Erasure Code based Cloud Storage File system-Minimizing IO Recovery

S.Annie Joice* J.Vasanth Wason**

(Department of Computer Science and Engineering, M.A.M. College of Engineering and Technology, Tiruchirappalli-621105) (Department of Computer Science and Engineering, M.A.M. College of Engineering and Technology, Tiruchirappalli-621105)

ABSTRACT

Cloud storage is a model of networked online storage where data is stored in virtualized pools of storage which are generally hosted by third parties. To reduce storage overhead, cloud file systems are transitioning from replication to erasure codes. This paper present an algorithm that finds the optimal number of codeword symbols needed for recovery for any XOR-based erasure code and produces recovery schedules that use a minimum amount of data. Several cloud systems have adopted Reed-Solomon (RS) codes, because of their generality and their ability to tolerate larger numbers of failures. We define a new class of rotated Reed-Solomon codes that perform degraded reads more efficiently than all known codes, but otherwise inherit the reliability and performance properties of Reed-Solomon codes.

Keywords-Erasure Code, Reed-Solomon code, Reliability, Replication, Virtualized pool

1. Introduction

A cloud storage system, consisting of a collection of storage servers, provides long-term storage services over the Internet. Storing data in a third party's cloud system causes serious concern over data confidentiality. The proposed cloud file system using erasure codes is inspired by Microsoft Azure [10]. It conforms well with HDFS [8] modified for RAID-6 [14] and Google's analysis of redundancy coding [15]. Some cloud file systems, such as Microsoft Azure and the Google File system, create an append-only write workload using a large block size. Writes are accumulated and buffered until a block is full and then the block is sealed; it is erasure coded and the coded blocks are distributed to storage nodes. Subsequent reads to sealed blocks often access smaller amounts data than the block size, depending upon workload [14, 46]. When examining erasure codes in the context of cloud file systems, two performance critical operations emerge. These are degraded reads to temporarily unavailable data and recovery from single failures. Although erasure codes tolerate multiple simultaneous failures, single failures represent 99.75% of recoveries [44]. Recovery performance has always been important. Previous work includes architecture support [13, 21]

and workload optimizations for recovery [22, 48, 45]. However, it is particularly acute in the cloud owing to scale. Massive systems have frequent component failures so that recovery becomes part of regular operation [16]. Frequent and temporary data unavailability in the cloud results in degraded reads. In the period between failure and recovery, reads are degraded because they must reconstruct data from unavailable storage nodes using erasure codes. This is by necessity a slower operation than reading the data without reconstruction. Temporary unavailability dominates disk failures. Transient errors in which no data are lost account for more than 90% of data center failures [15], owing to network partitions, software problems, or non-disk hardware faults. For this reason, Google delays the recovery of failed storage nodes for 15 minutes. Temporary unavailability also arises systematically when software upgrades take storage nodes offline. In many data centers, software updates are a rolling, continuous process. Only recently have techniques emerged to reduce the data requirements of recovering an erasure code. Two recent research projects have demonstrated how the RAID- 6 codes RDP and EVENODD may recover from single disk failures by reading significantly smaller subsets of codeword symbols than the previous standard practice of recovering from the parity drive [51, 49]. Recovery performance generalize these results to all XOR-based erasure codes, analyze existing codes to differentiate them based on recovery performance, and experimentally verify that reducing the amount of data used in recovery translates directly into improved performance for cloud file systems, but not for typical RAID array configurations.

An algorithm that finds the optimal number of symbols needed for recovering data from an arbitrary number of disk failures, which also minimizes the amount of data read during recovery. This paper includes an analysis of single failures in RAID-6 codes that reveals that sparse codes, such as Blaum-Roth [5], Liberation [34] and Liber8tion [35], have the best recovery properties, reducing data by about 30% over the standard technique that recovers each row independently. This paper also analyzes codes that tolerate three or more disk failures,

including the Reed-Solomon codes used by Google [15] and Microsoft Azure [10].

The algorithm demonstrates that minimizing recovery data translates directly into improved I/O performance for cloud file systems. Prior work on minimizing recovery I/O [51, 49, 27] is purely analytic, whereas our work incorporates measurements of recovery performance.

Reed-Solomon codes are particularly poor for degraded reads in that they must always read all data disks and parity for every degraded read. This is problematic because RS codes are popular owing to their generality and applicability to nearly all coding situations. A new class of codes, Rotated Reed-Solomon codes that exceed the degraded read performance of all other codes, but otherwise have the encoding performance and reliability properties of RS Codes. Rotated RS codes can be constructed for arbitrary numbers of disks and failures.

