is called a **matrix row (matrix column)**. The elements $M^{m-1}_{p,m}$ are arranged in a similar way as those in $M^2_{p,m}$. **Compensation matrices** $I^m_{p,i}$ where $1 \leq i \leq p$, are added to each matrix column. $I^m_{p,1}$ is a $p \times p^{m-1}$ matrix where the 1st row contains all 1's. The second row contains all 2's. The third row contains all 3's. The j-th row contains all j's and the p-th row contains all p's. $I^m_{p,2}$ is obtained by rotating the rows of $I^m_{p,1}$ once. $I^m_{p,j}$ is obtained by rotating the rows of $I^m_{p,1}$ j-th times.

$M^m_{p,j}$ is the layout for $n = p^m$. For example, the matrix $M^{m+1}_{5,1}$ is the layout for $p = 5$ and $n = p^{m+1}$, is shown below. We can see the resemblance to $M^2_{5,1}$. $I^m_{5,1}$ is a 5 by 5^{m-1} matrix where the 1st row contains all 1's. The second row contains all 2's. The third row contains all 3's. The fourth row contains all 4's and the fifth row contains all 5's.

$M^m_{5,1}$	$M^m_{5,1}$	$M^m_{5,1}$	$M^m_{5,1}$	$M^m_{5,1}$
$M^m_{5,1}$	$M^m_{5,2}$	$M^m_{5,3}$	$M^m_{5,4}$	$M^m_{5,5}$
$M^m_{5,1}$	$M^m_{5,3}$	$M^m_{5,5}$	$M^m_{5,2}$	$M^m_{5,4}$
$M^m_{5,1}$	$M^m_{5,4}$	$M^m_{5,2}$	$M^m_{5,5}$	$M^m_{5,3}$
$M^m_{5,1}$	$M^m_{5,5}$	$M^m_{5,4}$	$M^m_{5,3}$	$M^m_{5,2}$
$I^m_{5,1}$	$I^m_{5,2}$	$I^m_{5,3}$	$I^m_{5,4}$	$I^m_{5,5}$

Theorem 3 *For $n = p^m$, where m is an integer and $m \geq 2$, the layout $M^m_{p,j}$ is optimal in terms of distributed degradation and distributed reconstruction. That is, the load increase is equal to $\frac{k-1}{n-1}$.*

The technique used for $n < p^2$ can also be applied in the case where $p^{m-1} < n < p^m$. For $n < p^m$, the rightmost $p^m - n$ block columns of $M^m_{p,j}$ can be deleted, resulting in the table $M^m_{p,j,n}$. For $M^m_{p,j,n}$, if a disk has failed, the surviving disks will have at most $\sum_{i=1}^{m-1} p^i / \sum_{i=1}^{m} p^i$ increase in load.

Theorem 4 *For $n \leq p^m$, where m is an integer and $m \geq 2$, if layout $M^m_{p,j,n}$ is used, when a disk has failed, the load increase to each of the remaining disks is at most $\sum_{i=1}^{m-1} p^i / \sum_{i=1}^{m} p^i$.*

5 Properties of the new layout

We recall the seven layout goals listed in Section 1. We can assign a data mapping such that successive user data is mapped to successive data units within successive parity groups as shown in Figure 4. The parity blocks are highlighted in bold letters and the subscripts indicate the the order of user data units. It is straightforward to see that Prime-groups has the first three properties of (1) single failure correcting, (2) efficient mapping, and (3) large write optimization. Next we shall discuss the last four properties of distributed parity, distributed degradation and reconstruction, maximum read parallelism, and maximum write and reconstruction parallelism.

5.1 Distributed parity

In Figure 4, the parity blocks are highlighted in bold fonts. This shows one possible pattern of parity block distribution which is even. It is easy to see that this distribution can be generalized to the case with any value of prime p and $n = p^m$.

For a more general cases, let us look at Figure 5. The circled numbers represent the parity blocks. The arrangement of parity blocks shown achieves one of the most even parity distribution possible for the table. This method can be generalized to any value of n which is a multiple of p. For values of n which is not a multiple of p, less even parity distribution may result. However, the distribution can be made even by a technique similar to the one used in [10], namely, to repeat the matrix a number of times, but rotating the columns in each repetition. For example, the configuration in Figure 5 can be repeated 7 times, with each repetition rotating 3 matrix columns to the right. In this way, the number of parity blocks in each column of the combined matrix will be identical.

