USENIX Association

Proceedings of the Third USENIX Conference on File and Storage Technologies

San Francisco, CA, USA
March 31–April 2, 2004

Row-Diagonal Parity for Double Disk Failure Correction

Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac,
Steven Kleiman, James Leong, and Sunitha Sankar
Network Appliance, Inc.

Abstract

Row-Diagonal Parity (RDP) is a new algorithm for protecting against double disk failures. It stores all data unencoded, and uses only exclusive-or operations to compute parity. RDP is provably optimal in computational complexity, both during construction and reconstruction. Like other algorithms, it is optimal in the amount of redundant information stored and accessed. RDP works within a single stripe of blocks of sizes normally used by file systems, databases and disk arrays. It can be utilized in a fixed (RAID-4) or rotated (RAID-5) parity placement style. It is possible to extend the algorithm to encompass multiple RAID-4 or RAID-5 disk arrays in a single RDP disk array. It is possible to add disks to an existing RDP array without recalculating parity or moving data. Implementation results show that RDP performance can be made nearly equal to single parity RAID-4 and RAID-5 performance.

1 Introduction

Disk striping techniques [1, 2] have been used for more than two decades to reduce data loss due to disk failure, while improving performance. The commonly used RAID techniques, RAID-4 and RAID-5, protect against only a single disk failure. Among the standard RAID techniques, only mirrored stripes (RAID-10, RAID-01) provide protection against multiple failures. However, they do not protect against double disk failures of opposing disks in the mirror. Mirrored RAID-4 and RAID-5 protect against higher order failures [4]. However, the efficiency of the array as measured by its data capacity divided by its total disk space is reduced. Increasing the redundancy by small increments per stripe is more cost effective than adding redundancy by replicating the entire array [3].

The dramatic increase in disk sizes, the relatively slower growth in disk bandwidth, the construction of disk arrays containing larger numbers of disks, and the use of less reliable and less performant varieties of disk such as ATA combines to increase the rate of double disk failures, as will be discussed in Section 3. This requires the use of algorithms that can protect against double disk failures to ensure adequate data integrity. Algorithms that meet information theory's Singleton bound [6] protect against two disk failures by adding only two disks of redundancy to the number of disks required to store the unprotected data. Good algorithms meet this bound, and also store the data unencoded, so that it can be read directly off disk.

A multiple orders of magnitude improvement in the reliability of the storage system can simplify the design of other parts of the system for robustness, while improving overall system reliability. This motivates the use of a data protection algorithm that protects against double disk failures. At the same time, it is desirable to maintain the simplicity and performance of RAID-4 and RAID-5 single parity protection.

This paper describes a new algorithm, called Row-Diagonal Parity, or RDP, for protection against double failures. RDP applies to any multiple device storage system, or even to communication systems. In this paper, we focus on the application of RDP to disk array storage systems (RAID).

RDP is optimal both in computation and in I/O. It stores user data in the clear, and

requires exactly two parity disks. It utilizes only exclusive-or operations during parity construction as well as during reconstruction after one or two failures. Therefore, it can be implemented easily either in dedicated hardware, or on standard microprocessors. It is also simple to implement compared to previous algorithms. While it is difficult to measure the benefit of this, we were able to implement the algorithm and integrate it into an existing RAID framework within a short product development cycle.

In this paper, we make the case that the need for double disk failure protection is increasing. We then describe the RDP algorithm, proving its correctness and analysing its performance. We present some simple extensions to the algorithm, showing how to add disks to an existing array, and how to protect multiple RAID-4 or RAID-5 arrays against double failures with a single extra parity disk. Finally, we present some observations from our experience implementing RDP, and give some performance results for that implementation.

2 Related Work

There are several known algorithms that protect data against two or more disk failures in an array of disks. Among these are EVENODD [5], Reed Solomon (P+Q) erasure codes [6], DATUM [7] and RM2 [8]. RDP is most similar to EVENODD. RM2 distributes parity among the disks in a single stripe, or equivalently, adds stripes of parity data that are interspersed among the data stripes. EVENODD, DATUM, and Reed-Solomon P+Q all share the property that the redundant information can be stored separately from the data in each stripe. This allows implementations that have dedicated redundant disks, leaving the other disks to hold only data. This is analogous to RAID-4, although we have two parity disks, not one. We will call this RAID-4 style parity placement. Alternatively, the placement of the redundant information can be rotated from stripe to stripe, improving both read and write performance. We will call this RAID-5 style parity placement.

Both EVENODD and Reed-Solomon P+Q encoding compute normal row parity for one parity disk. However, they employ different techniques for encoding the second disk of redundant data. Both use exclusive-or operations, but Reed-Solomon encoding is much more computationally intensive than EVENODD [5]. DATUM uses encodings that generate any number of redundant information blocks. It allows higher order failure tolerance, and is similar to Reed-Solomon P+Q encoding in the case of protection against two disk failures.

RDP shares many of the properties of EVENODD, DATUM, and Reed-Solomon encoding, in that it stores its redundant data (parity) separately on just two disks, and that data is stored in the clear on the other disks. Among the previously reported algorithms, EVENODD has the lowest computational cost for protection against two disk failures. RDP improves upon EVENODD by further reducing the computational complexity. The complexity of RDP is provably optimal, both during construction and reconstruction. Optimality of construction is important as it is the normal, failure free operational mode. However, the optimality of reconstruction is just as important, as it maximizes the array's performance under degraded failure conditions [9].

3 Double Disk Failure Modes and Analysis

Double disk failures result from any combination of two different types of single disk failure. Individual disks can fail by whole-disk failure, whereby all the data on the disk becomes temporarily or permanently inaccessible, or by media failure, whereby a small portion of the data on a disk becomes temporarily or permanently inaccessible. Whole-disk failures may result from a problem in the disk itself, or in the channel or network connecting the disk to its containing system. While the mode and duration of the failures may vary, the class of failures that make the

data on a disk unaccessible can be categorized as one failure type for the purposes of recovery. Whole-disk failures require the complete reconstruction of a lost disk, or at least those portions of it that contain wanted data. This stresses the I/O system of the controller, while adding to its CPU load. (We will refer to the unit that performs construction of parity and reconstruction of data and parity as the *controller*.)