2. Related Work

Performance Metrics: Erasure codes have been evaluated historically on a variety of metrics, such as the CPU impact of encoding and decoding [3, 11, 37], the penalty of updating small amounts of data [5, 26, 52] and the ability to reconfigure systems without re-encoding [3, 7, 26]. The CPU performance of different erasure codes can vary significantly. However, for all code consider, encoding and decoding bandwidth is orders of magnitude faster than disk bandwidth. Thus, the dominant factor when sealing data is writing the erasure coded blocks to disk, not calculating the codes. Similarly, when decoding either for recovery or for degraded reads, the dominant factor is reading the data.

Updating small amounts of data is also not a concern in cloud file systems—the append-only write pattern and sealed blocks eliminate small writes in their entirety. System reconfiguration refers to changing coding parameters: changing the stripe width or increasing/decreasing fault tolerance. This type of reconfigurability is less important in clouds because each sealed block defines an independent stripe group, spread across cloud storage nodes differently than other sealed blocks. There is no single array of disks to be reconfigured. If the need for reconfiguration arises, each sealed block is reencoded independently.

There has been some work lowering I/O costs in erasure-coded systems. In particular, WEAVER [19], Pyramid [23] and Stepped Combination Codes [18] have all been designed to lower I/O costs on recovery. However, all of these codes are non-MDS, which means that they do not have the storage efficiency that cloud storage systems demand. The REO RAID Engine [26] minimizes I/O

in erasure-coded storage systems; however, its focus is primarily on the effect of updates on storage systems of smaller scale.

Cloud Storage Systems: The default storage policy in cloud file systems has become triplication (triple replication), implemented in the Google File system [16] and adopted by Hadoop [8] and many others. Triplication has been favored because of its ease of implementation, good read and recovery performance, and reliability.

The storage overhead of triplication is a concern, leading system designers to consider erasure coding as an alternative. The performance tradeoffs between replication and erasure coding are well understood and have been evaluated in many environments, such as peer-to-peer file systems [43, 50] and open-source coding libraries [37].

Investigations into applying RAID-6 (two fault tolerant) erasure codes in cloud file systems show that they reduce storage overheads from 200% to 25% at a small cost in reliability and the performance of large reads [14]. Microsoft research further explored the cost/benefit tradeoffs and expand the analysis to new metrics: power proportionality and complexity [53]. For these reasons, Facebook is evaluating RAID-6 and erasure codes in their cloud infrastructure [47].

Ford et al. [15] have developed reliability models for Google's cloud file system and validated models against a year of workload and failure data from the Google infrastructure. Their analysis concludes that data placement strategies need to be aware of failure groupings and failure bursts. They also argue that, in the presence of correlated failures, codes more fault tolerant than RAID- 6 are needed to to reduce exposure to data loss; they consider Reed-Solomon codes that tolerate three and four disk failures. Windows Azure storage employs Reed-Solomon codes for similar reasons [10]. The rotated RS codes that we present inherit all the properties of Reed- Solomon codes and improve degraded reads.

Recovery Optimization: Workload-based approaches to improving recovery are independent of the choice of erasure code and apply to minimum I/O recovery algorithm and rotated RS codes that we present. These include: load-balancing recovery among disks [22], recovering popular data first to decrease read degradation [48], and only recovering blocks that contain live data [45]. Similarly, architecture support for recovery can be applied to our codes, such as hardware that minimizes data copying [13] and parity declustering [21].

Reducing the amount of data used in recovery has only emerged recently as a topic and the first results have given minimum recovery schedules

for EVENODD [49] and row-diagonal parity [51], both RAID-6 codes. An algorithm that defines the recovery I/O lower bound for any XOR-based erasure code and allows multiple codes to be compared for I/O recovery cost. Regenerating codes provide optimal recovery bandwidth [12] among storage nodes. This concept is different than minimizing I/O; each storage node reads all of its available data and computes and sends a linear combination.

Regenerating codes were designed for distributed systems in which wide-area bandwidth limits recovery performance. Exact regenerating codes [39] recover lost data exactly (not a new linear combination of data). In addition to minimizing recovery bandwidth, these codes can in some cases reduce recovery I/O. The relationship between recovery bandwidth and recovery data size remains an open problem.

3. Background: Erasure Coded Storage

Erasure coded storage systems add redundancy for fault tolerance. Specifically, a system of n disks is partitioned into k disks that hold data and m disks that hold coding information. The coding information is calculated from the data using an erasure code. For practical storage systems, the erasure code typically has two properties. First, it must be Maximum Distance Separable (MDS), which means that if any m of the n disks fails, their contents may be recomputed from the k surviving disks. Second, it must be systematic, which means that the k data disks hold unencoded data.

