RAID Triple Parity

Atul Goel
NetApp Inc
atul.goel@netapp.com

Peter Corbett
NetApp Inc
peter.corbett@netapp.com

ABSTRACT

RAID triple parity (RTP) is a new algorithm for protecting against three-disk failures. It is an extension of the double failure correction Row-Diagonal Parity code. For any number of data disks, RTP uses only three parity disks. This is optimal with respect to the amount of redundant information required and accessed. RTP uses XOR operations and stores all data un-encoded. The algorithm's parity computation complexity is provably optimal. The decoding complexity is also much lower than that of existing comparable codes. This paper also describes a symmetric variant of the algorithm where parity computation is identical to triple reconstruction.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*

General Terms

Algorithms, Reliability, Theory

Keywords

Disk failure, RAID recovery, RDP code, recovery algorithm

1. INTRODUCTION

Enterprise customers expect very high availability, reliability, and performance guarantees from storage servers. As storage systems expand, it becomes increasingly important to protect against multiple simultaneous failure events. In the context of disks, failure events can typically be categorized as whole disk failures, partial failures, or intermittent failures. Traditional parity-based fault tolerance schemes like RAID 4 and RAID 5 [7] cannot protect against multiple failures. Mirroring-based schemes like RAID 1 or RAID 10 [7] are typically used to survive such failures. However, the substantially higher storage overhead makes mirroring unattractive from a cost perspective.

Various technology trends, for example, the use of cheap and unreliable ATA/SATA drives within data centers, further place new demands on software-based solutions to help compensate for less-reliable hardware. The high bit error rate (BER) on these drives, coupled with their large sizes, implies a significantly higher probability of encountering a partial failure, like a latent sector error, during reconstruction. While parity-based double failure protection schemes like RDP [3] and EVENODD [1] are currently able to address some of these issues, a triple erasure correction scheme can help further increase tolerance of the failure events, especially when an array is already degraded.

The primary motivation behind a triple fault tolerant scheme is the ability to survive shelf failures without exposing a RAID group to a potential data loss event due to media errors during reconstruction. Three simultaneous disk failures due to unrelated and independent reasons are probabilistically highly unlikely. However, such failures can occur due either to a loss of connectivity or a fault in a shelf. The technology trend towards bigger drives, combined with non-decreasing bit error rates, implies that the probability of data loss events due to media errors will increase proportionately. If two disks are lost due to a shelf failure, even double-parity RDP groups would be exposed to this failure during reconstruction. To prevent such scenarios, customers would have to configure RAID groups with only one disk per shelf. For example, in an 8-shelf system this would mean 6 data + 2 parity disks per RAID group, resulting in a data:parity ratio of 3:1. However, by striping two disks per shelf, a triple fault tolerant scheme can protect against similar failures by using 16-disk RAID groups (13 data + 3 parity). This results in much better storage utilization since it achieves a data:parity ratio of 4.3:1.

Another motivation for a triple fault tolerant scheme is that it provides a viable alternative to mirroring schemes such as RAID 1. As storage systems evolve, most hardware modules like adapters, loops, power supplies, shelf-controllers, heads, etc. can now be configured in a redundant mode to prevent a single point of failure. This implies that mirroring, which was originally intended to survive the failure of these components as well, is now relegated primarily to protecting against multiple simultaneous disk failures. Although mirroring is still essential for disaster recovery configurations, many customers use it within local data centers to help protect against multiple failures. As stated previously, shelf failures represent the most frequent cause of such

incidents. By striping disks across shelves, a triple failure protection scheme can be used to survive multiple failures with storage utilization significantly superior to mirroring. Hence, such a scheme helps lower the cost of storage while ensuring high levels of reliability and availability.

This paper describes a new algorithm, called RTP, for protecting against three-disk failures. The algorithm satisfies the Singleton bound [6] since it requires only three redundant disks to protect against three failures. Besides storage arrays, this algorithm can also be used to provide triple erasure correction capability to other applications relying on a redundant data stream. Data communication applications, where additional data is transmitted to reduce the reliance on retransmissions, can also use this algorithm to recover from a maximum of three erasures.

RTP can be efficiently incorporated within a log-structured file system. While other write in-place data layout schemes are forced to incur the parity-update overhead for small random writes, a log structured file system does a significantly better job owing to its write-anywhere nature and thereby provide a unique opportunity to implement advanced parity-based fault tolerance schemes which have a significantly higher storage efficiency compared to mirroring. In this paper we detail the construction of an RTP array, the parity computation scheme, and the reconstruction algorithm. We also present a variant of the algorithm, called Symmetric RTP, which can be used in distributed parity organizations supporting disk additions. Finally, we analyze the algorithm's performance and compare it against other known triple erasure correction schemes.

2. BASIC RTP ALGORITHM

RTP can be viewed as an extension of the double disk failure correction scheme, Row-Diagonal Parity (RDP) [3]. Recall that an RDP array requires a total of p+1 disks, where p is a prime > 2. The array is comprised of $p-1$ data disks, a row parity disk, and a diagonal parity disk. The array is divided into groups of $p-1$ stripes, each forming a self-contained row and diagonal parity set. Each stripe consists of a different block from each disk. Each block on the row parity disk stores the horizontal XOR sum of all data blocks within the same stripe. Blocks on the diagonal parity disk, on the other hand, store the XOR sum of $p-1$ diagonals which cover the row parity disk and all data disks. The diagonals are constructed over a group of $p-1$ stripes. Since there are p such diagonals, but space to store only $p-1$, the XOR sum of one of the diagonals is not computed. An important property of the diagonal formation used by RDP is that each diagonal misses a different disk (data or row parity) in the array.