To maintain uninterrupted service, the controller has to serve requests to the lost disk by reconstructing the requested data on demand. At the same time, it will reconstruct the other lost data. It is desirable during reconstruction to have a low response time for the on-demand reconstruction of individual blocks that are required to service reads, while at the same time exhibiting a high throughput on the total disk reconstruction.

Whole-disk failure rates are measured as an arrival rate, regardless of the usage pattern of the disk. The assumption is that the disk can go bad at any time, and that once it does, the failure will be noticed. Whole disk failure rates are the reciprocal of the Mean Time To Failure numbers quoted by the manufacturers. These are typically in the range of 500,000 hours.

Media failures are qualitatively and quantitatively different from whole-disk failures. Media failures are encountered during disk reads and writes. Media failures on write are handled immediately, either by the disk or by the controller, by relocating the bad block to a good area on disk. Media failures on read can result in data loss. While a media failure only affects a small amount of data, the loss of a single sector of critical data can compromise an entire system. Handling media failures on read requires a short duration recovery of a small amount of missing data. The emphasis in the recovery phase is on response time, but reconstruction throughput is generally not an issue.

Disks protect against media errors by relocating bad blocks, and by undergoing elaborate retry sequences to try to extract data from a sector that is difficult to read [10]. Despite these precautions, the typical media error rate in disks is specified by the manufacturers as one bit error per 10^{14} to 10^{15} bits read, which corresponds approximately to one uncorrectable error per 10TBytes to 100TBytes transferred. The actual rate depends on the disk construction. There is both a static and a dynamic aspect to this rate. It represents the rate at which unreadable sectors might be encountered during normal read activity. Sectors degrade over time, from a writable and readable state to an unreadable state.

A second failure can occur during reconstruction from a single whole-disk failure. At this point, the array is in a degraded mode, where reads of blocks on the failed disk must be satisfied by reconstructing data from the surviving disks, and commonly, where the contents of the failed disk are being reconstructed to spare space on one or more other disks. If we only protect against one disk failure, a second complete disk failure will make reconstruction of a portion of both lost disks impossible, corresponding to the portion of the first failed disk that has not yet been reconstructed. A media failure during reconstruction will make reconstruction of the two missing sectors or blocks in that stripe impossible. Unfortunately, the process of reconstruction requires that all surviving disks are read in their entirety. This stresses the array by exposing all latent media failures in the surviving disks.

The three double disk failure combinations are: whole-disk/whole-disk, whole-disk/media, and media/media. A properly implemented double failure protection algorithm protects against all three categories of double failures. In our analysis of failure rates, we discount media/media failures as being rare relative to the other two double failure modes. Whole-disk/whole-disk and whole-disk/media failures will normally be encountered during reconstruction from an already identified whole-disk failure.

RAID systems can protect against double failures due to media failures by periodically "scrubbing" their disks, trying to read each sector, and reconstructing and relocating data on any sector that is unreadable. Doing this before a single whole-disk failure oc-

curs can preempt potential whole-disk/media failures by cleansing the disks of accumulated media errors before a whole-disk failure occurs. Such preventive techniques are a necessary precaution in arrays of current large capacity disks.

The media and whole-disk failure rates assume uniform failure arrivals over the lifetime of the disk, and uniform failure arrival rates over the population of similar disks. Actual whole-disk failure rates conform to a bathtub curve as a function of the disk's service time: A higher failure rate is encountered during the beginning-of-life burn-in and end-of-life wear-out periods. Both of these higher rate periods affect the double disk failure rate, as the disks in an array will typically be the same age, and will be subject to the same usage pattern. This tends to increase the correlation of whole-disk failures among the disks in an array.

Disks in the array may be from the same manufacturing batch, and therefore may be subject to the same variations in manufacturing that can increase the likelihood of an individual disk failing. Disks in an array are all subject to the same temperature, humidity and mechanical vibration conditions. They may all have been subjected to the same mechanical shocks during transport. This can result in a clustering of failures that increases the double failure rate beyond what would be expected if individual disk failures were uncorrelated.

Once a single disk fails, the period of vulnerability to a second whole-disk failure is determined by the reconstruction time. In contrast, vulnerability to a media failure is fixed once the first disk fails. Reconstruction will require a complete read of all the surviving disks, and the probability of encountering a media failure in those scans is largely independent of the time taken by reconstruction.

If the failures are independent, and wide sense stationary [12], then it is possible to derive the rate of occurance of two whole-disk failures as [2]:

$$\lambda_2 \approx \lambda_1^2 t_r c \frac{n(n-1)}{2} \qquad (1)$$

where t_r is the reconstruction time of a failed disk, n is the total number of disks in the array, λ_1 is the whole-disk failure rate of one disk, and c is a term reflecting the correlation of the disk failures. If whole-disk failures are correlated, then the correction factor $c > 1$. We know from experience that whole-disk failures are not stationary, i.e., they depend on the service time of the disk, and also that they are positively correlated. These factors will increase the rate λ_2.

The other consideration is that the reconstruction time t_r is a function of the total data that must be processed during reconstruction. t_r is linearly related to the disk size, but also can be related to the number of disks, since the total data to be processed is the product dn, where d is the size of the disks. For small n, the I/O bandwidths of the individual disks will dominate reconstruction time. However, for large enough n, the aggregate bandwidth of the disks becomes great enough to saturate either the I/O or processing capacity of the controller performing reconstruction. Therefore, we assert that:

$$\begin{aligned} t_r &= \begin{cases} d/b_r & \text{if } n < m \\ dn/b_s & n \geq m \end{cases} \qquad (2) \\ m &= \left\lceil \frac{b_s}{b_r} \right\rceil \end{aligned}$$

where b_r is the maximum rate of reconstruction of a failed disk, governed by the disk's write bandwidth and b_s is the maximum rate of reconstruction per disk array.