An erasure coded storage system is partitioned into stripes, which are collections of disk blocks from each of the n disks. The blocks themselves are partitioned into symbols, and there is a fixed number of symbols for each disk in each stripe. We denote this quantity r. The stripes perform encoding and decoding as independent units in the disk system. Therefore, to alleviate hot spots that can occur because the coding disks may require more activity than the data disks, one can rotate the disks' identities on a stripe-by-stripe basis.

For the purpose of our analysis, we focus on a single stripe. There are k data disks labeled D0, . . . , Dk-1 and mcoding disks labeled C0, . . . , Cm-1. There are nr symbols in the stripe. We label the r symbols on data disk i as di,0, di,1, . . . , di,r-1 and on coding disk j as cj,0, cj,1, . . . , cj,r-1. We depict an example system in Figure 1. In this example, k = 6, m = 3 (and therefore n = 9) and r = 4.



Fig. 1: One stripe from an erasure coded storage system. The parameters are k = 6, m = 3 and r = 4.

Erasure codes are typically defined so that each symbol is a w-bit word, where w is typically small, often one. Then the coding words are defined as computations of the data words. Thus for example, suppose an erasure code were defined in Figure 1 for w = 1. Then each symbol in the stripe would be composed of one single bit. While that eases the definition of the erasure code, it does not map directly to a disk system. In reality, it makes sense for each symbol in a sealed block to be much larger in size, on the order of kilobytes or megabytes, and for each symbol to be partitioned into whit words, which are encoded and decoded in parallel.

3.1 Matrix-Vector Definition

All erasure codes may be expressed in terms of a matrix vector product. An example is pictured in Figure 2. This continues the example from Figure 1, where k = 6, m = 3 and r = 4; In this picture, the erasure code is defined precisely. This is a Cauchy Reed-Solomon code [6] optimized by the Jerasure library [38]. The word size, w equals one, so all symbols are treated as bits and arithmetic is composed solely of the XOR operation. The kr symbols of data are organized as a kr-element bit vector. They are multiplied by a nr×kr Generator matrix GT .1 The product is a vector, called the codeword, with nr elements. These are all of the symbols in the stripe. Each collection of r symbols in the vector is stored on a different disk in the system.

Since the the top kr rows of GT compose an identity matrix, the first kr symbols in the codeword contain the data. The remaining mr symbols are calculated from the data using the bottom mr rows of the Generator matrix. When up to m disks fail, the standard methodolgy for recovery is to select k surviving disks and create a decoding matrix B from the kr rows of the Generator matrix that correspond to them. The product of B-1 and the symbols in the k surviving disks yields the original data [6, 20, 33].

There are many MDS erasure codes that apply to storage systems. Reed-Solomon codes [40] are defined for all values of k and m. With a Reed-Solomon code, r = 1, and w must be such that $2w_n$ n. Generator matrices are constructed from a Vandermonde matrix so that any k × k subset of the Generator matrix is invertible. There is quite a bit of reference material on Reed-Solomon codes as they apply to storage systems [33, 36, 6, 41], plus numerous open-source Reed- Solomon coding libraries [42, 38, 30, 31].

Cauchy Reed-Solomon codes convert Reed-Solomon codes with r = 1 and w > 1 to a code where r = w and w = 1. In doing so, they remove the expensive multiplication of Galois Fields and replace it with additional XOR operations. There are an exponential number of ways to construct the Generator matrix of a Cauchy Reed-Solomon code. The Jerasure library attempts to construct a matrix with a minimal number of non-zero entries [38]. It is these matrices that we use in our examples with Cauchy Reed-Solomon codes.

For m = 2, otherwise known as RAID-6, there has been quite a bit of research on constructing codes where w = 1 and the CPU performance is optimized. EVENODD [3], RDP [11] and Blaum-Roth [5] codes all require r + 1 to be a prime number such that k _ r + 1 (EVENODD) or k _ r. The Liberation codes [34] require r to be a prime number and k _ r, and the Liber8tion code [35] is defined for r = 8 and k _ r. The latter three codes (Blaum-Roth, Liberation and Liber8tion) belong to a family of codes called Minimum Density codes, whose Generator matrices have a provably minimum number of ones.

Both EVENODD and RDP codes have been extrapolated to higher values of m [2, 4]. We call

these Generalized EVENODD and RDP. With m = 3, the same restrictions on r apply. For larger values of m, there are additional restrictions on r. The STAR code [24] is an instance of the generalized EVENODD codefor m = 3, where recovery is performed without using the Generator matrix.

All of the above codes have a convenient feature that disk C0 is constructed as the parity of the data disks, as in RAID-4/5. Thus, the r rows of the Generator matrix immediately below the identity portion are composed of k ($r \times r$) identity matrices. To be consistent with these RAID systems, we will refer to disk C0 as the "P drive."