5.2 Distributed degradation

In the last section, we have shown that our layout is optimal in terms of distributed degradation and reconstruction if p is prime and $n = p^m$. In this section, we will show that Prime-groups is very close to optimal when $n \neq p^m$, p is prime and n is not too small.

Theorem 5 *The Prime-groups layout is very close to optimal in degradation distribution for $n \neq p^m$, p is prime, and n is not too small.*

If n is very small, the above will not hold. For example, in the example of $n = 9$, $k = 3$ in Section 2, if we have instead $n = 6$, $k = 2$ so that the last 3 columns of the lookup table are truncated, the increase in load to the surviving disks on single disk failure is 1/4, while the optimal increase should be 1/5. However, the increase is only equal to another 25% above the optimal increase to the load. Hence, the layout is very close to optimal in terms of distributed degradation and distributed reconstruction if n is not too small.

If p is not prime, the increase in load is less than 50% for all n and p. This happens when the matrix used in the layout has only two different matrices in a column as in the case of $p = 4$.

1 2 3	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3
2 3 1	2 3 1	2 3 1	2 3 1	2 3 1	2 3 1	2 3 1
3 1 2	3 1 2	3 1 2	3 1 2	3 1 2	3 1 2	3 1 2
1 2 3	2 3 1	3 1 2	1 2 3	2 3 1	3 1 2	1 2 3
2 3 1	3 1 2	1 2 3	2 3 1	3 1 2	1 2 3	2 3 1
3 1 2	1 2 3	2 3 1	3 1 2	1 2 3	2 3 1	3 1 2
1 2 3	3 1 2	2 3 1	1 2 3	3 1 2	2 3 1	1 2 3
2 3 1	1 2 3	3 1 2	2 3 1	1 2 3	3 1 2	2 3 1
3 1 2	2 3 1	1 2 3	3 1 2	2 3 1	1 2 3	3 1 2
1 1 1	2 2 2	3 3 3	1 1 1	2 2 2	3 3 3	1 1 1
2 2 2	3 3 3	1 1 1	2 2 2	3 3 3	1 1 1	2 2 2
3 3 3	1 1 1	2 2 2	3 3 3	1 1 1	2 2 2	3 3 3
1 2 3	1 2 3	1 2 3	2 3 1	2 3 1	2 3 1	3 1 2
2 3 1	2 3 1	2 3 1	3 1 2	3 1 2	3 1 2	1 2 3
3 1 2	3 1 2	3 1 2	1 2 3	1 2 3	1 2 3	2 3 1
1 2 3	2 3 1	3 1 2	2 3 1	3 1 2	1 2 3	3 1 2
2 3 1	3 1 2	1 2 3	3 1 2	1 2 3	2 3 1	1 2 3
3 1 2	1 2 3	2 3 1	1 2 3	2 3 1	3 1 2	2 3 1
1 2 3	3 1 2	2 3 1	2 3 1	1 2 3	3 1 2	3 1 2
2 3 1	1 2 3	3 1 2	3 1 2	2 3 1	1 2 3	1 2 3
3 1 2	2 3 1	1 2 3	1 2 3	3 1 2	2 3 1	2 3 1
1 1 1	2 2 2	3 3 3	2 2 2	3 3 3	1 1 1	3 3 3
2 2 2	3 3 3	1 1 1	3 3 3	1 1 1	2 2 2	1 1 1
3 3 3	1 1 1	2 2 2	1 1 1	2 2 2	3 3 3	2 2 2
1 2 3	1 2 3	1 2 3	3 1 2	3 1 2	3 1 2	2 3 1
2 3 1	2 3 1	2 3 1	1 2 3	1 2 3	1 2 3	3 1 2
3 1 2	3 1 2	3 1 2	2 3 1	2 3 1	2 3 1	1 2 3
1 2 3	2 3 1	3 1 2	3 1 2	1 2 3	2 3 1	2 3 1
2 3 1	3 1 2	1 2 3	1 2 3	2 3 1	3 1 2	3 1 2
3 1 2	1 2 3	2 3 1	2 3 1	3 1 2	1 2 3	1 2 3
1 2 3	3 1 2	2 3 1	3 1 2	2 3 1	1 2 3	2 3 1
2 3 1	1 2 3	3 1 2	1 2 3	3 1 2	2 3 1	3 1 2
3 1 2	2 3 1	1 2 3	2 3 1	1 2 3	3 1 2	1 2 3
1 1 1	2 2 2	3 3 3	3 3 3	1 1 1	2 2 2	2 2 2
2 2 2	3 3 3	1 1 1	1 1 1	2 2 2	3 3 3	3 3 3
3 3 3	1 1 1	2 2 2	2 2 2	3 3 3	1 1 1	1 1 1