The result for disk arrays larger than m is:

$$\lambda_2 \approx \frac{\lambda_1^2 dc}{2b_s} n^2(n-1) \qquad (3)$$

The whole-disk/whole-disk failure rate has a cubic dependency on the number of disks in the array, and a linear dependency on the size of the disks. The double failure rate is related to the square of the whole-disk failure rate. If we employ disks that have higher failure rates, such as ATA drives, we can expect that the double failure rate will increase proportionally to the square of the increase in single disk failure rate.

As an example, if the primary failure rate is one in $500,000$ hours, the correlation factor is 1, the reconstruction rate is $100MB/s$,

in a ten disk array of 240 GByte disks, the whole-disk/whole-disk failure rate will be approximately 1.2×10^{-9} failures per hour.

Both the size of disks and their I/O bandwidth have been increasing, but the trend over many years has been that disk size is increasing much faster than the disk media rate. The time it takes to read or write an entire disk is the lower bound on disk recovery. As a result, the recovery time per disk has been increasing, further aggravating the double disk failure rate.

The rate of whole-disk/media failures is also related to disk size and to the number of disks in the array. Essentially, it is the rate of single whole-disk failures, multiplied by the probability that any of those failures will result in a double failure due to the inability to read all sectors from all surviving disks. The single whole-disk failure rate is proportional to the number of disks in the array. The media failure rate is roughly proportional to the total number of bits in the surviving disks of the array. The probability of all bits being readable is $(1-p)^s$ where p is the probability of an individual bit being unreadable, and s is the number of bits being read. This gives the a priori rate of whole-disk/media double failures:

$$f_2 = \lambda_1 n(1 - (1-p)^{(n-1)b}) \qquad (4)$$

where b is the size of each disk measured in bits.

For our example of a primary failure rate of 1 in 500,000 hours, a 10 disk array, 240 GB disks, and a bit error rate of 1 per 10^{14} gives a whole-disk/media double failure rate of 3.2×10^{-6} failures per hour.

In our example, using typical numbers, the rate of whole-disk/media failures dominates the rate of whole-disk/whole-disk failures. The incidence of media failures per whole-disk failure is uncomfortably high. Scrubbing the disks can help reduce this rate, but it remains a significant source of double disk failures.

The combination of the two double failure rates gives a Mean Time To Data Loss (MTTDL) of 3.1×10^5 hours. For our example, this converts to an annual rate of 0.028 data loss events per disk array per year due to double failures of any type.

To compare, the dominant triple failure mode will be media failures discovered during recovery from double whole-disk failures. This rate can be approximated by the analog to Equation 4:

$$f_3 = \lambda_2(1 - (1-p)^{(n-2)b}) \qquad (5)$$

Substituting λ_2 from Equation 1 gives:

$$f_3 \approx \frac{\lambda_1^2 dc}{2b_s} n^2(n-1)(1 - (1-p)^{(n-2)b}) \qquad (6)$$

For our example, the dominant component of the tertiary failure rate will be approximately 1.7×10^{-10} failures per hour, which is a reduction of over four orders of magnitude compared to the overall double failure rate.

The use of less expensive disks, such as ATA disks, in arrays where high data integrity is required has been increasing. The disks are known to be less performant and less reliable than SCSI and FCP disks [10]. This increases the reconstruction time and the individual disk failure rates, in turn increasing the double failure rate for arrays of the same size.

4 Row-Diagonal Parity Algorithm

The RDP algorithm is based on a simple parity encoding scheme using only exclusive-or operations. Each data block belongs to one row parity set and to one diagonal parity set. In the normal configuration, there is one row parity block and one diagonal parity block per stripe. It is possible to build either RAID-4 or RAID-5 style arrays using RDP, by either locating all the parity blocks on two disks, or by rotating parity from disk to disk in different stripes.

An RDP array is defined by a controlling parameter p, which must be a prime number greater than 2. In the simplest construction of an RDP array, there are $p+1$ disks. We

Data Disk 0	Data Disk 1	Data Disk 2	Data Disk 3	Row Parity	Diag. Parity
0	1	2	3	4	0
1	2	3	4	0	1
2	3	4	0	1	2
3	4	0	1	2	3

Figure 1: Diagonal Parity Set Assignments in a 6 Disk RDP Array, $p = 5$

define stripes across the array to consist of one block from each disk. In each stripe, one block holds diagonal parity, one block holds row parity, and $p - 1$ blocks hold data.

The bulk of the remainder of this paper describes one grouping of $p - 1$ stripes that includes a complete set of row and diagonal parity sets. Multiple of these stripe groupings can be concatenated to form either a RAID-4 style or RAID-5 style array. An extension to multiple row parity sets is discussed in Section 7.

Figure 1 shows the four stripes in a 6 disk RDP array ($p = 5$). The number in each block indicates the diagonal parity set the block belongs to. Each row parity block contains the even parity of the data blocks in that row, not including the diagonal parity block. Each diagonal parity block contains the even parity of the data and row parity blocks in the same diagonal. Note that there are $p = 5$ diagonals, but that we only store the parity of $p - 1 = 4$ of the diagonals. The selection of which diagonals to store parity for is completely arbitrary. We refer to the diagonal for which we do not store parity as the "missing" diagonal. In this paper, we always select diagonal $p - 1$ as the missing diagonal. Since we do not store the parity of the missing diagonal, we do not compute it either.