4. Optimal Recovery of XOR-Based Erasure Codes

When a data disk fails in an erasure coded disk array, it is natural to reconstruct it simply using the P drive. Each failed symbol is equal to the XOR of corresponding symbols on each of the other data disks, and the parity symbol on the P disk. We call this methodology "Reading from the P drive." It requires k symbols to be read from disk for each decoded symbol.

Although it is straightforward both in concept and implementation, in many cases, reading from the P drive requires more I/O than is necessary. In particular, depending on the erasure code, there are savings that can be exploited when multiple symbols are recovered in the same stripe. This effect was first demonstrated by Xiang et al. in RDP systems in which one may reconstruct all the failed blocks in a stripe by reading 25 percent fewer symbols than reading from the P drive [51]. In this section, we approach the problem in general.



Fig. 2: The matrix-vector representation of an erasure code. The parameters are the same as Figure 1: k = 6, m = 3 and r = 4. Symbols are one bit (i.e. w = 1). This is a Cauchy Reed-Solomon code for these parameters.

4.1 Algorithm to Determine the Minimum

Number of Symbols for Recovery

We present an algorithm for recovering from a single disk failure in any XOR-based erasure code with a minimum number of symbols. The algorithm takes as input a Generator matrix whose symbols are single bits and the identity of a failed disk and outputs equations to decode each failed symbol. The inputs to the equations are the symbols that must be read from disk. The number of inputs is minimized.

The algorithm is computationally expensive—for the systems evaluated for this paper, each instantiation took from seconds to hours of compute-time. However, for any realistic storage system, the number of recovery scenarios is limited, so that the algorithm may be run ahead of time, and the results may be stored for when they are required by the system.

We explain the algorithm by using the erasure code of Figure 3 as an example. This small code, with k = m = r = 2, is not an MDS code; however its simplicity facilitates our explanation. We label the rows of GT as Ri, $0 \le i < nr$. Each row Ri corresponds to a data or coding symbol, and to simplify our presentation, we will refer to symbols using Ri rather than di,j or ci,j. Consider a set of symbols in the codeword whose corresponding rows in the Generator matrix sum to a vector of zeroes. One example is {R0,R2,R4}. We call such a set of

symbols a decoding equation, because the fact their rows sum to zero allows us to decode any one symbol in the set as long as the remaining symbols are not lost.



Fig. 3: An example erasure code to explain the algorithm to minimize the number of symbols required to recover from failures.

We can recover all the symbols in F by selecting one decoding equation ei from each set Ei, reading the nonfailed symbols in ei and then XORing them to produce the failed symbol. To minimize the number of symbols read, our goal is to select one equation ei from each Ei such that the number of symbols in the union of all ei is minimized.

For example, suppose that a disk fails, and both R0 and R1 are lost. A standard way to decode the failed bits is to read from the P drive and use coding symbols R4 and R5. In equation form, F ={R0,R1} e0 = {R0,R2,R4} and e1 = {R1,R3,R5}. Since e0 and e1 have distinct symbols, their union is composed of six symbols, which means that four must be read for recovery. However, if we instead use {R1,R2,R7} for e1, then (e0 [e1) has five symbols, meaning that only three are required for recovery.

Thus, our problem is as follows: Given |F| sets of decoding equations $E0,E1, \ldots E|F|-1$, we wish to select one equation from each set such that the size of the union of these equations is minimized. Unfortunately, this problem is NP-Hard in |F| and |Ei|.2 However, we can solve the problem for practical values of |F| and |Ei| (typically less than 8 and 25 respectively) by converting the equations into a directed, weighted graph and finding the shortest path through the graph. Given an instance of the problem, we convert it to a graph as follows. First, we represent each decoding equation in set form as an nr-element bit string. For example, {R0,R2,R4} is represented by 10101000.

Each node in the graph is also represented by an nr-element bit string. There is a starting node Z whose string is all zeroes. The remaining nodes are partitioned into |F| sets, labeled S0, S1, . . . S|F|-1. For each equation e0 2 E0, there is a node s0 2 S0 whose bit string equals e0's bit string. There is an edge from Z to each s0 whose weight is equal to the number of ones in s0's bit string.

For each node si \in Si, there is an edge that corresponds to each ei+1 \in Ei+1. This edge is to a node si+1 \in Si+1 whose bit string is equal to the bitwise OR of the bit strings of si and ei+1. The OR calculates the union of the equations leading up to si and ei+1. The weight of the edge is equal to the difference between the number of ones in the bit strings of si and si+1. The shortest path from Z to any node in S|F|-1 denotes the minimum number of elements required for recovery. If we annotate each edge with the decoding equation that creates it, then the shortest path contains the equations that are used for recovery.