An RTP array is constructed by adding a third parity disk, called anti-diagonal parity, to an RDP array. Thus, an RTP array consists of $p+2$ disks, where p is a prime greater than 2. In addition to the diagonals used by RDP, RTP defines a set of p anti-diagonals which have an orthogonal slope to the diagonals. These anti-diagonals cover the row parity disk and the data disks. Anti-diagonals do not cover the diagonal parity disk, and vice versa. Anti-diagonals are constructed over the same group of $p-1$ stripes as the diagonals. The anti-diagonal parity disk stores the XOR sum of $p-1$ anti-diagonals. As with diagonals, since there are a total of p anti-diagonals but only sufficient space to store $p-1$, the XOR sum of one of the anti-diagonals is not computed. The anti-diagonals are formed in such a way that they share the diagonal property of missing a different disk (data or row parity) in the array.

Figure 1: Diagonal Parity Sets

D0	D1	D2	D3	D4	D5	R	Diag	A-Diag
0	1	2	3	4	5	6	0	
1	2	3	4	5	6	0	1	
2	3	4	5	6	0	1	2	
3	4	5	6	0	1	2	3	
4	5	6	0	1	2	3	4	
5	6	0	1	2	3	4	5	

Figure 2: Anti-diagonal Parity Sets

D0	D1	D2	D3	D4	D5	R	Diag	A-Diag
6	0	1	2	3	4	5		6
5	6	0	1	2	3	4		5
4	5	6	0	1	2	3		4
3	4	5	6	0	1	2		3
2	3	4	5	6	0	1		2
1	2	3	4	5	6	0		1

Figure 1 and 2 show an example of the arrangement of data and parity within a set of stripes in a 9-disk RTP array constructed using prime p = 7. The array contains 6 data disks (D0, D1, D2, D3, D4 and D5), a row parity disk (R), a diagonal parity disk (Diag) and an anti-diagonal parity disk (A-Diag). Figure 1 and 2, both representing the same array, depict the assignment of blocks to diagonal and anti-diagonal parity sets respectively.

In Figure 1 , the number in each block indicates the diagonal parity set to which the block belongs. Similarly, Figure 2 shows the anti-diagonal parity set to which a block belongs.

Numbering disks/columns as $i = 0 \cdots p$ (where index of $D0 = 0, D1 = 1$, etc.), and rows as $j = 0 \cdots (p-2)$ (vertically down), lets define $A[i, j]$ as the block on disk i and row j. The diagonal parity set to which block $A[i, j]$ belongs can be computed as

$$D(i, j) = (i + j) \mod p \quad (1)$$

The anti-diagonal parity set corresponding to block $A[i, j]$ will be given by

$$AD(i, j) = (i - j - 1) \mod p \quad (2)$$

Note that i has a maximum value of p since the computation of diagonal parity sets excludes the anti-diagonal parity disk (and vice versa). Hence, the anti-diagonal and diagonal parity disks don't have to be numbered when a parity set of the other type is computed. Blocks on the row parity disk are computed as the horizontal even parity sum of data blocks within the same stripe. Diagonal parity is computed in a manner identical to that used for RDP [4]. The anti-diagonal parity is computed in a manner similar to that for diagonal parity, however, using anti-diagonal blocks. Each anti-diagonal parity block contains the even parity of the data and row parity blocks on the same anti-diagonal. Equation 3 defines the value for a diagonal parity block at row/index x ($x = 0 \cdots (p-2)$), computed as the XOR sum of blocks on disks $i = 0 \cdots (p-1)$ (i.e $D0 \cdots R$). Similarly,

for figure 2 above, equation 4 defines the value of an anti-diagonal parity block at row/index x ($x = 0 \cdots (p-2)$). Note that the anti-diagonal parity block at row 0 corresponds to anti-diagonal 6. In addition, for any i, $A[i, p-1]$ is assumed to be 0.

$$Diag[x] = \sum_{i=0}^{i=p-1} A[i, (x-i) \mod p] \quad (3)$$

$$Anti - Diag[x] = \sum_{i=0}^{i=p-1} A[i, (x+i) \mod p] \quad (4)$$

In figures 1and 2 diagonal 6 and anti-diagonal 0 are dropped from the respective parity disks since there is only enough space for storing $6(= p-1)$ diagonals/anti-diagonals.

An astute observer might note that an RTP array can be viewed as composed of two sets of double fault tolerant RDP arrays that share the same data and row parity disks. This is because the set of data disks, row parity, and anti-diagonal parity disks itself constitutes an RDP array which uses diagonals with orthogonal slopes.

Although the above formulation requires $p-1$ data disks, the same scheme can also be used with fewer data disks by assuming fake zeroed disks to complete a prime parity set.

3. RECONSTRUCTION

This section discusses reconstruction from single, double and triple disk failures.

3.1 Single and Double Disk Reconstruction

Recovery from single disk failures can be accomplished either by using row parity or by computing the diagonal or anti-diagonal parity disk. Since RTP extends RDP, double disk reconstruction can be performed by using the RDP reconstruction algorithm. This is because the two failed disks belong to at least one of the diagonal or anti-diagonal RDP parity sets. If both diagonal and anti-diagonal parity disks fail, they can be independently reconstructed from the data and horizontal parity drives, using the RTP parity construction algorithm.

3.2 Triple Disk Reconstruction

Triple disk failure cases in an RTP array can be classified into three categories, depending upon the number of data and parity disks failed. For simplicity and ease of understanding, the row parity disk and data disks are collectively referred to as RAID 4 disks, since they are symmetric with respect to the diagonal and anti-diagonal parity sets.