The operation of the algorithm can be seen by example. Assume that data disks 1 and 3 have failed in the array of Figure 1. It is necessary to reconstruct from the remaining data and parity disks. Clearly, row parity is useless in the first step, since we have lost two members of each row parity set. However, since each diagonal misses one disk, and all diagonals miss a different disk, then there are two diagonal parity sets that are only missing one block. At least one of these two diagonal parity sets has a stored parity block. In our example, we are missing only one block from each of the diagonal parity sets 0 and 2. This allows us to reconstruct those two missing blocks.

Having reconstructed those blocks, we can now use row parity to reconstruct two more missing blocks in the two rows where we reconstructed the two diagonal blocks: the block in diagonal 4 in data disk 3 and the block in diagonal 3 in data disk 1. Those blocks in turn are on two other diagonals: diagonals 4 and 3. We cannot use diagonal 4 for reconstruction, since we did not compute or store parity for diagonal 4. However, using diagonal 3, we can reconstruct the block in diagonal 3 in data disk 3. The next step is to reconstruct the block in diagonal 1 in data disk 1 using row parity, then the block in diagonal 1 in data disk 3, then finally the block in diagonal 4 in data disk 1, using row parity.

The important observation is that even though we did not compute parity for diagonal 4, we did not require the parity of diagonal 4 to complete the reconstruction of all the missing blocks. This turns out to be true for all pairs of failed disks: we never need to use the parity of the missing diagonal to complete reconstruction. Therefore, we can safely ignore one diagonal during parity construction.

5 Proof of Correctness

Let us formalize the construction of the array. We construct an array of $p + 1$ disks divided into blocks, where p is a prime number greater than 2. We group the blocks at the same position in each device into a stripe. We then take groups of $p - 1$ stripes and, within that group of stripes, assign the blocks to diagonal parity sets such that with disks numbered $i = 0 \ldots p$ and blocks numbered $k = 0 \ldots p - 2$ on each disk, disk block (i, k) belongs to diagonal parity set $(i + k) \bmod p$.

Disk p is a special diagonal parity disk. We construct row parity sets across disks 0 to $p-1$ without involving disk p, so that any one lost block of the first p disks can be reconstructed from row parity. The normal way to ensure this is to store a single row parity block in one of the blocks in each stripe. Without loss of generality, let disk $p-1$ store row parity.

The key observation is that the diagonal parity disk can store diagonal parity for all but one of the p diagonals. Since the array only has $p-1$ rows, we can only store $p-1$ of the p possible diagonal parity blocks in each group of $p-1$ stripes. We could select any of the diagonal parity blocks to leave out, but without loss of generality, we choose to not store parity for diagonal parity set $p-1$, to conform to our numbering scheme.

The roles of all the disks other than the diagonal parity disk are mathematically identical, since they all contribute symmetrically to the diagonal parity disk, and they all contribute to make the row parity sums zero. So, in any stripe any one or more of the non-diagonal parity disks could contain row parity. We only require that we be able to reconstruct any one lost block in a stripe other than the diagonal parity block from row parity without reference to the diagonal parity block.

We start the proof of the correctness of the RDP algorithm with a necessary Lemma.

Lemma 1 *In the sequence of numbers* $\{(p-1+kj) \bmod p, k = 0 \ldots p\}$*, with* p *prime and* $0 < j < p$*, the endpoints are both equal to* $p-1$*, and all numbers* $0 \ldots p-2$ *occur exactly once in the sequence.*

Proof: The first number in the sequence is $p-1$ by definition. The last number in the sequence is $p-1$, since $(p-1+pj) \bmod p = p-1+(pj \bmod p) = p-1$. Thus the lemma is true for the two endpoints. Now consider the subsequence of $p-1$ numbers that begins with $p-1$. All these numbers must have values $0 \le x \le p-1$ after the modulus operation. If there were a repeating number x in the sequence, then it would have to be true that $(x+kj) \bmod p = x$ for some $k < p$. Therefore, $kj \bmod p = 0$ which means that kj is divisible by p. But since p is prime, no multiple of k or j or any of their factors can equal p. Therefore, the first $p-1$ numbers in the sequence beginning with $p-1$ are unique, and all numbers from $0 \ldots p-1$ are represented exactly once. The next number in the sequence is $p-1$. $\diamond$

We now complete the proof of the correctness of RDP.

Theorem 1 *An array constructed according to the formal description of RDP can be reconstructed after the loss of any two of its disks.*

Proof: There are two classes of double failures, those that include the diagonal parity disk, and those that do not.

Those failures that include the diagonal parity disk have only one disk that has failed in the row parity section of the array. This disk can be reconstructed from row parity, since the row parity sets do not involve the diagonal parity disk. Upon completion of the reconstruction of one of the failed disks from row parity, the diagonal parity disk can be reconstructed according to the definition of the diagonal parity sets.

This leaves all failures of any two disks that are not the diagonal parity disk.

From the construction of the array, each disk d intersects all diagonals except diagonal $(d+p-1) \bmod p = (d-1) \bmod p$. Therefore, each disk misses a different diagonal.

For any combination of two failed disks d_1, d_2 with $d_2 = d_1 + j$, the two diagonals that are not intersected by both disks are

$$\begin{aligned} g_1 &= (d_1 + p - 1) \bmod p \\ g_2 &= (d_1 + j + p - 1) \bmod p \end{aligned}$$

Substituting g_1 gives

$$g_2 = (g_1 + j) \bmod p$$

Since each of these diagonals is only missing one member, if we have stored diagonal parity for the diagonal we can reconstruct the

missing element along that diagonal. Since at most one of the diagonals is diagonal $p-1$, then we can reconstruct at least one block on one of the missing disks from diagonal parity as the first step of reconstruction.

For the failed disks d_1, d_2, if we can reconstruct a block from diagonal parity in diagonal parity set x on disk d_1, then we can reconstruct a block on disk d_2 in diagonal parity set $(x + j) \bmod p$, using row parity. Similarly, if we can reconstruct a block x from diagonal parity on disk d_2, then we can reconstruct a block on disk d_1 in diagonal parity set $(x - j) \bmod p$ using row parity.