To illustrate, suppose again that $F = \{R0, R1\}$, meaning f0 = R0 and f1 = R1. The decoding equations for E0 and E1 are denoted by ei,j where i is the index of the lost symbol in the set F and j is an index into the set Ei. E0 and E1 are enumerated below:

E ₀	E_1
$e_{0,0}=10101000$	e _{1,0} =01010100
$e_{0,1}=10010010$	$e_{1,1}=01101110$
$e_{0,2}=10011101$	e _{1,2} =01100001
$e_{0,3}=10100111$	e _{1,3} =01011011

These equations may be converted to the graph depicted in Figure 4, which has two shortest paths of length five: { $e_{0,0}$, $e_{1,2}$ } and { $e_{0,1}$, $e_{1,0}$ }. Both require three symbols for recovery: {R2,R4,R7} and {R3,R5,R6}. While the graph clearly contains an exponential number of nodes, one may program Dijkstra's algorithm to determine the shortest path and prune the graph drastically. For example, in Figure 5, the shortest path will be discovered before the the dotted edges and grayed nodes are considered by the algorithm. Therefore, they may be pruned.



Fig. 4: The graph that results when R0 and R1 are lost.

4.2 Algorithm for Reconstruction

When data disk i fails, the algorithm is applied for $F = \{di, 0, ..., di, r-1\}$. When coding disk j fails, $F = \{cj, 0, ..., cj, r-1\}$. If a storage system rotates the identities of the disks on a stripe-by-stripe basis, then the average number of symbols for all failed disks multiplied by the total number of stripes gives a measure of the symbols required to reconstruct a failed disk.

4.3 Algorithm for Degraded Reads

To take maximum advantage of parallel I/O, we assume that contiguous symbols in the file system are stored on different disks in the storage system. In

other words, if one is reading three symbols starting with symbol d0,0, then those three symbols are d0,0, d1,0 and d2,0, coming from three different disk drives.

To evaluate degraded reads, we assume that an application desires to read B symbols starting at symbol dx,y, and that data disk f has failed. We determine the penalty of the failure to be the number of symbols required to perform the read, minus B.

There are many cases that can arise from the differing values of B, f, x and y. To illustrate, first suppose that B < k (which is a partial read case) and that none of the symbols to be read reside on disk f. Then the failure does not impact the read operation — it takes exactly B symbols to complete the read, and the penalty is zero.

As a second case, consider when B = kr and dx, y = d0, 0. Then we are reading exactly one stripe in its entirety. In this case, we have to read the (k-1)r non-failed data symbols to fulfill the read request. Therefore, we may recover very easily from the P drive by reading all of its symbols and decoding. The read requires kr = B symbols. Once again, the penalty is zero.

However, consider the case when B = k, f = 0, and dx, y = d1, 0. Symbols d1, 0 through dk-1, 0 are non-failed and must be read. Symbol d0, 1 must also be read and it is failed. If we use the P drive to recover, then we need to read d1, 1 through dk-1, 0 and c0, 1. The total symbols read is 2k - 1: the failure has induced a penalty of k - 1 symbols.

In all of these cases, the degraded read is contained by one stripe. If the read spans two stripes, then we can calculate the penalty as the sum of the penalties of the read in each stripe. If the read spans more than two stripes, then we only need to calculate the penalties in the first and last stripe. This is because, as described above, whole-stripe degraded reads incur no penalty.

When we perform a degraded read within a stripe, we modify our algorithm slightly. For each non-failed data symbol that must be read, we set its bit in the state of the starting node Z to one. For example, in Figure 3, suppose we are performing a degraded read where B = 2, f = 0 and dx,y = d0,0. There is one failed bit: F = d0,0. Since d1,0 = R2 must be read, the starting state Z of the shortest path graph is labeled 00100000. The algorithm correctly identifies that only c0,0 needs to be read to recover d0,0 and complete the read.

5. Rotated Reed-Solomon Codes

Before performing analyses of failed disk reconstruction and degraded reads, we present two instances of a new erasure code, called the Rotated Reed-Solomon code. These codes have been designed to be MDS codes that optimize the performance of degraded reads for single disk failures. The general formulation and theoretical evaluation of these codes is beyond the scope of this paper; instead, we present instances for $m \in \{2, 3\}$.



Fig. 5: A Reed-Solomon code for k = 6 and m = 3. Symbols must be w-bit words such that $w \ge 4$, and arithmetic is over $GF(2^w)$.

The most intuitive way to present a Rotated Reed- Solomon code is as a modification to a standard Reed- Solomon code. We present such a code for m_3 in Equation 1. As with all Reed-Solomon codes, r = 1.

For
$$0 \le j < 3$$
, $C_{j,0} = \sum_{i=0}^{k-1} (2^j)^i d_{i,0}$ (1)

This is an MDS code so long as k, m, r and w adhere to some constraints, which we detail at the end of this section. This code is attractive because one may implement encoding with XOR and multiplication by two and four in GF(2w), which are all very fast operations. For example, the m = 2 version of this code lies at the heart of the Linux RAID-6 coding engine [1].