Figure 5: Lookup table for $M_{3,1}^3$ and $n = 21$

5.3 Maximal read parallelism

The Prime-groups layout scheme has the same level of maximal read parallelism as Alvarez, Burkhard, Stockmeyer and Cristian's[2] Prime and Relpr schemes because all three schemes has the same parallel read count $\tau_L(o) \leq \lceil o/n \rceil + 1$.

Theorem 6 *The Prime-groups layout has a parallel read count of $\tau_L(o) \leq \lceil o/n \rceil + 1$.*

Although Prime-groups layout does not have maximal read parallelism in the general case, maximal read parallelism is achieved when the read starts at the first data block in a row and o is divisibe by $(n - p)$.

The maximum number of rows that have to be read is $o/(n-p)$ instead of $\lfloor o/(n-p)\rfloor + 1$.

5.4 Maximal write parallelism

For a layout L, we define the function $\omega_L(o)$, the *parallel write count*, to be the maximum number of data blocks that any disk must supply when writing any o consecutive blocks of data and parity. It is obvious that $\omega_L(o) \geq \lceil o/n \rceil$. If $\omega_L(o) = \lceil o/n \rceil$, the layout L has maximal write parallelism. The layout created by Prime-groups has a parallel write count of $\omega_L(o) \leq \lceil o/n \rceil + 1$.

Theorem 7 *Prime-groups layout has a parallel write count of* $\omega_L(o) \leq \lceil o/n \rceil + 1$.

Although our layout does not have maximal write parallelism in the general case, maximal write parallelism is achieved when the write starts at the first block in a row and $o = cn$. The maximum number of rows that have to be written is o/n instead of $(o/n+1)$. Hence, maximal write parallelism is achieve when $o = n$. This is extremely useful in reconstruction and reorganization.

Suppose we have p parity groups in a row and a disk has failed. We can reconfigure the data and parity groups by reducing the number of parity groups from p to $p-1$. After the reconfiguration, the disk array can tolerate another disk failure. Since Prime-groups has maximal write parallelism when $o = n$, the refiguration can be done in parallel on all disks. The m-th block of every surviving disks can be read in parallel. The data in the failed disk is reconstructed and the data and parity can then be rewritten back to the m-th block of all the surviving disk. Furthermore, the reconfiguration process can be done in parallel with user's read or write request. When a user requests a read or write to consecutive blocks m_i to m_j, all the disks in the same row of the lookup table where blocks m_i to m_j are located, are read. The data and parity information in each row of the lookup table are then reconstructed and reconfigured. For a read operation, the reconstructed data is given to the user. The reconfigured data and parity is then rewritten back to all the surviving disks. Hence, the read and write user requests become part of the reconstruction and reconfiguration process.

6 Conclusion

A new layout method Prime-groups, to evenly distribute parity groups for declustered RAID is proposed. Prime-groups satisfies four out of seven layout goals: single failure correcting, efficient mapping, large write optimization, and distributed parity. It is near optimal for distributed degradation and distributed reconstruction, maximal read parallelism, and maximal write and reconstruction parallelism. Prime-groups and Alvarez, Burkhard, Stockmeyer and Cristian's[2] Prime and Relpr methods all satisfies large write optimization and have the same parallel read count. Their performane should be roughly the same in terms of read and write response time when there is no disk failure. If a disk has failed and we combine user access and reconfiguration, Prime-groups may perform better for Prime-groups satisfies maximal write and reconstruction parallelism when the write request begins in the first disk of the array.

The optimal layout by Alvarez et. el.[2] is only possible for very limited cases. The Prime layout is also possible only when n is prime and the Relpr layout may not have a very good declustering ratio for certain values of n. Prime-groups layout is near optimal in terms of declustering ratio when p is a prime. Hence, Prime-groups can compliment the layouts proposed by Alvarez et. el.[2] for the criteria to obtain a good layout is quite different between Prime-groups and theirs. Furthermore, Prime-group can also handle multiple faults. The number of parity block in each parity group can be any number f where $f < k$. The information dispersal method as proposed by Alvarez et. el.[3] can then be used.