Consider the pair of diagonals g_1, g_2 that are potentially reconstructable after the failure of disks d_1, d_2. If g_1 is reconstructable, then we can reconstruct all blocks on each diagonal $(g_1 - j) \bmod p, (g_1 - 2j) \bmod p, \ldots, p-1$ using alternating row parity and diagonal parity reconstructions. Similarly, if g_2 is reconstructable, then we can reconstruct all blocks on each diagonal $(g_2 + j) \bmod p, (g_2 + 2j) \bmod p, \ldots, p-1$ using alternating row parity and diagonal parity reconstructions. Since g_1 and g_2 are adjacent points on the sequence for j generated by Lemma 1, then we reach all diagonals $0 \ldots p-1$ during reconstruction.

If either $g_1 = p-1$ or $g_2 = p-1$, then we are only missing one block from the diagonal parity set $p-1$, and that block is reconstructed from row parity at the end of the reconstruction chain beginning with g_2 or g_1 respectively. If both $g_1 \neq p-1$ and $g_2 \neq p-1$, then the reconstruction proceeds from both g_1 and g_2, reaching the two missing blocks on diagonal $p-1$ at the end of each chain. These two blocks are each reconstructed from row parity.

Therefore, all diagonals are reached during reconstruction, and all missing blocks on each diagonal are reconstructed. $\diamond$

We do not need to store or generate the parity of diagonal $p-1$ to complete reconstruction.

6 Performance Analysis

Performance of disk arrays is a function of disk I/O as well as the CPU and memory bandwidth required to construct parity during normal operation and to reconstruct lost data and parity after failures. In this section, we analyse RDP in terms of both its I/O efficiency and its compute efficiency.

Since RDP stores data in the clear, read performance is unaffected by the algorithm, except to the extent that the disk reads and writes associated with data writes interfere with data read traffic. We consider write I/Os for the case where $p-1$ RDP stripes are contained within a single stripe of disk blocks, as described in Section 7. This implementation optimizes write I/O, and preserves the property that any stripe of disk blocks can be written independently of all other stripes. Data writes require writing two parity blocks per stripe. Full stripe writes therefore cost one additional disk I/O compared to full stripe writes in single disk parity arrays. Partial stripe writes can be computed by addition, i.e. recomputing parity on the entire stripe, or subtraction, i.e. computing the delta to the parity blocks from the change in each of the data blocks written to, depending on the number of blocks to be written in the stripe. Writes using the subtraction method are commonly referred to as "small writes". Writing d disk blocks by the subtraction method requires $d+2$ reads and $d+2$ writes. The addition method requires $n-d-2$ reads, and $d+2$ writes to write d disk blocks. If reads and writes are the same cost, then the addition method requires n I/Os, where n is the number of disks in the array, and the subtraction method requires $2d+4$ I/Os. The breakpoint between the addition and subtraction method is at $d = (n-4)/2$. The number of disk I/Os for RDP is minimal for a double failure protection algorithm; writing any one data block requires updating both parity blocks, since each data block must contribute to both parity blocks.

We next determine the computational cost of RDP as the total number of exclusive or (xor) operations needed to construct parity. Each data block contributes to one row par-

ity block. In an array of size $p-1$ rows $\times$ $p+1$ disks, there are $p-1$ data blocks per row, and $p-2$ xor operations are required to reduce those blocks to one parity block. Row parity thus requires $(p-1)(p-2) = p^2-3p+2$ xors. We also compute $p-1$ diagonal parity blocks. Each diagonal contains a total of $p-1$ data or row parity blocks, requiring $p-2$ xors to reduce to one diagonal parity block. Therefore diagonal parity construction requires the same number of xors as row parity construction, $(p-1)(p-2) = p^2-3p+2$. The total number of xors required for construction is $2p^2-6p+4$.

Theorem 2 *For an array of n data disks, a ratio of $2-2/n$ xors per block is the minimum number of xors to provide protection against two failures.*

Proof: Assume that we construct parity in the $n+2$ disk array within groups of r rows. We have a minimum of two parity blocks per row of n data blocks, from the Singleton bound. Each data block must contribute to at least two different parity blocks, one on each parity disk, to ensure that we can recover if the data block and one parity block is lost. Any pair of data blocks that contributes to two different parity blocks provides no additional information since losing both data blocks will make all the parity sets they contribute to ambiguous. Therefore, we need to construct $2r$ parity blocks from equations that in the minimal formulation contain no common pairs of data blocks. Since this allows no common subterms between any equations in the minimal formulation, the minimum number of separately xored input terms required to construct the $2r$ parity blocks is $2nr$. A set of $2r$ equations that reduces $2nr$ terms to $2r$ results using xors requires $2nr-2r$ xors. Therefore, the minimum number of xors per data block to achieve double parity protection is:

$$\frac{2nr-2r}{nr} = 2-\frac{2}{n} \tag{7}$$

$\diamond$

RDP protects $(p-1)^2$ data blocks using $2p^2-6p+4$ xors. Setting $n = p-1$, we get $2n^2-2n$ xors to protect n^2 data blocks, which meets the optimal ratio of $2-2/n$.

Data disks	RDP	EVENODD	Difference
4	6	6.67	11.1%
6	10	10.8	8.0%
8	14	14.86	6.1%
12	22	22.91	4.1%
16	30	30.93	3.1%

Table 1: Per Row XOR Counts for Parity Construction

We can compare RDP to the most computationally efficient previously known algorithm, EVENODD. For an array with n data disks, each with $n-1$ data blocks, EVENODD requires $(n-1)(n-1)$ xors to compute row parity, and $(n-2)n$ xors to compute the parity of the n diagonals. [1] EVENODD then requires a further $n-1$ xors to add the parity of one distinguished diagonal to the parity of each of the other $n-1$ diagonals to complete the calculation of stored diagonal parity. This results in a total of $2n^2-3n$ xors to construct parity in EVENODD for an $n(n-1)$ block array. Therefore, EVENODD requires $(2n^2-3n)/(n^2-n) = 2-1/(n-1)$ xors per block.