We present the code pictorally in Figure 5. A chain of circles denotes taking the XOR of di,0; a chain of triangles denotes taking the XOR of 2idi,0, and a chain of squares denotes taking the XOR of 4idi,0. To convert this code into a Rotated Reed-Solomon code, we allow r to take on any positive value, and define the coding symbols with Equation 2.

$$c_{j,b} = \sum_{i=0}^{\frac{kj}{m}-1} (2^j)^i d_{i,(b+1)\%r} + \sum_{i=\frac{kj}{m}}^{k-1} (2^j)^i d_{i,b}.$$
 (2)

Intuitively, the Rotated Reed-Solomon code converts the one-row code in Figure 6 into a multi-row code, and then the equations for coding disks 1 and 2 are split across adjacent rows. We draw the Rotated Reed- Solomon codes for k = 6 and $m = \{2, 3\}$ and r = 3 in Figures 6 and 7.



Fig. 6: A Rotated Reed-Solomon code for k = 6, m = 2 and r = 3.

These codes have been designed to improve the penalty of degraded reads. Consider a RAID-6 system that performs a degraded read of four symbols starting at d5,0 when disk 5 has failed. If we reconstruct from the P drive, we need to read d0,0 through d4,0 plus c0,0 to reconstruct d5,0. Then we read the non-failed symbols d0,1, d1,1 and d2,1. The penalty is 5 symbols. With Rotated Reed-Solomon coding, d5,0, d0,1, d1,1 and d2,1 all participate in the equation for c1,0. Therefore, by reading c1,0, d0,1, d1,1, d2,1, d3,0 and d4,0, one both decodes d5,0 and reads the symbols that were required to be read. The penalty is only two symbols.

With whole disk reconstruction, when r is an even number, one can reconstruct any failed data disk by reading $\frac{r}{2}\left(k+\frac{k}{m}\right)$ symbols. The process is exemplified for k = 6, m = 3 and r = 4 in Figure 8. The first data disk has failed, and the symbols required to reconstruct each of the failed symbols is darkened and annotated with the equation that is used for reconstruction. Each pair of reconstructed symbols in this example shares four data symbols for reconstruction. Thus, the whole reconstruction process requires a total of 16 symbols, as opposed to 24 when reading from the P Drive.



Fig. 7: A Rotated Reed-Solomon code for k = 6,m = 3 and r = 3.

The process is similar for the other data drives. Reconstructing failed coding drives, however does not have the same benefits. We are unaware at

present of how to reconstruct a coding drive with fewer than the maximum kr symbols.

As an aside, when more than one disk fails, Rotated Reed-Solomon codes may require much more computation to recover than other codes, due to the use of matrix inversion for recovery. We view this property as less important, since multiple disk failures are rare occurrences in practical storage systems, and computational overhead is less important than the I/O impact of recovery.



Fig. 8: Reconstructing disk 0 when it fails, using Rotated Reed-Solomon coding for k = 6, m = 3, r = 4.

5.1 MDS Constraints

The Rotated Reed-Solomon code specified above in Section 5 is not MDS in general. In other words, there are settings of k, m, w and r which cannot tolerate the failure of any m disks. Below, we detail ways to constrain these variables so that the Rotated Reed-Solomon code is MDS. Each of these settings has been verified by testing all combinations of m failures to make sure that they may be tolerated.

They cover a wide variety of system sizes, certainly much larger than those in use today.

The constraints are as follows: $m \in \{2, 3\}$ $k \le 36$, and $k + m \le 2w + 1$ $w \in \{4, 8, 16\}$ $r \in \{2, 4, 8, 16, 32\}$

Moreover, when w = 16, r may be any value less than or equal to 48, except 15, 30 and 45. It is a matter of future research to derive general-purpose MDS constructions of Rotated Reed-Solomon codes.

6. Conclusion

The paper provide guidance as to how to deploy erasure coding in the cloud file systems with respect to choosing a specific code and the size of sealed blocks . Cloud file systems distribute the coded blocks from each stripe (sealed block) on a different set of storage nodes. This strategy provides load balance and incremental scalability in the data

center. It also prevents correlated failures from resulting in data loss and mitigates the effect that any single failure has on a data set or application [15]. However, it does mean that each stripe is recovered independently from a different set of disks. To achieve good recovery performance when recovering independent stripes, codeword symbols need to be large enough to amortize disk seek overhead. Our results recommend a minimum symbol size of 4 MB and prefer 16 MB. This translates to a minimum sealed block size of 144 MB and preferred size of 576 MB for RDP and GenRDP, for example. Cloud file systems would benefit from increasing the sealed blocks to these size from the 64 MB default Increasing the symbol size has drawbacks as well. It increases memory consumption during recovery and increases the latency of degraded reads, because larger symbols need to recover more data.