- One of RAID 4, diagonal and anti-diagonal disk failed: This case is trivial since the missing RAID 4 disk can be recovered using row parity. The RTP parity computation algorithm can then be applied to recover the missing diagonal and anti-diagonal parity disks.
- Two RAID 4 and one diagonal (or anti-diagonal) disks failed: For this case, reconstruction proceeds by first applying the RDP double reconstruction algorithm [3] using the good anti-diagonal (or diagonal) parity disk. After the failed RAID 4 disks are recovered, the missing diagonal (or anti-diagonal) can be recovered using the RTP parity computation algorithm.

Figure 3: Notations

$A[..]$	Data blocks
$R[..]$	Stored row parity blocks
$Diag[..]$	Stored diagonal parity blocks
$Anti–Diag[..]$	Stored anti-diagonal parity blocks
$A_{dropped\,diag}$	Data blocks on the dropped diagonal
$R_{dropped\,diag}$	Row parity blocks on the dropped diagonal
$AR[..]$	RAID 4 i.e both data and row parity blocks
$AR_{dropped\,diag}$	RAID 4 blocks on the dropped diagonal
$AR_{dropped\,anti-diag}$	RAID 4 blocks on the dropped anti-diagonal

- Three RAID 4 disks failed: The primary step in the process of reconstructing three RAID 4 disks involves computing p 4-tuple XOR sums on one of the missing disks. Each 4-tuple sum (equation 7, section 3.2.2) is computed as the XOR sum of 4 blocks on one of the failed disks. The set of linear equations corresponding to these sums can then be solved in various ways to recover that missing disk. Subsequently, the remaining two disks can then be recovered using the RDP double reconstruction algorithm.

For this process to work, parity for all diagonal and anti-diagonals must be available. Hence, the triple disk reconstruction steps can be broadly classified as:

1. Compute the dropped parity block for both the diagonal and anti-diagonal parity sets.
2. Compute a set of 4-tuple XOR sums on one of the failed disks
3. Recover one of the failed disks.
4. Use RDP double reconstruction to recover the remaining two disks

The remainder of this section details each of the steps when reconstructing three RAID 4 disks. Figure 3 describes the notations used in this section.

3.2.1 Compute Diagonal And Anti-Diagonal Parity

In a set of $p-1$ stripes forming a complete row, diagonal and anti-diagonal parity set, the parity for the missing/dropped diagonal in an RTP (or even RDP) array can be computed as the XOR sum of the $p-1$ blocks on the diagonal parity disk. Similarly, the missing anti-diagonal can be computed as the XOR sum of blocks on the anti-diagonal disk.

PROOF. Since each diagonal spans both data and row parity disks, and one diagonal is dropped owing to insufficient space, the XOR sum of the blocks stored on the diagonal parity disk can be computed as:

$$\sum Diag[..] = \Big(\sum A[..] + \sum A_{dropped\,diagonal} + \sum R[..] + \sum R_{dropped\,diagonal} \Big)$$

Substituting $\sum R[..]$ by $\sum A[..]$ since blocks on the row parity disk are themselves the XOR sums of data blocks, we get

X=0	Y=1	d2	d3	Z=4	d5	R
0	1Φ	2	3	4	5	6
1	2Φ	3	4	5	6	0
2	3	4	5	6	0	1
3	4Φ	5	6	0	1	2
4	5Φ	6	0	1	2	3
5	6	0	1	2	3	4
6	0	1	2	3	4	5

Figure 4: 4-tuple sum (row 0): Steps 1 and 2

X=0	Y=1	d2	d3	Z=4	d5	R
6	0Φ	1	2	3	4	5
5	6Φ	0	1	2	3	4
4	5	6	0	1	2	3
3	4Φ	5	6	0	1	2
2	3Φ	4	5	6	0	1
1	2	3	4	5	6	0
0	1	2	3	4	5	6

Figure 5: 4-tuple sum (row 0): Steps 3 and 4

$$\sum Diag[..] = \sum A_{dropped\,diagonal} + \sum R_{dropped\,diagonal} = \sum AR_{dropped\,diag} \quad (5)$$

Another way to think of this is that the sum of all the RAID 4 blocks equals the sum of all the diagonal parity blocks including the dropped diagonal parity block. Therefore, subtracting the sum of the stored diagonal parity blocks from the sum of all RAID 4 blocks gives the sum of the RAID 4 blocks on the dropped diagonal. Similarly,

$$\sum Anti\text{–}diag[..] = \sum AR_{dropped\,anti-diag} \quad (6)$$

After computing the dropped diagonal and anti-diagonal parity, parity for all diagonals and anti-diagonals is available. For each row, diagonal and anti-diagonal, the XOR sum of missing blocks on the three failed disks can now be computed by summing the surviving blocks on the corresponding row, diagonal and anti-diagonal respectively. Let $XOR_r(k)$, $XOR_d(d)$, and $XOR_a(a)$ represent these values corresponding to row k, diagonal d, and anti-diagonal a respectively. Since with even parity the sum of all blocks in one parity stripe or diagonal is zero, then the sum of any subset of items in the set must equal the sum of the remaining items not in the subset. The sum of the surviving items in a parity set is often called a "parity syndrome" in the literature.

□

3.2.2 Compute 4-tuple sums on one of the failed disks

The remaining sets of steps are described in the context of the 9-disk RTP array shown in figure 1 and 2. Figures 4 and 5 represent an equivalent diagonal and anti-diagonal layout within the 7-disk RAID 4 array (the diagonal and anti-diagonal parity disks are not shown). Although each group of $p-1$ stripes consists of only six rows, the seventh row (the shaded row at the bottom of the table) is shown to simplify understanding of the reconstruction process. Blocks on the seventh row, which are assumed to be zero, are assigned to the appropriate diagonals and anti-diagonals using Equations 1 and 2 respectively. Disks X=0, Y=1 and Z=4, represent the 3 failed drives within the array.