The two algorithms both have an asymptotic cost of two xors per block. However, the difference in computational cost is significant for the small values of n typical in disk arrays, as shown in Table 1, ignoring the fact that the two algorithms do not function correctly for the same array sizes.

RDP's computational cost of reconstruction after a failure is also optimal. Reconstruction from any single disk failure requires exactly $(p-1)(p-2) = p^2-3p+2$ xors, since each of the $p-1$ lost row parity sets or diagonal parity sets are of the same size p, and we must reconstruct the lost block in each by xoring the surviving $p-1$ blocks, using $p-2$ xor operations. Again setting $n = p-1$, we are recovering n blocks with n^2-n xors, which is $n-1$ xors per parity block. Since we have already shown that we have the minimum number of xors for construction of an array that double protects parity, and since

[1] This is fewer operations than the result given in the EVENODD paper [5], which we believe overcounts the number of xors required to compute parity along a diagonal.

Data disks	RDP	EVENODD	Difference
4	6	9.67	61.2%
6	10	13.80	83.0%
8	14	17.86	27.6%
12	22	25.91	17.8%
16	30	33.93	13.1%

Table 2: Per Row XOR Counts for Data Reconstruction

all parity sets are the same size, then the cost to repair any one lost disk is the same and is also a minimum. We can't make the individual parity sets any smaller and still protect against double failures, and we are reconstructing each block from exactly one parity set. This is true for any disk we might lose.

Reconstructing from any double failure that includes the diagonal parity disk is exactly the same cost as parity construction, since we first reconstruct the lost data or row parity disk from row parity, then reconstruct the diagonal parity disk. Reconstructing any of the data disks from row parity has the same cost as contructing row parity.

The cost of reconstructing any combination of two data or row parity disks can also be determined. We have to reconstruct exactly $2(p-1)$ blocks. Each parity set is of size $p-1$, so the cost to reconstruct each block is again $p-2$ xors. This gives us exactly the same computational cost as construction, and as the other reconstruction cases: $2p^2-6p+4 = 2n^2-2n$ xors. Again, this is optimal.

Comparing again to EVENODD, using the data reconstruction algorithm described in the EVENODD paper, we see an advantage for RDP, as shown in Table 2.

For the numbers of disks that are typical in disk arrays, the performance of the RDP and EVENODD in construction and reconstruction is significantly different. Both are much lower in compute cost than Reed-Solomon coding [5]. RDP is optimal both in compute efficiency and I/O efficiency, during construction in normal operation and reconstruction after a failure.

7 Algorithm Extensions

Single Stripe Implementation: Selecting p to be one of the primes that meets the condition $p = 2^n + 1$ for some n (e.g. 5, 17, 257), allows us to define diagonal parity sets within a group of 2^n stripes. This allows us to define the block size for RDP purposes to be the usual system block size divided by 2^n. Since the disk block sizes are usually powers of two, we can define a self-contained RDP parity set within a single stripe of blocks. For example, if the system's disk block size is 4kB, we can select $p = 17$, giving us 16 RDP blocks per stripe, with each RDP block containing 256 bytes. This allows us to construct an array using all the existing software and techniques for reading and writing a single stripe, adding one disk to contain diagonal parity.

Multiple Row Parity Groups: RDP requires that we be able to recover a single lost block in a stripe using row parity in any case where that block is not on the diagonal parity disk. In both the RAID-4 and RAID-5 style configurations, it is possible to have more than one row parity set per stripe, each with its own row parity block. This means that the portion of the array that does not include the diagonal disk can use any single disk reconstruction technique, including concatenation of more than one RAID-4 or RAID-5 array, or declustered parity techniques [11]. We define diagonal parity sets across all of these disks, and construct diagonal parity from these sets, regardless of whether the individual blocks stored are parity or data blocks. This allows a cost-performance tradeoff between minimizing the number of parity disks, and making reconstruction from single failures faster while still protecting against double failures in the wider array. In such an array, double failures that affect disks in two different row parity sets can be repaired directly from row parity.

There is one other technique for expanding diagonal parity to cover more than one row parity group. Imagine that we have several RDP arrays, all with the same file system block size, although not necessarily with the same value of p. If we xor all of their diagonal

parity blocks together, we will get a single diagonal parity block. We could store a single diagonal parity disk, which is the combination of the diagonal parity disks of each of the constituent arrays. Storing this is sufficient to allow reconstruction from any double disk loss in the array. Any two failures that occur in two different subarrays can be recovered by local row parity. Any two failures that occur in the same subarray must be recovered by diagonal parity. The diagonal parity block for any subarray can be reconstructed by constructing the diagonal parity for each of the intact subarrays, and subtracting it from the stored merged diagonal parity disk. Once we have reconstructed the needed diagonal parity contents, we use normal RDP reconstruction to rebuild the lost blocks of the subarray that we are reconstructing.

Expandable Arrays: The discussion so far has implied that the number of disks in an array is fixed at $p + 1$ for any selection of p. This is not actually the case. We can underpopulate an RDP array, putting fewer data disks than the maximum allowed in the array for a given value of p. $p+1$ simply sets the maximum array size for a given value of p. When we underpopulate an array, we are taking advantage of the fact that given fewer than $p - 1$ data disks, we could fill the remainder of the array with unused disks that contain only zeros. Since the zero-filled disks contribute to parity blocks, but do not change the contents of any parity block, we can remove them from the array while still imputing their zero-filled contents in our parity calculations. This allows us to expand the array later by adding a zero-filled disk, and adjusting parity as we later write data to that disk.