Codes differ substantially in recovery performance, which demands a careful selection of code and parameters for cloud file systems. Optimally-sparse, Minimum- Density codes tend to perform best. The Liber8tion code and Generalized RDP are preferred for m = 2 and m = 3 respectiveley. Reed-Solomon codes will continue to be popular for their generality. For some Reed- Solomon codes, including rotated-RS codes, recovery performance may be improved by more than 20%. However, the number of symbols per disk (r) has significant impact. For k = 6 data disks, the best values are r = 7for m = 2 and r = 4 for m = 3.

Several open problems remain with respect to optimal recovery and degraded reads. While our algorithm an determine the minimum number of symbols needed for recovery for any given code, it remains unknown how to generate recovery-optimal erasure codes. We are pursuing this problem both analytically and through a programmatic search of feasible generator matrixes. Rotated RS codes are a first result in lowering degraded read costs. Lower bounds for the number of symbols needed for degraded reads have not been determined.

We have restricted our treatment to MDS codes, since they are used almost exclusively in practice because of their optimal storage efficiency. However, some codes with decreased storage efficiency have much lower recovery costs than MDS [27, 18, 28, 23, 19]. Exploring non-MDS codes more thoroughly will help guide those building cloud systems in the tradeoffs between storage efficiency, fault-tolerance, and performance.

References

[1] H. P. Anvin. The mathematics of RAID-6. *http://kernel.org/pub/linux/kernel/* people/hpa/raid6.pdf, 2009.

- M. Blaum. A family of MDS array codes [2] with minimal number of encoding operations. In IEEE International Symposium Information Theory, on September 2006.
- [3] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computing*,44(2):192–202, February 1995.
- [4] M. Blaum, J. Bruck, and A. Vardy. MDS array codes with independent parity symbols. *IEEE Transactions on Information Theory*, 42(2):529–542, February 1996.
- [5] M. Blaum and R. M. Roth. On lowest density MDS codes. *IEEE Transactions on Information Theory*, 45(1):46–59, January 1999.
- [6] Blomer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman. An XOR-based erasure-resilient coding scheme. *Technical Report TR-95-048, International Computer Science Institute*, August 1995.
- [7] V. Bohossian and J. Bruck. Shortening array codes and the perfect 1-Factorization conjecture. In *IEEE International Symposium on Information Theory*, pages 2799–2803, 2006.
- [8] D. Borthakur. The Hadoop distributed file system: Architecture and design. http://hadoop.apache.org/ common/docs/current/hdfs-design.html, 2009.
- [9] E. Brewer. Lessons from giant-scale services. *Internet Computing*, 5(4), 2001.
- [10] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. Fahim ul Haq, M. Ikram ul Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. Windows Azure storage: A highly available cloud storage service with strong consistency. In Symposium on Operating Systems Principles, 2011.
- [11] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row diagonal parity for double disk failure correction. *In Conference on File and Storage Technologies*, March 2004.
- [12] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Trans. Inf. Theor.*, 56(9):4539–4551, September 2010.

- [13] A. L. Drapeau et al. RAID-II: A highbandwidth network file server. *In International Symposium on Computer Architecture*, 1994.
- [14] B. Fan, W Tanisiriroj, L. Xiao, and G. Gibson. DiskReduce: RAID for dataintensive scalable computing. In Parallel Data Storage Workshop, 2008.
- [15] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed file systems. In Operating Systems Design and Implementation, 2010.
- [16] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In ACM SOSP, 2003.
- [17] K. Greenan, E. Miller, and T. J. Schwartz. Optimizing Galois Field arithmetic for diverse processor architectures and applications. In Modeling, Analysis and Simulation of Computer and Telecommunication Systems, September 2008.
- [18] K. M. Greenan, X. Li, and J. J. Wylie. Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. *Mass Storage Systems and Technologies*, 2010.
- [19] J. L. Hafner. Weaver codes: Highly fault tolerant erasure codes for storage systems. In Conference on File and Storage Technologies, 2005.
- [20] J. L. Hafner, V. Deenadhayalan, K. K. Rao, and J. A. Tomlin. Matrix methods for lost data reconstruction in erasure codes. *In Conference on File and Storage Technologies*, 2005.
- [21] M. Holland and G. A. Gibson. Parity declustering for continuous operation in redundant disk arrays. *In Architectural Support for Programming Languages and Operating Systems*. ACM, 1992.
- [22] R. Y. Hou, J. Menon, and Y. N. Patt. Balancing I/O response time and disk rebuild time in a RAID5 disk array. In Hawai'i International Conference on System Sciences, 1993.
- [23] C. Huang, M. Chen, and J. Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *Network Computing and Applications*, 2007.
- [24] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers*, 57(7):889–901, July 2008.
- [25] H. Jin, J. Zhang, and K. Hwang. A raid reconfiguration scheme for gracefully

degraded operations. EuroMicro Conference on Parallel, Distributed, and Network-Based Processing, 0:66, 1999.