Assuming an ordering of failed disks as X, Y and Z, let's define the distance between them as

$$g = (Y - X)$$

and

$$h = (Z - Y)$$

For the above ordering, lets refer to disk Y as the middle disk.

We define a 4-tuple sum as the XOR sum of 4 blocks on the middle disk, Y. Within a group of $p-1$ stripes, forming a self-contained parity set, a 4-tuple XOR sum corresponding to row k on the middle disk Y is computed in the following manner:

1. Retrieve the row parity sum of blocks on missing disks corresponding to row k. Let $XOR_r(k)$ denote this sum. The row parity sum represents the XOR sum of blocks $A[X,k], A[Y,k], A[Z,k]$

2. Compute the diagonal d for the block on disk Z and row k, $A[Z,k]$, using Equation 1 as $d(Z,k) = (Z+k) \mod p$. Retrieve the diagonal parity sum of blocks on missing disks corresponding to this diagonal. Let $XOR_d(d(Z,k))$ denote this sum. The row at which diagonal d intersects disk X can be computed as $q = (k + Z - X) \mod p$.

   $XOR_d(d(Z,k))$ represents the XOR sum of 3 blocks, one on each of the missing disks. These blocks are $A[Z,k], A[Y,(k+Z-Y) \mod p], A[X,(k+Z-X) \mod p]$.

3. Compute the anti-diagonal, a, for the block on disk X and row k, $A[X,k]$. Using Equation 2 $a(X,k) = (X-k-1) \mod p$. Retrieve the anti-diagonal parity sum of blocks on missing disks corresponding to this anti-diagonal. Let $XOR_a(a(X,k))$ denote this sum. The anti-diagonal a will also intersect disk Z at row q due to the fact that diagonal and anti-diagonal constructions use orthogonal slopes.

   $XOR_a(a(X,k))$ represents the XOR sum of 3 blocks, one on each of the missing disks. These blocks are $A[X,k]$, $A[Y,(k+Y-X) \mod p]$, $A[Z,(k+Z-X) \mod p]$.

4. Retrieve the row parity sum of blocks on missing disks corresponding to row q, where q is the same as that computed in step 2.

   Let $XOR_r(q)$ denote this sum. $XOR_r(q)$ represents the XOR sum of blocks $A[X,(k+Z-X) \mod p]$, $A[Y,(k+Z-X) \mod p]$, $A[Z,(k+Z-X) \mod p]$

5. Compute the 4-tuple XOR sum for row k as

$$XOR_r(k) + XOR_d(d(Z,k)) + XOR_a(a(X,k)) + XOR_r(q) \quad (7)$$

A total of p such 4-tuple sums are computed, one for each row. Since the array only contains $p-1$ rows, the p^{th} 4-tuple sum is formed by assuming an imaginary p^{th} row of zeroed

blocks and by using the dropped diagonal and anti-diagonals on disks Z and X respectively.

Since the XOR of a term with itself is 0, summing the blocks in steps 1 to 4 results in the cancellation of four pairs of duplicate terms in rows k and q of disks X and Z. For example, $XOR_r(k)$ and $XOR_d(d(Z,k))$ both include the block $A[Z,k]$. Similarly, $XOR_r(k)$ and $XOR_a(a(X,k))$ include the common block $A[X,k]$.

The 4-tuple sum represents the XOR sum of at most 4 blocks on the middle disk, Y. This set of blocks can be represented as:

$\{A[Y,k], A[Y,(k+Z-Y) \mod p], A[Y,(k+Y-X) \mod p], A[Y,(k+Z-X) \mod p]\}$

Substituting $g = Y - X$ and $h = Z - Y$, the above set of blocks can be re-written as

$\{A[Y,k], A[Y,(k+h) \mod p], A[Y,(k+g) \mod p], A[Y,(k+g+h) \mod p]\}$

Dropping the common disk index, and assuming modulo p addition, the 4-tuple corresponding to stripe k can be represented as

$$T(k) = \{k, k+h, k+g, k+g+h\} \quad (8)$$

Figure 4 and 5 illustrate an example of computing the 4-tuple sum corresponding to row 0. Figure 4 depicts steps 1 and 2 involving the XOR sums of missing blocks in row 0 (thick-bordered cells) and diagonal 4 (lightly shaded cells). Figure 5 illustrates steps 3 and 4 involving the sums of blocks on anti-diagonal 6 (lightly shaded cells) and row 4 (thick-bordered cells). Blocks $A[0,0], A[4,0], A[0,4]$ and $A[4,4]$ are cancelled since they are included twice. This results in the 4-tuple sum of blocks $A[1,0], A[1,1], A[1,3]$ and $A[1,4]$ on disk Y. These blocks are shown in Figures 3 and 4 via the symbol Φ. For computing the p^{th} 4-tuple sum, the above process is repeated for the seventh row (shaded).

For the disk array shown above, the set of all 4-tuples for which sums are computed is $[0,1,3,4], [1,2,4,5], [2,3,5,6], [3,4,6,0], [4,5,0,1], [5,6,1,2]$, and $[6,0,2,3]$ (the column index representing disk Y is dropped).

Alternate ordering of failed disks.