References

[1] M. Rabin, Efficient dispersal of information for security, load balancing, and fault tolerance. Journal of th ACM, 36(2) April 1989, pages 335-348.

[2] Guillermo A. Alvarez, Walter A. Burkhard, Larry J. Stockmeyer and Flaviu Cristian, Declustered Disk Array Architectures with Optimal and Near-optimal Parallelism, in Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA'98), June 1998, pages 109-120.

[3] G.A. Alvarez, W. A. Burkhard, F. Cristian, Tolerating Multiple Failures in RAID Architectures with Optimal Storage and Uniform Declustering, in Proceedings of the 24th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA'97), Colorado, June 1997, pages 62-72.

[4] M. Blaum, J. Brady, J. Bruck, and J. Menon, Even-odd: An efficient scheme for tolerating double disk failures in RAID architecture, in Proceeding of the Annual International Symposium on Computer Architecture, 1994, pages 245-254.

[5] P.M. Chen et al., RAID: High-Performance, Reliable Secondary Storage, In ACM Computer Surveys, Vol. 26, No. 2, June 1994, pages 145-185.

[6] Peter M. Chen, Edward K. Lee, Garth A. Gibson, R.H. Katz, D.A. Petterson, RAID: High-Performance, Reliable Secondary Storage. In ACM Computing Surveys, Vol. 26, No. 2, June 1994.

[7] Edited by C.J. Colbourn and J.H. Dinitz, The CRC Handbook of Combinatorial Designs. CRC Press, 1996.

[8] Marshall Hall, Jr., Combinatorial Theory, 2nd Edition, Wiley-Interscience series in discrete mathematics, 1986.

[9] Mark Holland and Garth A. Gibson and Daniel P. Siewiorek, Architectures and Algorithms for On-Line Failure Recovery in Redundant Disk Arrays. In Distributed and Parallel Databases, Vol. 2, pages 295-335, 1994.

[10] Mark Holland and Garth A. Gibson, Parity Declustering for Continuous Operation in Redundant Disk Arrays. In Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 23-35. Association of Computing Machinery, New York, 1992.

[11] E. Lee and R. Katz, Performance Consequences of Parity Placement in Disk Arrays, Proceedings of ASPLOS-IV, 1991, pages 190-199.

[12] Arif Merchant and Philip S. Yu, Design and Modeling of Clustered RAID. In Digests of Papers of the 22nd International Symposium on Fault-Tolerant Computing, Pages 140-149. The Computer Society, IEEE, June, 1992.

[13] Arif Merchant and Philip S. Yu, Analytic modeling of clustered RAID with mapping based on nearly random permutation, IEEE Transactions on Computers, Vol.45, No. 3, Pages 367-373.

[14] R. Muntz and J. Lui, Performance Analysis of Disk Arrays under Failure. In Proceedings of the 16th Conference of Very Large Data Bases, Pages 162-173, 1990.

[15] Spencer W. Ng and Richard L. Mattson, Maintaining Good Performance in Disk Arrays During Failure via Uniform Parity Group Distribution, In Proceedings of the First International Symposium on High Performance Distributed Computing, Pages 260-269, IEEE Computer Society, 1992.

[16] D.A. Petterson, Garth A. Gibson, R.H. Katz, A case for Redundant Arrays of Inexpensive Disks (RAID). In Proceedings of the ACM SIGMOD Conference, pages 109-116, Association of Computing Machinery, 1988.

[17] A.L. Reddy and P.Banerjee, Gracefully Degradable Disk Arrays. In Digests of Papers of the 21st International Symposium on Fault-Tolerant Computing, Pages 401-408, The Computer Society, IEEE, June, 1991.

[18] J. Wilkes, R. Golding, C. Staelin, T. Sullivan, The HP AutoRaid Hierarchical Storage System, ACM Transactions on Computer Systems, Vol 14, No. 1, Feb 1996, pages 108-136.

[19] E.J. Schwabe and I. M. Sutherland, Improved Parity-Declustered Layouts for Disk Arrays, Journal of Computer and System Sciences, Vol. 53, No. 3, Academic Press, Dec, 1996, Pages 328-343.

[20] E.J. Schwabe and I. M. Sutherland, Flexible Usage of Parity Storage Space in Disk Arrays, Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures; Padua, Italy, June, 1996.