- [26] D. Kenchammana-Hosekote, D. He, and J. L. Hafner. REO: A generic RAID engine and optimizer. *In Conference on File and Storage Technologies*, pages 261–276, 2007.
- [27] O. Khan, R. Burns, J. S. Plank, and C. Huang. In search of I/O-optimal recovery from disk failures. *In Workshop on Hot Topics in Storage Systems*, 2011.
- [28] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In 29th Annual ACM Symposium on Theory of Computing, pages 150–159, El Paso, TX, 1997. ACM.
- [29] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes, Part I.* North-Holland Publishing Company, 1977.
 13
- [30] Onion Networks. Java FEC Library v1.0.3. Open source code distribution: http://onionnetwor ks.com/ fec/javadoc/, 2001.
- [31] A. Partow. Schifra Reed-Solomon ECC Library. Open source code distribution: http://www.schifra.com/downloads.html, 2000-2007.
- [32] W. W. Peterson and E. J. Weldon, Jr. *Error-Correcting Codes, Second Edition.* The MIT Press, 1972.
- [33] J. S. Plank. A tutorial on Reed-Solomon coding for faulttolerance in RAID-like systems. *Software—Practice & Experience*, 27(9):995–1012, 1997.
- [34] J. S. Plank. The RAID-6 Liberation codes. In Conference on File and Storage Technologies, 2008.
- [35] J. S. Plank. *The RAID-6 Liber8Tion code*. *Int. J. High Perform. Comput. Appl.*, 23:242–251, August 2009.
- [36] J. S. Plank and Y. Ding. Note: Correction to the 1997 tutorial on Reed-Solomon coding. Software – Practice & Experience, 35(2):189–194, February 2005.
- [37] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z.Wilcox- OHearn. A performance evaluation and examination of open-source erasure coding libraries for storage. *In Conference on File and Storage Technologies*, 2009.
- [38] J. S. Plank, S. Simmerman, and C. D. Schuman. Jerasure: A library in C/C++ facilitating erasure coding for storage applications Version 1.2. Technical Report CS-08-627, University of Tennessee, August 2008.
- [39] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramachandran. Explicit construction of

optimal exact regenerating codes for distributed storage. *In Communication, Control, and Computing,* 2009.

- [40] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, 1960.
- [41] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM SIGCOMM Computer Communication Review*, 27(2):24–36, 1997.
- [42] L. Rizzo. Erasure codes based on Vandermonde matrices. Gzipped tar file posted at http://planete-bcast.inrialpes.fr/ rubrique.php3?id rubrique=10, 1998.
- [43] R. Rodrigues and B. Liskov. High availability in DHTS: Erasure coding vs. replication. In Workshop on Peer-to- Peer Systems, 2005.
- [44] B. Schroeder and G. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 mean to you? *In Conference on File and Storage Technologies*, 2007.
- [45] M. Sivathanu, V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving storage system availability with D-GRAID. In Conference on File and Storage Technologies, 2004.
- [46] Apache Software. *Pigmix. https://cwiki.apache.org/confluence/display/PIG/ PigMix,* 2011.
- [47] A. Thusoo, D. Borthakur, R. Murthy, Z. Shao, N. Jain, H. Liu, S. Anthony, and J. S. Sarma. *Data warehousing and analytics infrastructure at Facebook*. In SIGMOD, 2010.
- [48] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song. PRO: multi-threaded popularity-based а reconstruction optimization for RAIDstructured systems. storage In File Conference on and Storage Technologies, 2007.
- [49] Z. Wang, A. G. Dimakis, and J. Bruck. *Rebuilding for array codes in distributed storage systems.* CoRR, abs/1009.3291, 2010.
- [50] H. Weatherspoon and J. Kubiatowicz. *Erasure coding vs.replication: A quantitative comparison. In Workshop on Peer-to-Peer Systems,* 2002.
- [51] L. Xiang, Y. Xu, J. C. S. Lui, and Q. Chang. Optimal recovery of single disk failure in RDP code storage systems. *In ACM SIGMETRICS*, 2010.
- [52] L. Xu and J. Bruck. X-Code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45(1):272–276, January 1999.

[53] Z. Zhang, A. Deshpande, X. Ma, E. Thereska, and D. Narayanan. *Does erasure* coding have a role to play in my data center? Microsoft Technical Report MSRTR- 2010-52, 2010.