The step for computing 4-tuple sums can be performed by assuming an arbitrary ordering of failed disks. In the above example, the order chosen is $X = 0, Y = 1$ and $Z = 4$. Instead, a different ordering $X = 0, Y = 4$, and $Z = 1$ could have been chosen. In this case, the first disk to be reconstructed, the middle disk, would be $Y = 4$, and the values for g and h are $Y - X = 4$ and $Z - Y = -3$ respectively. For three-disk failures, there are a total of six possible orderings. Since each ordering results in a different set of values for g and h, the set of 4-tuple sums generated is also different. As a result, the number of these sums which must be combined to recover blocks on the middle disk differs for different orderings. Hence, the best computational efficiency is achieved by choosing an ordering which minimizes the number of $XORs$ required.

The mechanism used for picking the most efficient ordering depends on the methodology used for solving the equations represented by the collection of 4-tuple sums. As described in section 3.2.3, if the approach involves reducing to pairwise sums, then the number (m) of 4-tuple sums which must be XOR-ed to calculate each pairwise sum can be computed as a function of g, h, and p (details in section 3.2.3). Hence, the most efficient ordering is the one which yields the minimum value for m.

Equidistant failures.

The distance between failed disks also plays a crucial role when computing 4-tuple sums. For example, if $g == h$, two additional blocks on the middle disk, represented by $(k+h) \mod p$ can be cancelled. This converts the 4-tuple sum to a 2-tuple sum. Since pairwise sums are obtained, the process for recovering the middle disk can be greatly simplified. Representing the 4-tuple sum corresponding to row 0 as $[0, g, h, h+g]$, one can generalize scenarios where pairwise sums are obtained to satisfy the condition $g = h \mod p$ or $(g-h) \mod p = 0$. Triple disk failures where an ordering of failed disks satisfies this condition can be categorized as equidistant failures. The condition $(g-h) \mod p = 0$ implies that the second and the third blocks within the 4-tuple are identical and hence can be cancelled. No other pairing of blocks within the 4-tuple sum can be identical.

3.2.3 *Recover one of the failed disks*

The collection of 4-tuple sums represents a set of p linear equations over p variables. Each variable represents a block on the middle disk and contributes to exactly four equations. One of these variables, the block on the imaginary p^{th} row, is known to have the value zero. This set of linear equations can be solved in various ways to recover the middle disk.

To illustrate the process of recovering the middle disk, this section describes one approach where a subset of 4-tuples is selected and reduced to a pairwise sum. Repeating this process $p-1$ times by starting with a different tuple helps generate $p-1$ pairwise sums which are separated by a constant distance. At least one such pair, however, has only one unknown block and hence can be recovered. The set of other pairwise sums can then be used to recover the remaining blocks on the middle disk. This approach is similar to that used by [5] where a ring of crosses is reduced to a pairwise sum. However, the tuple approach used by RTP requires fewer XOR operations.

Other, more-efficient approaches are also possible for recovering the middle disk.

Reduce to pairwise sums.

One approach for selecting a subset of tuples is to start with a 4-tuple corresponding to row k and to choose subsequent tuples at an offset g (or h). At each step, common blocks are cancelled and the process continues until only two unknown blocks are left. This results in a pairwise sum.

For example, starting with the 4-tuple sum corresponding to row $0, T(0) = [0, g, h, g+h]$, choosing another tuple at an offset g helps cancel two blocks while adding two new blocks, thus keeping the total number of unknown blocks the same. This is because the tuple corresponding to row g is $T(g) = [g, 2g, h+g, 2g+h]$. XOR-ing this tuple to $T(0)$ cancels common blocks g and $h+g$ since they are present in both tuples. (All additions and multiplications are assumed to be modulo p). Hence, starting with $T(0)$ (lets treat this as the first step), and selecting consecutive tuples at an offset g, step m results in the XOR sum of blocks $[0, m*g, h, m*g+h]$.

Now we use the property that if p is a prime and $g, h < p$, one can always find an $m (0 < m < p)$ such that $((m*g+h) \mod p) = 0$ is true. Similarly, one can always find an m such that $((m*g-h) \mod p) = 0$ is true. Hence, by choosing

an m such that $((m * g + h) \mod p == 0)$, the first and the fourth blocks in the result $[0, m * g, h, m * g + h]$ can be cancelled after the m^{th} step. Alternately, by choosing an m such that $((m * g - h) \mod p) == 0$, the second and the third blocks can be cancelled after the m^{th} step. Since only two unknown blocks are left, the process of selecting tuples can be terminated at this step. Therefore, starting with a tuple $T(k)$ results in the pairwise sum of $[k, m * g + h + k]$.

Repeating the above step by starting with 4-tuple sums at each of the $p-1$ rows results in $p-1$ pairwise sums where each is separated by the same distance, $(m * g + h) \mod p$.

For the disk failure scenario described in section 3.2.2, g and h can be computed as $g = 1$, and $h = 3$. Hence, choosing tuples at offset 1 (= g), a pairwise sum of two unknown blocks can be computed by using either $m = 3$ tuples (since $((3 * 1 - 3) \mod 7 == 0)$) or m=4 tuples (since $((4 * 1 + 3) \mod 7 == 0)$). Since the array construction uses only $p - 1$ rows, block $p - 1$ (i.e., p^{th} block) on disk Y is assumed as zero. Thus, the value of the block which is pairwise summed with the p^{th} block can automatically be computed. Using other pairwise sums, the remaining blocks on disk Y can then be recovered.

3.2.4 *Use RDP double reconstruction to recover remaining two disks*

Having recovered the middle disk, RDP double reconstruction can be used to recover the remaining two disks, thus completing the process of reconstructing three RAID 4 disks.