By the same reasoning, it is allowable to have disks of different sizes in the array. The diagonal parity disk must be one of the largest disks in the array, and all rows must have at least one row parity block. The contributions of the smaller disks to stripes that do not contain blocks from those disks are counted as zeros.

By selecting $p = 257$, we allow the RDP array to grow to up to 255 data disks. This is a sufficiently large number to accomodate any expected disk array size.

8 Implementation Experience

RDP has been implemented as a new feature of Network Appliance's data storage system software (Data ONTAP) version 6.5. Data ONTAP is a complete software system, including an operating system kernel, networking, storage, file system, file system protocols, and RAID code. The RAID layer manages the final layout of data and parity within RAID groups. A volume consists of one or more RAID groups, and each RAID group is independently recoverable. This section contains some observations we made that improved the implementation of the algorithm.

During parity construction, most of the subblocks of each data disk contribute to both a row parity subblock and a diagonal parity subblock. We also note that the contributions of the subblocks of any data disk to the diagonal parity disk are ordered in sequence. This allows us to perform parity construction in a memory efficient manner on modern microprocessors. We process each data block in one pass, xoring its contents into both the row parity and diagonal parity destination blocks. By properly tuning the code to the microprocessor, it is possible to work on all three blocks in the top level of CPU cache. We work on one data block at a time, incrementally constructing the two target parity blocks, which remain in cache. The latency of memory operations gives us a budget for completing two xor operations per 128 bit field on the Pentium 4. We further optimize by ensuring that the data bytes are xored into both destinations once loaded into processor registers. In our implementation, we had enough instructions available per cache line load to complete both xors and a data integrity checksum calculation on the data in each cache line, without a significant loss of performance. This overlap of cpu execution and memory operations greatly reduced the effective cost of computing the second redundant parity block in RDP.

Another observation is that, having protected the array against double failures, our remaining vulnerability is to triple and higher order failures. Whole-disk/media failures are

corrected as they are encountered, by resorting to using RDP for reconstruction only in those stripes that are missing two blocks. The remainder of the missing disk can be reconstructed using row parity, unless it is the diagonal parity disk.

In the case of whole-disk/whole-disk failures, reconstruction of the first disk to fail typically is already underway when the second disk fails. In our implementation, reconstruction from a single disk failure starts from block number 0 and proceeds sequentially to the last disk block. When the second failure occurs, the first stripes of the array are missing only one block, since we have completed reconstruction of the single failure in some stripes. So, we only need to run RDP reconstruction on the remaining stripes of the array. Stripes with two missing blocks are always reconstructed before those with one missing block, reducing the window of vulnerability to a third disk failure. All combinations of disk failures are handled, including those involving reconstructing disks.

Existing RAID-4 and RAID-5 arrays can be easily upgraded to RDP by constructing the diagonal parity disk, using the same code as is used for reconstructing from single diagonal disk failures. Downgrading from RDP to a single parity protection scheme is as simple as removing the diagonal disk.

9 Measured Performance

Data ONTAP version 6.5 runs on a variety of hardware platforms. The current highest performing platform is the FAS980, which includes two 2.8GHz Intel Pentium 4 CPUs per file server (filer). At any time, up to one full CPU can be running RAID code, including xor calculations for RDP. We ran several performance benchmarks using the implementation of RDP in Data ONTAP 6.5.

The first set of experiments is *xortest*, which is our own synthetic benchmark for testing RAID xor and checksum code. The checksum is a modified Adler checksum that is 64 bits wide, computed on each input block. The input is a stripe that is one block deep by n blocks wide. Blocks are 4kB. The RDP prime is 257, and we divide each 4kB block into 256 sixteen byte subblocks for RDP calculations. The *xortest* experiment is run with cold caches, using random data generated in memory. There is no disk access in the test; it is simply a test of memory and CPU processing speed.

We ran two sets of tests. In the first, we computed parity from the input blocks, and also computed the checksum of each input block and the output parity blocks. In the second, we computed no checksums, only parity. In each set of tests, we computed single parity (RAID-4), double parity (RDP), and also performed RDP reconstruction on two randomly selected missing data blocks. We repeated each experiment five times and took the best results, to eliminate the effects of other activity in the operating system. Generally, the best results are repeatable, with a few bad outliers that represent experiments that were interfered with by other activity affecting the processor and cache.

Figures 2 and 3 present the results of the experiments. Note that all the graphs are very linear, with a small offset due to fixed overhead in the algorithm. In each case, the single parity calculation of RAID-4 is fastest. Table 3 shows the measured calculation rates for the various operations. Note that the RDP reconstruction rate is very close to the RDP construction rate. The difference in timings between the two is due primarily to the completion step in reconstruction, which requires a series of operations on the 16 byte RDP blocks. This step is required regardless of the number of blocks in the stripe. Otherwise, the per block computations during RDP construction and reconstruction are basically the same in our implementation. The reconstruction completion step is accounted for in the overhead per operation, determined as the time taken by a hypothetical calculation on a stripe of zero data blocks. The overhead for RDP reconstruction is significantly higher due to the completion step in both cases (Table 3).