3.3 Reconstruction Using Matrix Methods

The reconstruction of an array can also be performed using matrix methods, as described by Hafner et al. [4]. The algorithm described in that paper will not be repeated here, but essentially, it provides a technique for recovering from arbitrary combinations of data loss with a deterministic matrix method. We improved significantly on the algorithm in testing the ability of RTP to correct against arbitrary combinations of three disk failures. The essence of the improvement was to reduce the memory consumption tremendously, by a factor of 10,000, by reducing the set of surviving symbols to their equivalent syndromes (a syndrome is defined as the XOR sum of surviving symbols along a given direction, row, diagonal, or anti-diagonal).

It may be simpler to use the matrix method of code reconstruction, especially after failures of three RAID 4 disks, than to use the recursive reconstruction method described in the previous section. Using the matrix method will completely recover all the disks using the minimum number of symbols to recover each missing symbol directly. Thus there is no need to recover the middle failed disk independently, because it and the other two failed disks can be recovered directly from the surviving syndromes in parallel.

4. SYMMETRIC RTP

This section describes a symmetric variant of RTP. Symmetric RTP uses the algorithm for computing parity identical to that used for triple reconstruction.

The mechanism used for converting asymmetric RTP to its symmetric variant is generic enough and can be used for converting other asymmetric horizontal codes to their symmetric equivalents as well.

Symmetric RTP shares the same advantages as any other symmetric code (e.g., RAID 5). It enables a dynamic distributed parity organization which supports parity relocation during disk additions. Asymmetric RTP, on the other hand, cannot be used in a similar RAID array. This is because diagonal and anti-diagonal parity blocks cannot be relocated to newly added disks since they do not participate in row parity computation.

D0	D1	D2	D3	X(P1)	Y(P2)	R(P3)
0	1	2	3	4	5	6
1	2	3	4	5	6	0
2	3	4	5	6	0	1
3	4	5	6	0	1	2
4	5	6	0	1	2	3
5	6	0	1	2	3	4

Diag	A-diag
0	0
0	0
0	0
0	0
0	0
0	0

Figure 6: A symmetric RTP array

A distributed parity organization not only allows all spindles to be used for user I/Os, but also facilitates efficient small writes. Since parity relocation can be seamlessly achieved, these benefits can be incorporated within a disk-topology-aware file system,without losing the ability to add disks to a RAID group.

4.1 Array Construction and Parity Computation

A symmetric RTP array can be viewed as identical to its asymmetric variant where the diagonal and anti-diagonal parity disks are always zero. Parity computation within a symmetric RTP array can be best understood as a triple reconstruction process within an equivalent asymmetric RTP array.

Lets assume that we have an asymmetric RTP array with the additional constraint that data can only be written to $p-3$ disks. This implies that there are two fewer disks available for data. Figure 6 shows the construction of an RTP array (formed using prime $p = 7$) with this restriction. Although the figure only shows the diagonal parity arrangement and skips anti-diagonal formations, equivalent assertions apply for both diagonal types. The data disks which cannot be written to are marked X(P1) and Y(P2). Let R(P3) be the row parity for the array.

When writing a stripe (covering only $p - 3$ disks), instead of computing the diagonal and anti-diagonal parity lets rather assume them to be zero. The diagonal and anti-diagonal parity disks are shown as two separate columns below the data disks. In practice, they are not physically present as their contents are always zero. Using the asymmetric RTP triple reconstruction algorithm, compute blocks on disks X, Y, and R. Triple reconstruction in this case follows the algorithm used for recovering three RAID 4 disks. Since we know that asymmetric RTP can correctly recover from triple erasures, it implies that for a given combination of values on disks D0, D1, D2, D3, Diag and A-Diag parity, there exists a unique set of values for disks X, Y, and R such that the array satisfies the parity invariant along the three dimensions, row, diagonal, and anti-diagonal. For each of the p diagonals and anti-diagonals, an array constructed using this method satisfies the following properties:

- Diagonal parity sum is zero including the missing diagonal, since all the blocks on the diagonal parity disk are zeroes.
- Anti-diagonal parity sum is zero including the missing

anti-diagonal, since all the blocks on the anti-diagonal are zeroes.

By updating X, Y, and R in this manner, we can guarantee that the diagonal and anti-diagonal disks are always zero. Hence, the diagonal and anti-diagonal parity disks can be dropped from the array, and instead disks X and Y can be treated as two parity disks. Together with the row parity disk, R, the disks X, Y and R now form the three parity disks (renamed $P1, P2$, and $P3$) for the RAID group.

Full stripe writes (spanning $p-3$ data disks) and parity computation by recalculation can be performed using the method just described. Parity computation by subtraction, on the other hand, first requires computing the delta for the modified blocks. Starting with the data delta and ignoring data disks not being written to, the parity computation algorithm (same as triple reconstruction) can now be applied to compute the parity delta. Contents of the parity disks can then be updated using this delta to compute the new values.

5. ANALYSIS

In this section we analyze the performance of RTP, from the perspective of both I/O as well as computation complexity measured in terms of the number of XORs required. We also compare RTP with another triple erasure correcting code called STAR [5]. Since RTP stores data unencoded, there is no penalty on read operations. The amount of I/O required for both parity computation as well as triple reconstruction is optimal. However, since a complete row, diagonal and anti-diagonal parity set is formed over $p-1$ stripes, the number of I/Os required for full stripe writes by the file system might be sub-optimal. As stated in the implementation considerations section this can, however, be handled by dividing a single block into $p-1$ sub-chunks and forming a selfcontained parity set within a single stripe of such blocks.

5.1 Parity computation complexity

For an RTP array consisting of $p-1$ rows and $p-1$ data disks (i.e., $p+2$ disks including parity), the number of XORs required for writing a set of stripes constituting a self-contained parity set can be computed as:

$$\begin{aligned} &number\ of\ XORs\ required\ for\ row\ parity \\ &+diagonal\ parity + anti\text{-}diagonal\ parity \\ &= (p-1)(p-2) + (p-1)(p-2) + (p-1)(p-2) \\ &= 3p^2 - 9p + 6 \end{aligned} \quad (9)$$

One can prove that for an array of n data disks, the minimum number of per-block XORs required to protect against three failures is $3-3/n$. The proof follows from observations similar to those used for proving RDP optimality [3]. The singleton bound requires a minimum of three parity blocks per row of, say, n data blocks. Each data block must contribute to at least three different parity blocks, one on each parity disk to ensure that recovery is possible if the data block and two parity disks are lost. In the minimal formulation, parity blocks are computed using equations that contain no common pairs of data blocks. This is because in the absence of one of the parity disks it must still be possible to recover from two additional failures. Hence, equations corresponding to the surviving parity disks cannot contain a common pair of data blocks. Assuming r rows, the minimum number of separately XOR-ed input terms required to construct $3r$ parity blocks is thus $3nr$. A set of $3r$ equations that reduces $3nr$ terms to $3r$ using XORs requires $3nr-3r$ XORs. Therefore, the minimum number of per-block XORs required for triple protection is

$$\big(3nr - 3r\big)/nr = 3 - 3/n \quad (10)$$

Substituting $p-1=n$ in Equation 9 (since there are $p-1$ data blocks in each stripe), the number of XORs required to protect n^2 blocks can be computed as $3n^2-3n$, which is the minimum based on Equation 10. Thus, parity computation complexity for RTP is optimal.

5.2 Reconstruction complexity

Since reconstruction complexity for double-parity RDP is optimal, the same is true for single and double disk reconstructions in RTP. In addition, the computation complexity for triple disk reconstruction is also optimal for all combinations of failures except those involving three RAID 4 disks. For failure scenarios involving three RAID 4 disks, the computation complexity depends upon the distance between failed disks. This section analyzes the number of XORs required for this case. Computing dropped diagonal and anti-diagonal parity (section 3.2.1) requires XOR-ing $p-1$ blocks on both diagonal as well as anti-diagonal parity disks, for a total of $2(p-2)$ XORs. For an RTP array using k data disks, row sums of missing blocks require $(k-3)(p-1)$ XORs. Diagonal and anti-diagonal sums of missing blocks, on the other hand, require an extra $(p-1)$ XORs each since the diagonal and anti-diagonal parity blocks must also be included. Hence, the total computation complexity of this step is equivalent to the computation complexity for computing dropped diagonal D' and anti-diagonal parity A' as

$$2(p-2) + 3(k-3)(p-1) + 2(p-1) \quad (11)$$

Each 4-tuple sum (section 3.2.2) requires 3 XORs. Since p such tuples are created, the complexity of this step is equivalent to the computation complexity for creating the 4-tuple, which is $3p$.

$$\begin{aligned} &Computation\ complexity\ for \\ &creating\ the\ 4-tuple\ sums = 3p \end{aligned} \quad (12)$$

If recovery of the middle disk is achieved by using the approach for reducing tuples to pairwise sums, then $(m-1)$ XORs are required for computing each pairwise sum (as defined in section 3.2.3, m is the number of 4-tuple sums which must be combined to reduce to a pairwise sum). Since $(p-1)$ such pairs are computed, the complexity of this step is equivalent to the complexity of reducing tuples to pair-wise sums, computed as $(m-1)(p-1)$.

$$\begin{aligned} &The\ complexity\ of\ reducing\ tuples \\ &to\ pairwise\ sums = (m-1)(p-1) \end{aligned} \quad (13)$$

Starting with pairwise sums, the number of XORs required to recover all blocks on the middle disk is $(p-2)$.

$$\begin{aligned} &The\ number\ of\ XORs\ required\ to\ fully \\ &recover\ middle\ disk = (p-2) \end{aligned} \quad (14)$$

To recover the remaining two disks (section 3.2.4) using RDP double reconstruction [3], blocks on the middle disk must be added to the row and diagonal sums of missing blocks. Alternately, double reconstruction could be performed using anti-diagonals instead of diagonals. Although there are p diagonals/anti-diagonals, double reconstruction doesn't require those which are dropped. Subsequently, recovering blocks on the remaining two disks requires $2(p-1)-1$ XORs. Hence, the complexity of this step is

$$2(p-1)+2(p-1)-1 \tag{15}$$

Summing equations 11 through 15, we can calculate the total triple reconstruction complexity for an RTP array with k disks as

$$\begin{aligned} &2(p-2)+3(k-3)(p-1)+2(p-1)+3p+ \\ &(m-1)(p-1)+(p-2)+2(p-1)+2(p-1)-1 \\ &\qquad = (3k+m+2)(p-1)-1 \end{aligned} \tag{16}$$

The reconstruction complexity in an RTP array depends on m. The minimum complexity case arises for equidistant failures where $m = 1$. Since $1 \leq m \leq (p-1)$, the reconstruction complexity could be high for higher values of m. Hence, an ordering of failed disks is chosen in a manner which results in the smallest m.

5.3 Comparison with STAR

STAR [5] is a horizontal triple erasure correcting code which is derived from EVENODD. STAR extends EVENODD by adding a new parity disk which is used for storing orthogonal diagonals. This is similar to RTP. However, since an RTP array is constructed by extending RDP, it is computationally more efficient compared to STAR. For n data disks and $n(n-1)$ data blocks, parity computation in STAR requires a total of $(n-1)(n-1)+2n(n-2)+2(n-1) = 3n^2-4n-1$ XORs. Hence, the number of XORs required per block $= (3n^2-4n-1)/(n^2-n) = 3-1/(n-1)$ which is greater than that required for RTP. The difference is more pronounced on smaller disk array sizes, which are typical of most storage systems.