The construction and reconstruction rates are close to those obtained for RAID-4 con-

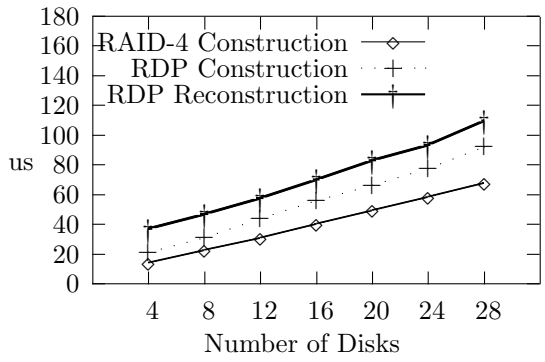

Figure 2: *Xortest* RAID-4 and RDP Performance with Checksum Computations (One 4k block per disk)

		Rate (GB/s)	Overhead (μs)
With Checksum	RAID-4	1.82	4.7
	RDP cons.	1.39	8.3
	RDP recons.	1.37	23.0
Without Checksum	RAID-4	1.90	2.6
	RDP cons.	1.55	4.6
	RDP recons.	1.60	19.8

Table 3: *Xortest* Derived Results

Algorithm	Config. $g \times (d+p)$	Rate (MB/s)
RAID-4	$6 \times (7+1)$	158.3
RDP	$6 \times (7+2)$	149.1
RDP	$4 \times (10+2)$	155.1
RDP	$3 \times (14+2)$	157.2

Table 4: *Aggwrite* Results

struction. (RAID-4 construction and reconstruction are identical computations.) The difference in the rates reflects our inability to completely overlap computation with cache loads and stores to main memory. The theoretical memory bandwidth of the processor is 3.2 GB/s. We are achieving from 43 to 59 percent of this rate, which indicates that we are stalling on cache line loads or are saturating the processor. A calculation of the instruction counts per cache line indicates that we are consuming all of the processing budget available per cache line in the checksum cases.

Aggwrite is a test of the filer's aggregate write performance. The workload is supplied by an array of NFS clients, performing 32kB write operations. Again, these tests are performed using an FAS980 with dual 2.8GHz Intel Pentium 4 processors. The filer runs the entire data path, from network, NFS protocol, file system, RAID and storage. We compared RAID-4 with various configurations of RDP, using 40 or 42 data disks in each case, and measured the achievable write bandwidth. Table 4 gives the aggwrite results.

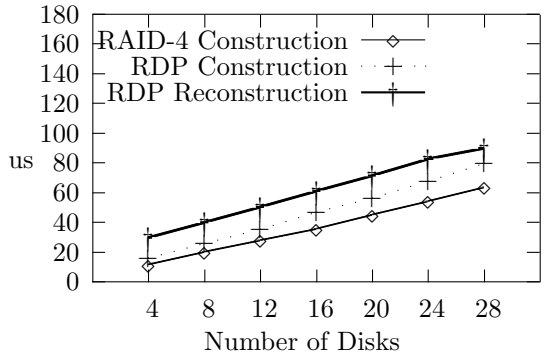

Figure 3: *Xortest* RAID-4 and RDP Performance without Checksum Computations (One 4k block per disk)

The configuration column of Table 4 presents $g \times (d+p)$, where g is the number of separate RAID groups connected to the filer, d is the number of data disks per RAID group, and p is the number of parity disks per RAID group. The WAFL file system uniformly distributes writes across all the data disks. Table 4 indicates that in all cases, RDP performance is within 6 percent of RAID-4. With RDP, we can increase the size of the RAID groups, still realizing an increase in data protection, while achieving comparable write performance. Using RDP RAID groups of 16 disks (14 data and 2 parity) we achieve performance almost equivalent to RAID-4, with the same total number of data and parity disks, and with much improved data protection.

10 Conclusions

RDP is an optimally efficient double disk failure protection algorithm. The combination of a single whole-disk failure with one or more media failures is becoming particularly troublesome as disks get large relative to the expected bit error rate. Utilizing RDP, we can significantly reduce data loss due to all types of double failures. The fact that RDP is optimal in I/O and in computational complexity proved valuable in achieving performance that is very close to our single parity RAID-4 implementation. The simplicity and flexibility of RDP allowed us to implement it within our existing RAID framework. An interesting open problem is whether the algorithm can be extended to cover three or more concurrent failures.

11 Acknowledgements

The authors would like to thank Loellyn Cassell, Ratnesh Gupta, Sharad Gupta, Sanjay Kumar, Kip Macy, Steve Rogridues, Divyesh Shah, Manpreet Singh, Steve Strange, Rajesh Sundaram, Tom Theaker, and Huan Tu for their significant contributions to this project.

References

[1] K. Salem, and H. Garcia-Molina, "Disk striping", *Proceedings of the 2nd Internation Conference on Data Engineering*, pgs.336-342, Feb. 1986.

[2] 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*, pgs. 109-116, 1988.

[3] W. Burkhard, and J. Menon, "Disk array storage system reliability". *Proceedings of the International Symposium on Fault-tolerant Computing*, pgs.432-441, 1993.

[4] Qin Xin, E. Miller, T. Schwarz, D. Long, S. Brandt, W. Litwin, "Reliability mechanisms for very large storage systems", *20th IEEE/11th NASA Boddard Conference on Mass Storage Systems and Technologies*, San Diego, CA, pgs. 146-156, Apr. 2003.

[5] 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*, pgs. 245-254, 1994.

[6] F. J. MacWilliams and J. J. A. Sloane. *The Theory of Error-Corrrecting Codes*, North-Holland, 1977.

[7] G. A. Alvarez, W. A. Burkhard, and F. Christian, "Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering" *Proceedings of the 24th Annual Symposium on Computer Architecture* pgs. 62-72, 1997.

[8] C. I. Park, "Efficient placement of parity and data to tolerate two disk failures in disk array systems". *IEEE Transactions on Parallel and Distributed Systems*, Nov. 1995.

[9] R. Muntz, and J. Lui, "Performance analysis of disk arrays under failure." *Proceedings of the 16th VLDB Conference* pgs.162-173, Brisbane, June 1990.

[10] D. Anderson, J. Dykes, and E. Riedel, "More than an interface - SCSI vs. ATA". *2nd USENIX Conference on File and Storage Technologies*, San Francisco, CA. pgs. 245-257, March, 2003.

[11] M. Holland, and G. Gibson, "Parity deculstering for continuous operation on redundant disk arrays", *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems.* pgs. 23-25, 1992.

[12] A. Papoulis, *Probability, Random Variables, and Stochastic Processes, Second Edition*, McGraw-Hill, New York, 1984.