For triple reconstructions, the pairwise sum approach used by RTP is similar to that used by STAR. However, the set of linear equations described by the collection of 4-tuple sums can be solved by other, perhaps more efficient, methods as well.

The triple reconstruction complexity of STAR is:

$$(3k+2m+n)(p-1) \tag{17}$$

In Equation 17, n refers to the number of rows in a ring which are cancelled through substitution. Since each new cross substitutes symbols in at most two rows, the inequality $m \leq n \leq 2m$ must hold true. The value of m (number of crosses in a ring) depends upon the distance between failed disks and the prime p used for constructing the array. For the purpose of this comparison, m used in Equation 17 can be treated as identical to that used for RTP in Equation 16. In general, m satisfies the inequality $1 \leq m \leq (p-1)$. Hence, for a given p, the worst-case reconstruction complexity for STAR can be much higher than that for RTP. For example, in a 12 data disk array constructed using $p = 31, m = 5$ implies that STAR would require approximately 23% more XORs than RTP.

While the average value of m is much lower than p, the same is not necessarily true for its relation with k, the number of data disks. As a result, for large primes, the term $(2m+n)(p-1)$ can be a significant contributor to the reconstruction complexity for STAR [5]. The RTP triple reconstruction algorithm would perform significantly better than STAR in such cases.

6. IMPLEMENTATION CONSIDERATIONS

To ensure efficient full stripe writes, where parity computation can be done without reading any blocks, a complete parity set must be contained within a single system block size. Since block sizes are typically chosen as a power of 2, selecting a prime of the form $2^n + 1$ allows formation of row, diagonal, and anti-diagonal parity sets within a single system stripe by dividing a block into n sub-blocks.

Since parity computation in Symmetric RTP is based on RTP's triple reconstruction algorithm, it is computationally less efficient than the asymmetric version. Besides, parity computation complexity is variable depending on the distance between the parity disks. The minimum complexity case occurs when the distance between the parity disks is the same. When creating a new Symmetric RTP array, parity placement can be chosen keeping the above consideration in mind. As parity gets relocated to newly added disks, it is possible for the computational complexity to increase. Since the worst-case complexity is high, it might not be possible to always achieve a good parity balance across newly added disks. However, single disk additions are rare. By imposing restrictions on the minimum number of disks that can be added to a RAID group, it should be possible to relocate parity blocks while still ensuring efficient parity computations. This should help keep bounds on the computational complexity to a predetermined maximum.

7. RELATED WORK

Other algorithms that provide protection against three or more drive failures include STAR [5] , Reed Solomon (P+Q) erasure codes [5], MDS array codes [2], and Datum. Like RTP, STAR and Reed Solomon codes are horizontal codes where redundant information is stored separately from data blocks. This ensures flexible data layout since these algorithms do not place any constraint on mixing parity information with data

Both Reed Solomon and STAR codes compute row parity in a manner similar to that for RAID 4 or RAID 5. However, the algorithm used for computing the second and third parity information is different. Although Reed Solomon codes can tolerate higher orders of failures, the computation complexity is significantly higher since these codes use finite field operations intensively.

RTP most closely resembles STAR. This is primarily because RTP extends from RDP [3], which is similar to EVENODD [1]. STAR, in turn, is derived from EVENODD. Unlike STAR, the encoding complexity for RTP is optimal. Of the existing algorithms, STAR and MDS codes have the lowest decoding complexity. However, the complexity of MDS codes is high for RAID arrays using a small number of disks. RTP improves upon STAR and MDS codes by further reducing the decoding complexity.

8. CONCLUSION

RTP is a triple disk failure protection algorithm that is provably optimal with respect to parity computation. This is important since in a normal, failure-free operation mode, the parity computation overhead is incurred for every write. The decoding complexity for RTP is also much lower than for other triple fault-tolerant erasure codes.

For parity-based RAID arrays, the ability to tolerate three failures ensures extremely high availability of the array as a whole. It permits the use of cheap, unreliable disks (e.g., ATA, SATA) while still ensuring high levels of reliability. It also provides a viable alternative to mirroring schemes like RAID 1 since the worst- case fault tolerance levels are the same. RTP can be efficiently incorporated within a log-structured file system, which does not suffer from the small-write parity update overhead problem.

Future work on RTP involves further reducing the decoding complexity and adapting it to handle higherorder failures.

9. REFERENCES

[1] M. Blaum, J. Brady, J. Bruck, and J. Menon. Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *In Proc. of the Annual International Symposium on Computer Architecture.*, pages 245–254, 1994.

[2] M. Blaum, J. Bruck, and A. Vardy. Mds array codes with independent parity symbols. *IEEE Trans. Information Theory.*, 42(2):529–542, March 1996.

[3] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. *Proc. of USENIX FAST*, March-April 2004.

[4] J. L. Hafner, V. Deenadhayalan, and K. K. Rao. Matrix methods for lost data reconstruction in erasure codes. *Proc. of USENIX FAST, San Francisco, CA*, December 2005.

[5] C. Huang and L. Xu. Star : An efficient coding scheme for correcting triple storage node failures. *Proc. of USENIX FAST*, December 2005.

[6] F. J. MacWilliams and J. J. A. Sloane. The theory of error-correcting codes. 1977.

[7] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (raid). *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 109–116, 1988.